On Pruning for Top-K Ranking in Uncertain Databases

Chonghai Wang, Li Yan Yuan, Jia-Huai You, Osmar R. Zaiane
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
E-mail: {chonghai,yuan,you,zaiane}@cs.ualberta.ca

Jian Pei
School of Computing Science
Simon Fraser University
Burnaby, BC Canada V5A 1S6
E-mail: jpei@cs.sfu.ca

ABSTRACT

Top-k ranking for an uncertain database is to rank tuples in it so that the best k of them can be determined. The problem has been formalized under the unified approach based on parameterized ranking functions (PRFs) and the possible world semantics. Given a PRF, one can always compute the ranking function values of all the tuples to determine the top-k tuples, which is a formidable task for large databases. In this paper, we present a general approach to pruning for the framework based on PRFs. We show a mathematical manipulation of possible worlds which reveals key insights in the part of computation that may be pruned and how to achieve it in a systematic fashion. This leads to concrete pruning methods for a wide range of ranking functions. We show experimentally the effectiveness of our approach.

1. INTRODUCTION

Uncertain databases, also called probabilistic databases, are proposed to deal with uncertainty in a variety of application domains, such as in sensor network and data cleaning [3, 9]. Typically, an uncertain database consists of a set of tuples each of which comes with a numeric value representing the *score* of the tuple and a *membership probability*. Uncertainty may be due to incompleteness of data, limitation of equipment, or loss in data transfer, etc. Different uncertain data models have been proposed for uncertain databases [2, 5, 8], some adopting the possible world semantics [5, 8]. As a reasonable approximation to the uncertain nature of data, the data model based on *x-tuples* is often adopted in the study of uncertain databases.

The interaction of the information associated with a tuple – the score representing the importance of the tuple, and the likelihood of a tuple representing the true information, has made top-k ranking an intriguing issue, with different definitions of top-k tuples proposed [1, 4, 10, 11]. In [7] however, a general framework of top-k ranking, based on *parameterized ranking function* (PRF) and the *possible world semantics* is formulated, which generalizes many previously proposed ranking functions. Two classes of PRFs, under the names PRF^{ω} and PRF^{e}, are proposed, and their computational properties are studied under the *probabilistic and/xor tree model* [6], of which x-tuples is a special (but a dominating) case. As commented in [7], before PRF all the other ranking functions for uncertain databases do not consider more complex correlations than the one of x-tuple.

Depending on the underlying ranking function, a top-k algorithm may run quadratic time or higher, which is considered too expensive for large databases. There are generally two ways to improve the performance: design approximation algorithms or avoid the computation of the ranking function values of some tuples that are guaranteed not to be in top-k. The latter is called *pruning*, which is the focus of this paper. Some pruning techniques have been proposed in the past, but they are formulated only for some fixed semantics (e.g., [1, 4]). As a result, it is often not clear whether they are applicable to different ranking functions.

In this paper we present such a study for the framework based on PRF^{ω} and PRF^{e}. We show a partitioning of possible worlds which leads to insights in what constitutes an upper bound and the mathematics of deriving it. This leads to concrete pruning methods for a wide range of ranking functions. Our experiments on synthetic data as well as real data show orders of magnitude speedup over algorithms without pruning, and substantial improvement over the existing pruning methods.

The next section reviews some definitions, followed by a general upper bound method in Section 3, which leads to concrete computational methods in Section 4. Section 5 studies pruning for PRF^{e}. Section 6 presents experimental results, and Section 7 concludes the paper.

2. RANKING IN UNCERTAIN DATABASES

Under the x-tuple model, an *uncertain database* (or an *uncertain table*) T contains a set of tuples, each t of which is associated with a membership probability, denoted by $Pr(t)$, such that $Pr(t) > 0$. Each tuple t is associated with a score, $score(t)$, which is determined by a scoring function: $T \rightarrow \Re$. A *generation rule* r of an uncertain database T is an exclusive relation of one or more tuples in T, written as $r = t_1 \oplus t_2 \oplus ... \oplus t_e$ (we will also use r to denote the set of these tuples), and the sum of the membership probabilities of the involved tuples, denoted by $Pr(r)$, satisfies $Pr(r) \leq 1$. We assume each tuple in T appears in exactly one generation rule. When $Pr(r) < 1$, to represent the probability of the missing information, we define $Pr(\bar{r}) = 1 - Pr(r)$. A tuple involved in a single-tuple generation rule is called an *independent tuple*.

A *possible world* W is a set of tuples in an uncertain database T, such that for each generation rule r on T, W consists of exactly one tuple in r if $Pr(r) = 1$, and zero or one tuple in r if $Pr(r) < 1$. The *probability* of W, denoted by $Pr(W)$, is the product of the membership probabilities of all the tuples in W and all of $Pr(\bar{r})$, for each r where W contains no tuples from it. It is clear that $\sum_{W \in \Lambda} Pr(W) = 1$, where Λ is the set of all possible worlds for T.

Example 2.1. *Figure 1 shows an example of an uncertain table, where the score of a tuple is its speed. The generation rules here are $t_2 \oplus t_3, t_4 \oplus t_5, t_6 \oplus t_7$, and t_1.*

	Time	Radar	Model	Plate No	Speed	Prob
t_1	11:45	L1	Honda	X-123	120	1.0
t_2	11:50	L2	Toyota	Y-245	130	0.7
t_3	11:35	L3	Toyota	Y-245	95	0.3
t_4	12:10	L4	Mazda	W-541	90	0.4
t_5	12:25	L5	Mazda	W-541	110	0.6
t_6	12:15	L6	Chevy	L-105	105	0.5
t_7	12:20	L7	Chevy	L-105	85	0.4

Figure 1: A sample uncertain database

Before [7], top-k tuples are typically defined by a fixed ranking function that combines the effects of scores and probabilities. In [7], the authors show the conflicting behavior of some of these ranking functions and argue that a single specific ranking function may not be appropriate to rank different uncertain databases in practice. They then propose a framework based on parameterized ranking functions, which covers a large number of different definitions of top-k tuples with different parameter values.

Formally, the *parameterized ranking function* for a given tuple t is defined as: $\Upsilon(t) = \sum_{W \in PW(t)} \omega(t, \beta_W(t)) \times Pr(W)$, where $PW(t)$ is the set of all the possible worlds containing t, $\beta_W(t)$ is the *position* of t in the possible world W (according to *score*(t)), and $\omega(t, i)$ is a *weight function*: $T \times N \rightarrow C$ (C is the set of complex numbers). We call $\Upsilon(t)$ *the PRF^ω value of t. A top-k query returns the k tuples with highest $|\Upsilon(t)|$ values.* In this paper, we restrict $\omega(t, i)$ to $\omega(i)$, which means the weight function is independent of t, and the values of $\omega(i)$ to real numbers. Further, we assume $\omega(i)$ is *monotonically non-increasing*. This means $\omega(i) \geq \omega(i+1)$, for all i with $1 \leq i \leq n-1$, assuming n tuples. This assumption is reasonable, since in normal cases a higher position is at least as desirable as those behind it and thus should be given a higher weight. Algorithms are proposed in [7] to compute the PRF^ω values (see Appendix A).

Example 2.2. *In Example 2.1, suppose the weight function is $\omega(i) = 5 - i$. Consider t_3. It is included in 6 possible worlds: $PW_1 = \{t_1, t_3, t_4, t_6\}$, $PW_2 = \{t_1, t_3, t_4, t_7\}$, $PW_3 = \{t_1, t_3, t_4\}$, $PW_4 = \{t_1, t_3, t_5, t_6\}$, $PW_5 = \{t_1, t_3, t_5, t_7\}$, and $PW_6 = \{t_1, t_3, t_5\}$. We calculate $Pr(PW_1) = 0.06$, $Pr(PW_2) = 0.048$, $Pr(PW_3) = 0.012$, $Pr(PW_4) = 0.09$, $Pr(PW_5) = 0.072$, and $Pr(PW_6) = 0.018$. Clearly, $\Upsilon(t_3) = 2 \times 0.06 + 3 \times 0.048 + 3 \times 0.012 + 1 \times 0.09 + 2 \times 0.072 + 2 \times 0.018 = 0.48$.*

In [7], a special class of PRF^ω functions, named PRF^e, is proposed. A PRF^e function requires the weight function to be $\omega(i) = \alpha^i$, where α is a constant real number. Here we assume $0 < \alpha < 1$. We make this assumption because only in this case, $\omega(i) = \alpha^i$ is monotonically non-increasing (we will not consider the trivial cases where $\alpha = 0$ or 1).

Except for the early proposal of U-Topk [10], more recent definitions of top-k tuples in uncertain databases are all closely related to PRF. For example, Expected Rank [1] under the x-tuple model can be computed from PRF [7]; U-kRanks [10] belongs to PRF, but the weight function is not monotonically non-increasing so our pruning does not apply; PT-k query [4] belongs to PRF where the weight function is monotonically non-increasing, so our pruning does apply.

2.1 Outline of computing top-k tuples

Given an algorithm for computing top-k tuples, we can combine it with our pruning methods. Below, we outline this process for the algorithms given in [7] for PRF^ω, where the tuples of an uncertain database are sorted in a descending order based on their scores, and are retrieved one by one.

In the combined algorithm, we maintain a heap L_k of the k tuples with the highest PRF^ω values retrieved so far in the descending order of their PRF^ω values. For the first k tuples, they are stored in L_k along with their computed PRF^ω values. For a subsequently retrieved tuple, its PRF^ω value may or may not be computed, depending on the computed upper bound of its PRF^ω value.

We maintain a special tuple, called t_{lowest}, from the retrieved tuples (the details on how to choose it will be given later), which will be used in computing the upper bound of the next retrieved tuple. Thus, after the first k tuples, we retrieve a tuple t_{new} and compute its upper bound, say u. If for any t in L_k, $u \leq \Upsilon(t)$, then it is guaranteed that t_{new} is not a top-k tuple, hence $\Upsilon(t_{new})$ is not computed. Otherwise, $\Upsilon(t_{new})$ is computed and t_{lowest} updated if necessary. If $\Upsilon(t_{new})$ is higher than the lowest PRF^ω value in L_k, we replace a tuple in L_k with the lowest PRF^ω value with t_{new}. After all the tuples are retrieved, L_k holds the top-k tuples.

The most important missing detail in this outline is how to compute an upper bound of $\Upsilon(t_{new})$. Related questions include: (i) what could be an upper bound of $\Upsilon(t_{new})$, (ii) in what sense it is a tight upper bound, and (iii) is there a simple way to determine such an upper bound for ranking functions of PRF^e? These questions will be answered next.

3. A GENERAL UPPER BOUND METHOD

We present a general approach to generating an upper bound of $\Upsilon(t)$, for a given tuple t. For simplicity, we just call it *an upper bound of* t. A key finding is a new representation of PRF^ω values, followed by a general upper bound method.

Throughout this section, given an uncertain database T, we consider a set of q tuples $Q = \{t_1, t_2, ..., t_q\}$. As we are interested in the upper bound of a tuple $t \in Q$ and in our method we will use at least one another tuple as a reference, we assume $q \geq 2$. Given Q, there is a set of generation rules $R = \{r_1, r_2, ..., r_l\}$ associated with Q, i.e., every tuple in Q is in some generation rule in R and every $r_i \in R$ contains at least one tuple in Q. Clearly, $l \leq q$.

For any $t \in Q$, our interest is to find an upper bound of it. For this, we want to find some real numbers c_i such that

$$\sum_{i=1}^{q} c_i \Upsilon(t_i) \geq 0 \tag{1}$$

Note that one of the $t_i \in Q$ is t. Let us denote its coefficient above by c. If $c < 0$, (1) can be transformed to

$$\Upsilon(t) \leq \sum_{t_i \in Q, t_i \neq t} -\frac{c_i}{c} \Upsilon(t_i) \tag{2}$$

That is, the value of $\Upsilon(t)$ cannot be higher than the right hand side of (2), which is thus an upper bound of t. The lower such an upper bound, the better (tighter). But (2) does not tell us how to compute such an upper bound, for two reasons: (i) we may not have computed all of the $\Upsilon(t_i)$ values of the other tuples in Q, and (ii) we were not told how to choose c_i so that inequation (2) is guaranteed to hold *for any given q tuples*. The theoretical development of this section aims at the methods that resolve these questions.

3.1 Representation of PRF^ω values

For each generation rule $r_i \in T$, if $Pr(r_i) < 1$, for technical convenience, we create a tuple $\oslash_i$, called *the virtual tuple* of r_i. Virtual tuples do not participate in ranking, hence their scores are irrelevant. Thus we assume a virtual tuple has no score. But a virtual tuple has a probability, $Pr(\oslash_i) = 1 - Pr(r_i)$. To distinguish, we call the other tuples of this generation rule the *real tuples* of the generation rule. A virtual tuple has the same exclusive relation with other tuples of the same generation rule. Below, a *tuple set* may contain virtual tuples if not said otherwise.

Given a tuple $t_i \in Q$ (note that Q contains only real tuples), its PRF^ω value $\Upsilon(t_i)$ is defined in Section 2 as: $\Upsilon(t_i) = \sum_{W \in PW(t_i)} \omega(\beta_W(t_i)) \times Pr(W)$. Here we derive a new representation. The idea is to divide all possible worlds containing t_i into l groups, such that $\Upsilon(t_i)$ can be expressed by the sum of the *part PRF^ω values* of t_i in each group.

Suppose, among l generation rules $R = \{r_1, ..., r_l\}$, $t_i \in r_d$, for some $r_d \in R$. Consider a tuple set η of size l, such that $t_i \in \eta$ and each tuple in η is from a distinct generation rule in R (thus all tuples in η are from different generation rules). Let us write it in the form

$$\{t_{s_1}, t_{s_2}, ..., t_{s_{d-1}}, t_i, t_{s_{d+1}}, ..., t_{s_l}\}$$

where $t_{s_j} \in r_j$. Let us denote by Δ_i the set of all such tuple sets. A property of the tuple sets in Δ_i is that they are *symmetric* to each other, which means, assuming $\eta_1, \eta_2 \in \Delta_i$, for every possible world W_1 such that $\eta_1 \subset W_1$, there exists a possible world $W_2 = (W_1 - \eta_1) \cup \eta_2$, and vice versa.

We now divide Δ_i into l sets S_{ij}, for $0 \leq j \leq l-1$:

$$S_{ij} = \{S \in \Delta_i \mid \text{ there are exactly } j \text{ real tuples in } S \text{ s.t. for each such tuple } t, score(t) > score(t_i)\}$$

Let $\eta \in S_{ij}$, and $PW(\eta)$ be the set of all possible worlds containing all the tuples in η. We define $\Upsilon_\eta(t_i)$ as:

$$\Upsilon_\eta(t_i) = \sum_{W \in PW(\eta)} \omega(\beta_W(t_i)) \times Pr(W) \tag{3}$$

Let $Pr(\eta)$ be the sum of the probabilities of the possible worlds in $PW(\eta)$. We then can show

THEOREM 3.1. *For any two tuple sets $\eta_1, \eta_2 \in S_{ij}$, we have $\frac{\Upsilon_{\eta_1}(t_i)}{Pr(\eta_1)} = \frac{\Upsilon_{\eta_2}(t_i)}{Pr(\eta_2)}$.*

The theorem says that the ratio between $\Upsilon_\eta(t_i)$ and $Pr(\eta)$ is the same for all tuple sets $\eta \in S_{ij}$. As this result is critical in our theory of pruning, let us give it a special notation: Given a non-empty S_{ij} and $\eta \in S_{ij}$, define *the PRF^ω value ratio of S_{ij}*, denoted U_{ij}, as

$$U_{ij} = \frac{\Upsilon_\eta(t_i)}{Pr(\eta)} \tag{4}$$

When S_{ij} is empty, it is meaningless to define U_{ij}. For technical convenience, we can define U_{ij} to be a real number for an empty S_{ij} (we leave the details to Appendix B).

The tuple set η in (3) is just one in S_{ij}. We are interested in all η in S_{ij}. Let $PW(S_{ij}) = \cup_{\eta \in S_{ij}} PW(\eta)$, and define

$$\Upsilon_{S_{ij}}(t_i) = \sum_{\eta \in S_{ij}} \Upsilon_\eta(t_i)$$

Note that $\Upsilon_{S_{ij}}(t_i)$ is the part of the PRF^ω value obtained from the possible worlds in $PW(S_{ij})$.

Let us have the notation: $Pr(S_{ij}) = \sum_{\eta \in S_{ij}} Pr(\eta)$. Note that $Pr(S_{ij}) = 0$ when S_{ij} is empty. It is easy to check that $\Upsilon(t_i) = \sum_{j=0}^{l-1} \Upsilon_{S_{ij}}(t_i) = \sum_{j=0}^{l-1} \sum_{\eta \in S_{ij}} \Upsilon_\eta(t_i)$
$= \sum_{j=0}^{l-1} \sum_{\eta \in S_{ij}} U_{ij} \times Pr(\eta)$ *by* (4)
$= \sum_{j=0}^{l-1} U_{ij} \times Pr(S_{ij})$.

Thus, we have arrived at a new representation of $\Upsilon(t_i)$:

$$\Upsilon(t_i) = \sum_{j=0}^{l-1} U_{ij} \times Pr(S_{ij}) \tag{5}$$

Equation (5) is quite intuitive: for each j, $U_{ij} \times Pr(S_{ij})$ is the part of the PRF^ω value of t_i obtained from the possible worlds in $PW(S_{ij})$; then the sum of all these PRF^ω values of t_i is $\Upsilon(t_i)$.

Here, $\Upsilon(t_i)$ is expressed in terms of U_{ij} and $Pr(S_{ij})$. We will compute the latter but not the former (if we do compute both, we then have computed $\Upsilon(t_i)$ - there is no pruning).

Notice that

$$Pr(t_i) = \sum_{j=0}^{l-1} Pr(S_{ij}) \tag{6}$$

since both sides equal the sum of the probabilities of all the possible worlds containing t_i. This equation will be referenced later in this paper.

For the computation of $Pr(S_{ij})$, we can show that, for each tuple t_i, we need $O(l^2 + l\tau)$ time to compute $Pr(S_{ij})$, where τ is the maximum number of real tuples in a generation rule. Hence for q tuples, the complexity is $O(ql^2 + ql\tau)$. In real applications, τ is usually a very small number compared with the number of tuples in an uncertain database. The details can be found in Appendix C.

As a summary, we provide the following table of symbols.

Q	$\{t_1, ..., t_q\}$ $(q \geq 2)$, the given set of real tuples
R	$\{r_1, ..., r_l\}$ $(l \leq q)$, the set of relevant gen. rules
Δ_i	the set of tuple sets of size l s.t. $\forall \eta \in \Delta_i$, $t_i \in \eta$ and tuples in η are from distinct gen. rules in R
S_{ij}	$S_{ij} \subseteq \Delta_i$ s.t. $\forall \eta \in S_{ij}$, there are exactly j tuples with a score higher than $score(t_i)$
U_{ij}	the PRF^ω value ratio of S_{ij}
$PW(t)$	the set of possible worlds (PWs) that contain t
$PW(\eta)$	the set of PWs that contain all tuples in η
$PW(S_{ij})$	the union of $PW(\eta)$, for each $\eta \in S_{ij}$
$Pr(S_{ij})$	the sum of the prob's of all PWs in $PW(S_{ij})$

We now show a key insight in our theory of pruning.

THEOREM 3.2. *For any S_{ij_1}, S_{ij_2}, S_{i_1j}, and S_{i_2j}, where $(1 \le i, i_1, i_2 \le q)$ and $(0 \le j, j_1, j_2 \le l-1)$, we have*

(i) if $j_1 \le j_2$ then $U_{ij_1} \ge U_{ij_2}$, and

(ii) if $score(t_{i_1}) \ge score(t_{i_2})$ then $U_{i_1j} \ge U_{i_2j}$.

For (i) for example, for a tuple $t_i \in \{t_1, ..., t_q\}$, the larger the j value, the lower the U_{ij}, since higher j means more tuples are "ahead" in a possible world containing η.

3.2 A general method to determine upper bounds

Recall that, given a set of real tuples $Q = \{t_1, ..., t_q\}$, our goal is to compute an upper bound of a tuple $t \in Q$, expressed in the form of (2). In this section we present a generic method, which is independent of the number of tuples in Q, as long as there are at least two.

We know that for each tuple $t_i \in Q$, its PRF^ω value can be expressed in the form of (5). For this equation, we can multiply both sides with a constant c_i to get

$$c_i \Upsilon(t_i) = c_i \sum_{j=0}^{l-1} U_{ij} \times Pr(S_{ij})$$

For each tuple in Q we have such an equation, so we have q equations. Let us add them together to get

$$\sum_{i=1}^{q} c_i \Upsilon(t_i) = \sum_{i=1}^{q} \sum_{j=0}^{l-1} c_i \times U_{ij} \times Pr(S_{ij}) \qquad (7)$$

Recall that to get (2), all we need is to establish inequality (1) (of course, with the assumption $c < 0$), which can be obtained from (7) if we make its right hand side non-negative. This is our focus in the following exploration.

Since $Pr(S_{ij})$ in (7) is to be computed, we are left with U_{ij} and a choice of the values of c_i. As we want to avoid the computation of U_{ij}, we will utilize the relation developed in Theorem 3.2. The idea is to transform the right hand side of (7) to a form in terms of U_{ij}, which is guaranteed to be non-negative without actually computing U_{ij}. In the right hand side of (7), each U_{ij} has a coefficient which could be positive or negative. If we can transform this expression to a summation of m ($m \ge 1$) terms, each consisting of a large U_{ij} minus a small $U_{i'j'}$ with a positive coefficient, then clearly this expression must be non-negative. Let us write this summation as

$$\sum_{k=1}^{m} a_k (U_{i_kj_k} - U_{i'_kj'_k}) \qquad (8)$$

where $a_k > 0$ is a real number, and each term involves a pair of distinct $U_{i_kj_k}$ and $U_{i'_kj'_k}$ such that $U_{i_kj_k} \ge U_{i'_kj'_k}$, where $1 \le i_k, i'_k \le q$, $0 \le j_k, j'_k \le l-1$.

EXAMPLE 3.3. *Assume two tuples t_1 and t_2 with $score(t_1) \ge score(t_2)$, and we are interested in an upper bound of t_2. Suppose the PRF^ω values of the two tuples, expressed in form (5), are*

$$\Upsilon(t_1) = 0.03U_{10} + 0.06U_{11} \quad \Upsilon(t_2) = 0.2U_{20} + 0.7U_{21}$$

From Theorem 3.2, we know that

$$U_{10} \ge U_{11},\ U_{20} \ge U_{21},\ U_{10} \ge U_{20},\ U_{11} \ge U_{21}$$

If we set $c_1 = 1$ and $c_2 = -0.1$, we get the expression in (7)

$$\Upsilon(t_1) - 0.1\Upsilon(t_2) = 0.03U_{10} + 0.06U_{11} - 0.02U_{20} - 0.07U_{21}$$

Now let us transform the right hand side of the above as

$$\begin{aligned}&0.03U_{10} + 0.06U_{11} - 0.02U_{20} - 0.07U_{21}\\ &= 0.02(U_{10} - U_{20}) + 0.06(U_{11} - U_{21}) + 0.01(U_{10} - U_{21})\end{aligned}$$

The expression at the right hand side of equality above is in the form (8), which involves three pairs of U_{ij} with positive coefficients. Clearly, the value of this expression is non-negative, which guarantees $\Upsilon(t_1) - 0.1\Upsilon(t_2) \ge 0$. So an upper bound of t_2 is obtained by $\Upsilon(t_2) \le 10\Upsilon(t_1)$.

In general, there is no guarantee that there exist an assignment of c_i so that a transformation to (8) is possible. In this case we can relax the condition by allowing an extra expression, as in

$$\sum_{k=1}^{m_1} a_k (U_{i_kj_k} - U_{i'_kj'_k}) + \sum_{k'=1}^{m_2} b_{k'} U_{i_{k'}j_{k'}} \qquad (9)$$

where $m_1 \ge 0$ and $m_2 \ge 1$. The first summation is similar to (8). The second involves a subset of PRF^ω value ratios, for $1 \le i_{k'} \le q$ and $0 \le j_{k'} \le l-1$, with coefficients $b_{k'} > 0$. If $\omega(\beta(t)) \ge 0$ for any position $\beta(t)$ (i.e., the weight function ω is *non-negative*), then according to (3) and (4), all U_{ij} must be non-negative. Then (9) must be non-negative. It follows that the right hand side of (7) must be non-negative.

EXAMPLE 3.4. *Assume two tuples t_1 and t_2 with $score(t_1) \ge score(t_2)$, and suppose $\Upsilon(t_1) = 0.02U_{10} + 0.08U_{11}$ and $\Upsilon(t_2) = 0.2U_{20} + 0.6U_{21}$. Let c_1 and c_2 be the coefficients of t_1 and t_2 respectively, as in equation (7). Assume the weight function is non-negative. From Theorem 3.2, we know that $U_{10} \ge U_{11}, U_{20} \ge U_{21}, U_{10} \ge U_{20}$, and $U_{11} \ge U_{21}$. It can be shown that there does not exist an assignment that leads to a transformation to (8). However, a transformation to (9) is possible. If we set $c_1 = 1$ and $c_2 = -0.1$, we will get*

$$\begin{aligned}&\Upsilon(t_1) - 0.1\Upsilon(t_2)\\ &= 0.02U_{10} + 0.08U_{11} - 0.02U_{20} - 0.06U_{21}\\ &= 0.02(U_{10} - U_{20}) + 0.06(U_{11} - U_{21}) + 0.02U_{11}\end{aligned}$$

The second line above is in the form of the right hand side of (7), and the last expression is (9), whereas the term $0.02U_{11}$ corresponds to the second summation. Thus we conclude $\Upsilon(t_1) - 0.1\Upsilon(t_2) \ge 0$.

THEOREM 3.5. *Let $Q = \{t_1, ..., t_q\}$. Assume $t \in Q$ and there exists a tuple $s \in Q$ such that $s \ne t$ and $score(s) \ge score(t)$. Then, there exists at least one assignment θ of c_i such that the right hand side of (7) can be transformed to an expression in the form of (8), and if not, to an expression in the form of (9).*

In the following, given any q (real) tuples $Q = \{t_1, ..., t_q\}$, we say that *an assignment θ (of c_i w.r.t. Q) induces an upper bound of t*, where $t \in Q$, if the right hand side of (7), with θ substituted, can be transformed to an expression either in the form of (8) or in the form of (9).

Question arises. In the transformation from (7) to either (8) or (9), there can be different choices of coefficients resulting in different upper bounds of a tuple. In this context, is there a notion of lowest upper bound for a tuple t w.r.t. a given Q, where $t \in Q$? We can answer this question when the size of Q is two. This corresponds to the practical situation of computing an upper bound - to compute an upper bound of a newly retrieved tuple t, we use exactly one another tuple as a reference which is retrieved earlier and whose PRF^ω value is known.

THEOREM 3.6. *Let T be an uncertain table, $Q = \{t', t\}$ be a set of tuples from T. The upper bound u of t, induced by any assignment w.r.t. Q, satisfies $u \geq \frac{Pr(t)}{Pr(t')}\Upsilon(t')$.*

In general, $\frac{Pr(t)}{Pr(t')}\Upsilon(t')$ may or may not be an upper bound of t. When it is, it is guaranteed that there be no assignment w.r.t. $Q = \{t', t\}$ that can yield a lower upper bound.

The lowest upper bound above is defined w.r.t. Q. For different Q's, the lowest upper bounds may well be different. This prompts the question that in practice among all retrieved tuples (let's denote the set by P), which tuple should be chosen as the reference. Clearly, if $t_{lowest} \in P$ is such that $\frac{\Upsilon(t_{lowest})}{Pr(t_{lowest})} \leq \frac{\Upsilon(t')}{Pr(t')}$, for any $t' \in P$, then the upper bound of t is not lower than $\frac{Pr(t)}{Pr(t_{lowest})}\Upsilon(t_{lowest})$ when using the two tuple relation given in Theorem 3.6.

If a computed upper bound is not the lowest according to Theorem 3.6, we can get additional tuples involved in order to improve it. This is easy to understand as with more tuples, we have more information to use and we may get a better upper bound. Let us see an example.

EXAMPLE 3.7. *Consider Example 2.1, where t_6 and t_4 are from different generation rules and $score(t_6) > score(t_4)$. We have*

$$\Upsilon(t_6) = Pr(S_{10})U_{10} + Pr(S_{11})U_{11} = 0.2U_{10} + 0.3U_{11}$$
$$\Upsilon(t_4) = Pr(S_{20})U_{20} + Pr(S_{21})U_{21} = 0.2U_{20} + 0.2U_{21}$$

Consider an upper bound of t_4. By Theorem 3.6, we know that the upper bound of t_4 induced from any assignment w.r.t. $\{t_6, t_4\}$ is no smaller than $\frac{Pr(t_4)}{Pr(t_6)}\Upsilon(t_6)$. But there does not exist an assignment w.r.t. $\{t_6, t_4\}$, which induces this upper bound. Now let us introduce t_3. We then have

$$\begin{aligned}\Upsilon(t_6) &= Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11} + Pr(S_{12}) \times U_{12} \\ &= 0.06U_{10} + 0.23U_{11} + 0.21U_{12} \\ \Upsilon(t_4) &= Pr(S_{31}) \times U_{31} + Pr(S_{32}) \times U_{32} = 0.2U_{31} + 0.2U_{32}\end{aligned}$$

If we set $c_1 = \frac{1}{Pr(t_6)} = 2$, $c_2 = 0$, and $c_3 = -\frac{1}{Pr(t_4)} = -2.5$ (note that coefficients c_1, c_2, c_3 are associated with tuples, in the order of scores, t_6, t_3, t_4, respectively), we will get

$$\begin{aligned}\frac{\Upsilon(t_6)}{Pr(t_6)} - \frac{\Upsilon(t_4)}{Pr(t_4)} &= 0.04(U_{10} - U_{31}) + 0.46(U_{11} - U_{31}) \\ &+ 0.08(U_{10} - U_{32}) + 0.42(U_{12} - U_{32}) \geq 0\end{aligned}$$

So we get an upper bound of t_4 as $\frac{Pr(t_4)}{Pr(t_6)}\Upsilon(t_6)$. This upper bound is induced from an assignment w.r.t. $\{t_6, t_4, t_3\}$.

The above example shows that there are cases where by using an additional tuple we can get a better upper bound.

4. DERIVING PRACTICAL METHODS

The method given in the last section is only a scheme, which does not tell us how to choose the coefficients c_i so that the targeted transformations are possible. In this section, we show how to instantiate this scheme to generate practical upper bound methods.

Briefly, if two tuples t_1 and t_2 ($score(t_1) \geq score(t_2)$) are from the same generation rule, there is a choice of c_i such that the lowest upper bound of t_2 w.r.t. $\{t_1, t_2\}$ is guaranteed. In the case that two tuples are from different generation rules, if some condition is satisfied, the lowest upper bound is guaranteed, and if it is not we provide a conservative estimate, i.e., an upper bound is generated in any case. We may continue this exercise to three tuples and so on.

4.1 Two tuples from the same generation rule

Assume two real tuples t_1 and t_2 from the same generation rule, with $score(t_1) \geq score(t_2)$. As they are involved in the same generation rule, $l = 1$. So according to (5), we have

$$\Upsilon(t_1) = Pr(S_{10}) \times U_{10} \qquad \Upsilon(t_2) = Pr(S_{20}) \times U_{20}$$

where, as $l = 1$, $Pr(t_1) = Pr(S_{10})$ and $Pr(t_2) = Pr(S_{20})$ (by equation (6)). It is also clear that $U_{10} \geq U_{20}$. Then, by setting $c_1 = \frac{1}{Pr(t_1)}$ and $c_2 = -\frac{1}{Pr(t_2)}$ the right hand side of equation (7) $\frac{\Upsilon(t_1)}{Pr(t_1)} - \frac{\Upsilon(t_2)}{Pr(t_2)} = U_{10} - U_{20} \geq 0$ is an instance of (8). This is to say that if we know the PRF^{ω} value of t_1, we can compute the lowest upper bound of t_2.

THEOREM 4.1. *Let T be an uncertain table, and t_1 and t_2 be two real tuples in T that are involved in the same generation rule, with $score(t_1) \geq score(t_2)$. Then, we have $\Upsilon(t_2) \leq \frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$.*

The time complexity to compute this upper bound is $O(1)$.

4.2 Two tuples from different generation rules

Assume two real tuples t_1 and t_2 from different generation rules and $score(t_1) \geq score(t_2)$. As they belong to different generation rules, $l = 2$. So, according to (5), we have

$$\Upsilon(t_1) = Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11}$$
$$\Upsilon(t_2) = Pr(S_{20}) \times U_{20} + Pr(S_{21}) \times U_{21}$$

Let us set $c_1 = \frac{1}{Pr(t_1)}$ and $c_2 = -\frac{1}{Pr(t_2)}$. We can show

THEOREM 4.2. *Let T be an uncertain table, and t_1 and t_2 be two real tuples in T belonging to different generation rules. Assume $score(t_1) \geq score(t_2)$. If $\frac{Pr(S_{10})}{Pr(t_1)} \geq \frac{Pr(S_{20})}{Pr(t_2)}$, then $\Upsilon(t_2) \leq \frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$.*

In Theorem 4.2, we need compute $Pr(S_{ij})$. As discussed in Section 3.1, the time complexity of computing $Pr(S_{ij})$ is $O(ql^2 + ql\tau)$. Here $q=2$ and $l=2$. Thus the time complexity of computing $Pr(S_{ij})$ is $O(\tau)$, so is the cost of computing the upper bound of t_2.

If the condition in Theorem 4.2 is not satisfied, question arises as whether there is a reasonable way to estimate an upper bound of t_2. The answer is yes. For this, let us set $c_1 = \frac{Pr(S_{20})}{Pr(S_{10}) \times Pr(t_2)}$ and $c_2 = -\frac{1}{Pr(t_2)}$. We can show

THEOREM 4.3. *Let T be an uncertain table, and t_1 and t_2 be two real tuples in T from different generation rules, with $score(t_1) \geq score(t_2)$. If $\frac{Pr(S_{10})}{Pr(t_1)} < \frac{Pr(S_{20})}{Pr(t_2)}$ and the weight function is non-negative, we have $\Upsilon(t_2) \leq \frac{Pr(S_{20})}{Pr(S_{10})}\Upsilon(t_1)$.*

That is, if the condition in Theorem 4.2 is not satisfied (i.e., its negation $\frac{Pr(S_{10})}{Pr(t_1)} < \frac{Pr(S_{20})}{Pr(t_2)}$ is satisfied), we still can get an upper bound of t_2, which is $\Upsilon(t_1)$ multiplying the factor $\frac{Pr(S_{20})}{Pr(S_{10})}$. Obviously, this upper bound is higher.

The cost of computing the upper bound above is $O(\tau)$.

4.3 Three tuples

When the condition in Theorem 4.2 is not satisfied, it is possible to improve the upper bound given in Theorem 4.3. Let t_1, t_2, t_3 be real tuples from different generation rules, with $score(t_1) \geq score(t_2) \geq score(t_3)$. From (5) we get

$$\Upsilon(t_1) = Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11} + Pr(S_{12}) \times U_{12}$$
$$\Upsilon(t_2) = Pr(S_{20}) \times U_{20} + Pr(S_{21}) \times U_{21} + Pr(S_{22}) \times U_{22}$$
$$\Upsilon(t_3) = Pr(S_{30}) \times U_{30} + Pr(S_{31}) \times U_{31} + Pr(S_{32}) \times U_{32}$$

Through some derivations, we can get

THEOREM 4.4. *Let T be an uncertain table, and t_1, t_2, and t_3 be three real tuples in T from three different generation rules, with $score(t_1) \geq score(t_2) \geq score(t_3)$. Let $a, b \in \{1, 2, 3\}$ and $a < b$. If $\frac{Pr(S_{a0})}{Pr(t_a)} \geq \frac{Pr(S_{b0})}{Pr(t_b)}$ and $\frac{Pr(S_{a2})}{Pr(t_a)} \leq \frac{Pr(S_{b2})}{Pr(t_b)}$, then we have $\Upsilon(t_b) \leq \frac{Pr(t_b)}{Pr(t_a)} \times \Upsilon(t_a)$.*

The cost of computing the upper bound above is $O(\tau)$.

5. PRUNING FOR PRF^E

It is known that the PRF^e value of a tuple can be determined in constant time when all the tuples are sorted according to scores [7]. Thus, the method of pruning given in previous sections may not be worthwhile for PRF^e. Here, we show a special property of PRF^e for pruning, which can terminate the top-k computation earlier.

THEOREM 5.1. *Let T be an uncertain table and $t \in T$. Suppose the ranking function is PRF^e and $\omega(i) = \alpha^i$, where α is a real number and $0 < \alpha < 1$. Then for any tuple $t' \in T$ such that $score(t') \leq score(t)$, $\Upsilon(t') \leq \frac{1}{\alpha} \times \frac{1}{Pr(t)}\Upsilon(t)$.*

That is, if we know $PRF^e(t)$, then $\frac{1}{\alpha} \times \frac{1}{Pr(t)}\Upsilon(t)$ is an upper bound of all tuples whose scores are smaller than $score(t)$. Following the top-k algorithm given in [7], where tuples in an uncertain table are sorted according to their scores, when this value is lower than the kth largest PRF^e value found so far the computation can safely terminate.

6. EXPERIMENTS

We combine our pruning methods with top-k algorithms for PRF^ω and PRF^e, respectively. The procedure outlined in Section 2.1 uses two tuples for pruning, which was employed in our experiments. All the experiments were run on a quad-core 2.3GHZ PC with 16GB RAM, running Linux operating system. The algorithms are implemented in Microsoft Visual C++ V6.0.

6.1 Implemented algorithms

We now fill more details on the combined algorithm in Section 2.1 (a complexity analysis is given in Appendix D).

After the first k tuples, we retrieve a new tuple t_{new}. To compute the upper bound of t_{new}, we maintain a tuple called t_{lowest}, which is among the retrieved tuples whose PRF^ω values have been computed. The tuple t_{lowest} must be the one that has the lowest ratio between its PRF^ω value and its membership probability among all retrieved tuples, i.e., $\frac{\Upsilon(t_{lowest})}{Pr(t_{lowest})} \leq \frac{\Upsilon(t')}{Pr(t')}$, for any retrieved tuple t' such that $\Upsilon(t')$ is computed. If t_{new} is involved in the same generation rule with t_{lowest}, we get the upper bound of t_{new} as $\frac{Pr(t_{new})}{Pr(t_{lowest})}\Upsilon(t_{lowest})$ from Theorem 4.1, which is the lowest for the set of retrieved tuples whose PRF^ω values are computed when using the two tuple relation.

If t_{new} and t_{lowest} are from different generation rules, we check whether the condition in Theorem 4.2 is satisfied. If it is, we get the lowest upper bound of t_{new}. Otherwise Theorem 4.3 provides an upper bound of t_{new}.

Below, we use "computed tuples" to mean the number of tuples whose PRF^ω or PRF^e values are actually computed in running top-k algorithms in [7] combined with pruning.

In our experiments, we implemented the PRF^ω algorithm given in [7] whose complexity is $O(n^3)$. In [7], another algorithm with complexity $O(n^2 log^2 n)$ is also provided. If we use the latter, the "computed tuples" remains the same but the gaps in running times will be smaller. For PRF^e, we implemented the algorithm given in [7], combined with the early termination condition given in Section 5.

6.2 Data sets and weight functions

Normal data sets: Synthetic data sets are generated, each containing some tuples and some multi-tuple generation rules. The number of tuples involved in each multi-tuple generation rule follows the normal distribution, so does the probabilities of independent tuples and multi-tuple generation rules. To generate different data sets, we vary the expectation of the membership probabilities of the independent tuples and the size of a multi-tuple generation rule.

Special data sets: Intuitively, pruning seems ineffective on data sets like the following: the scores of the tuples are in an descending order and their membership probabilities are in an ascending order. Although these data sets do not seem to be typical in the real world, it can be used to illustrate an interesting phenomenon of performance changes from variants of these data sets, which are obtained as follows: we update such a data set by swapping the membership probabilities of different tuples such that the membership probabilities of the resulting tuples are not strictly in an ascending order. To get different data sets, we vary the ratio between the number of swapping tuples and the number of all tuples.

Real data set: We also experimented with a real data set, with conclusions similar to those with normal data sets. We provide the details in Appendix F, along with an experiment to show the closeness of the computed upper bounds compared with real PRF^ω values.

Weight functions: We experimented with different weight functions $\omega(i)$ for PRF^ω: randomly generated weight functions (RGWFs), the weight function $\omega(i) = n - i$, where n is the number of tuples in an uncertain database and i is the position of a tuple in a possible world, and the weight function from the *PT-k query answer* [4]. All these functions are related only to i and monotonically non-increasing. RGWFs are generated as follows: we generate n random numbers and sort them into a descending order; the number in the position i of this order is the weight value of $\omega(i)$.

In our experiments, we found that the performance of our pruning methods for RGWFs is similar to that using the weight function $\omega(i) = n - i$. So we only give the results for RGWFs and the one based on PT-k query answer.

6.3 Results

In each figure included in this section (except Figure 4), there are three graphs, (a), (b) and (c). Each graph shows computed tuples or running times (on y-axis) over one of the three parameters (on x-axis): the expected membership probabilities of independent tuples, the expected number of tuples in a generation rule, and different values of k. Note that the first two parameters result in different data sets. Figure 4 contains two graphs over swapping ratios.

Figure 2 shows the computed tuples for normal data sets with pruning, for a RGWF *random*1 and *PT-k query answer*. The size of the database is 100,000 tuples. We set $k = 50$ in both (a) and (b). In Figure 2(a), we see that the computed tuples is between 50 and 400. Compared with the size of the data set, with pruning we only need to compute the PRF^ω values for a very small portion of the database. This is an impressive improvement. With the increase of the expected membership probability of the independent tuples in a data set, less tuples need be computed. In Figure 2(b), we see that the computed tuples is between 100 and 400. In Figure 2(c), we vary the value of k from 50 to 250. We see that only a small number of tuples are computed for their PRF^{ω} values. Most tuples are pruned.

Figure 3 shows the running times for normal data sets. Here we also test for 2 weight functions: a RGWF $random2$ and PT-k query answer. The size of the data set is 2000 tuples (due to much longer computation time, we use smaller data sets for the experiments without pruning or where pruning is not effective). We set $k = 50$ for $random2$ and $k = 150$ for PT-k query answer in both (a) and (b) (for RGWF $random2$, the performance gains will be smaller with larger values of k). The improvement is orders of magnitude. The improvement is similar for larger data sets (we scaled the size up to 10,000 tuples and tested some selectively).

Figure 4 uses the special data sets with different ratios between the number of swapping tuples and the number of all tuples, and different weight functions. The size of the data sets is 2000. We set $k = 50$ for $random2$ and $k = 150$ for the PT-k query answer. In the figure ((a) is about computed tuples and (b) running times), we see that when the swapping ratio is low, pruning produces little performance gains. With the increase of the swapping ratio, more tuples are pruned and the running times reduced substantially.

Comparison with pruning of [4]: It can be shown that the main pruning theorems of [4] are special cases of our pruning theorems (see Appendix E). We compare with the simple pruning technique used in [4] on the normal data sets, with size 2000. We used two weight functions: a RGWF $random2$ and the PT-k query answer. Figure 5 compares the computed tuples, while Figure 6 is on the running times. In the graphs (a) and (b) of both Figures 5 and 6, we set $k = 50$ for $random2$ and $k = 150$ for PT-k query answer. It can be noticed that substantial improvement is generated.

Early termination for PRF^e: We use the early termination condition given in Section 5 to terminate a computation when the condition is satisfied. The test data sets are normal data sets, with the size being 1,000,000 tuples. We set $\alpha = 0.95$ (the impact of this value is not big as long as it is close to 1). We summarize the results as follows. When k=50000 or higher, the computation terminates right after the first k tuples are retrieved. With smaller k's, the number of retrieved tuples could be higher than k, but not substantially. The running times are shortened to one fifth to one twentieth of the ones without pruning. This shows that Theorem 5.1 yields a highly effective pruning method.

Experiments with 3-tuples: We start with two rules t_{lowest} and t_{new} as described above, and introduce a third tuple. It can be shown that this new tuple must be involved in a generation rule which contains some tuples whose score is between $score(t_{lowest})$ and $score(t_{new})$ (which therefore must already been retrieved). Thus, in our experiments a third tuple is randomly chosen satisfying the above condition. However, our experiments with the four weight functions for the normal and special synthetic data sets, as well as for the real data set (see Appendix F), didn't show performance gains, as the number of computed tuples and running times are very close to those by the pruning method with two tuples. One observation is that, in most cases, the lowest upper bound has already been reached using two tuples.

7. FINAL REMARKS

There are three assumptions in our work. The first is that the weight function is independent of tuples. Technically, this assumption can be removed if $\omega(t, i)$ is restricted to $\omega(score(t), i)$ (i.e., anything that depends on t depends on $score(t)$, and $\omega(score(t), i)$ is non-negative and monotonically non-decreasing w.r.t. $score(t)$). Another assumption in our work is that $\omega(t, i)$ is monotonically non-increasing w.r.t. i. This assumption is reasonable, as in most real applications a smaller i (hence a higher position) is clearly more important than a larger i. The third assumption requires the weight function to be non-negative. We are currently looking into whether this assumption can be removed. Further, it is interesting to see whether our theory and resulting methods can be extended to general data models such as the probabilistic and/xor tree model. Such an extension is non-trivial. Applying our top-k algorithms to real world applications is another important task for future work.

Acknowledgment: All authors of this paper were supported in part by grants from NSERC. L. Yuan and J. You also acknowledge the support from the 863 Project of China under grant 2009AA01Z150.

8. REFERENCES

[1] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *Proc. ICDE*, pages 305–316, 2009.

[2] N. Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *Proc. VLDB*, pages 696–707, 1990.

[3] A. Fuxman, E. Fazli, and R. Miller. Conquer: Efficient management of inconsistent databases. In *Proc. SIGMOD*, pages 155–166, 2005.

[4] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *Proc. SIGMOD*, pages 1357–1364, 2008.

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

[6] J. Li and A. Deshpande. Consensus answers for queries over probabilistic databases. In *Proc. PODS*, pages 259–268, 2009.

[7] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probablistic databases. *The VLDB Journal*, pages 249–275, 2011. (A preliminary version appeared in *Proc. VLDB-09*, pp 305-316).

[8] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *Proc. VLDB*, pages 15–26, 2007.

[9] A. Silberstein, R. Braynard, C. Ellis, and K. Munagala. A sampling-based approach to optimizing top-k queries in sensor networks. In *Proc. ICDE*, 2006.

[10] M. Soliman, I. Ilyas, and K. Chang. Top-k query processing in uncertain databases. In *Proc. ICDE*, pages 896–905, 2007.

[11] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. *IEEE Transactions on Knowledge and Data Engineering*, 20(12):1699–1711, 2008.

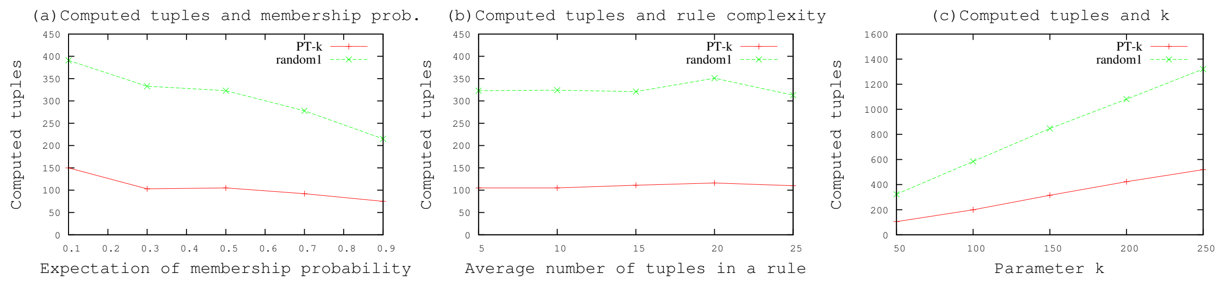

Figure 2: Computed tuples for PRF^{ω} on normal data sets

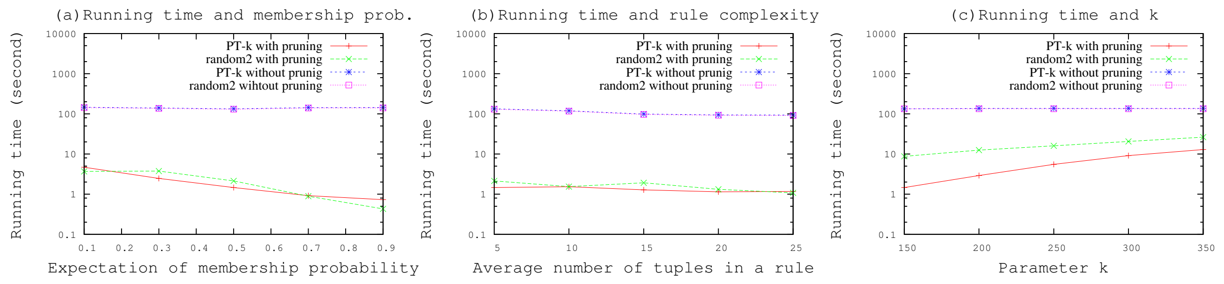

Figure 3: Running times for PRF^{ω} on normal data sets

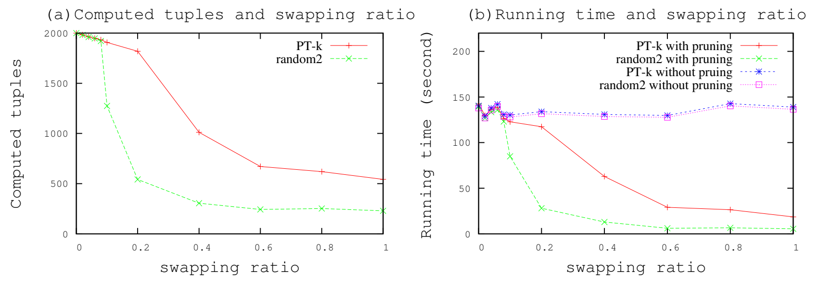

Figure 4: Comparison on special data sets

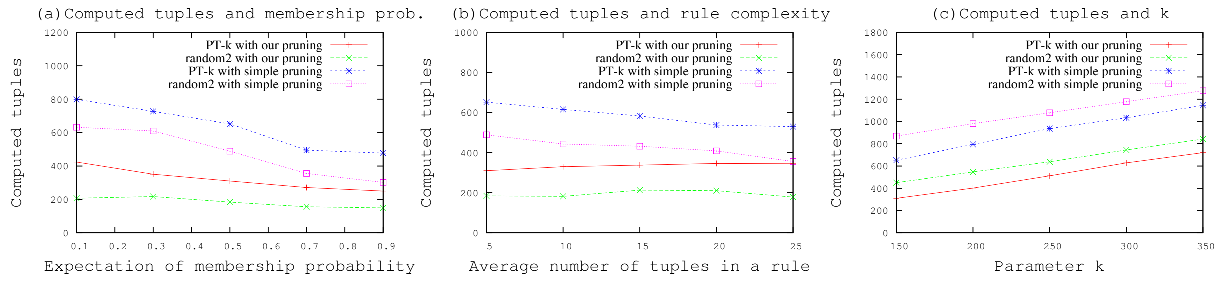

Figure 5: Comparison with previous pruning method: computed tuples

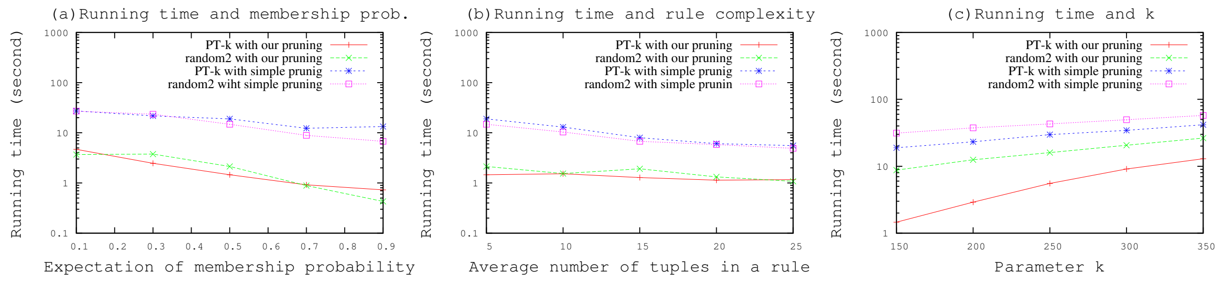

Figure 6: Comparison with previous pruning method: running times

APPENDIX

A. TOP-K ALGORITHMS

We introduce the algorithms in [7] for computing the PRF^ω value of a tuple and top-k tuples. Let T be an uncertain database where the tuples $t_1, ..., t_n$ are sorted in a non-increasing order by scores. To compute the PRF^ω value of t_i, we only need to consider $T_i = \{t \in T \mid \mathit{score}(t) > \mathit{score}(t_i)\}$.

In [7], a different but equivalent representation of $\Upsilon(t)$ is given: $\Upsilon(t) = \sum_{i>0} \omega(t, i) \cdot Pr(r(t) = i)$, where $Pr(r(t) = i)$ is the sum of the probabilities of the possible worlds in which t ranks at i-th position. Assume the correlation of tuples in T is that of the x-tuple and let R be the set of all the generation rules of T. Let $t_i \in r'$, and $R_i = \{r \in R - \{r'\} \mid \exists t \in T_i, t \in r\}$. Let $\Theta_i(r)$ be the set of tuples that are in T_i and involved in r. Consider the following generating function: $F^i(x, y) = Pr(t_i)y \prod_{r \in R_i}(1 - \sum_{t \in \Theta_i(r)} Pr(t) + x \sum_{t \in \Theta_i(r)} Pr(t))$. The coefficient of the term $x^{j-1}y$ in the expansion of $F^i(x, y)$ is $Pr(r(t_i) = j)$. Then we can compute $\Upsilon(t_i)$. If we expand $F^i(x, y)$ by polynomial multiplication one by one, the complexity of the algorithm to get top-k tuples (computing $\Upsilon(t)$ for all the tuples and choosing the k tuples with highest PRF^ω values) is $O(n^3)$. If we use FFT (Fast Fourier Transformation) for the multiplication of polynomials, the complexity is $O(n^2 log^2 n)$.

B. THEORETICAL DEVELOPMENT

We provide more details on the theoretical development. We give proofs for Theorems 3.1, 3.2, 3.5 and 3.6, as they form the basis of our theory. For practical pruning methods, we provide proofs of Theorems 4.2 and 4.3. Finally, a proof of Theorem 5.1 will show why the pruning for PRF^e works.

We will present the theoretical development in this order. It is not difficult to prove the following lemmas.

Lemma B.1. *Let W and W' be two possible worlds and η and η' be two tuple sets such that $\eta \subset W$ and $\eta' \subset W'$. If $W' = (W - \eta) \cup \eta'$ and $W = (W' - \eta') \cup \eta$, then $\frac{Pr(W)}{Pr(W')} = \frac{\prod_{t \in \eta} Pr(t)}{\prod_{t \in \eta'} Pr(t)}$.*

Lemma B.2. *For a tuple set η where all tuples are involved in different generation rules, $Pr(\eta) = \prod_{t_i \in \eta} Pr(t_i)$.*

Proof of Theorem 3.1:
We know there exist possible worlds $W_1, W_2, ..., W_e$ which contain all the tuples in η_1 and possible worlds $W_1', W_2', ..., W_e'$ which contain all the tuples in η_2, such that $W_v' = (W_v - \eta_1) \cup \eta_2$ and $W_v = (W_v' - \eta_2) \cup \eta_1$ $(1 \le v \le e)$. As $\eta_1, \eta_2 \in S_{ij}$, η_1 and η_2 both contain j tuples which have a higher score than t_i. So we know that t_i has the same position in W_v and W_v'. Thus $\beta_{W_v}(t_i) = \beta_{W_v'}(t_i)$ and $\omega(\beta_{W_v}(t_i)) = \omega(\beta_{W_v'}(t_i))$. From equation (3), we have

$$\Upsilon_{\eta_1}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i)) \times Pr(W_v)$$
$$\Upsilon_{\eta_2}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v'}(t_i)) \times Pr(W_v')$$

By Lemma B.1, we have $\frac{Pr(W_v)}{Pr(W_v')} = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)}$. So we get $\Upsilon_{\eta_1}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v'}(t_i)) \times \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times Pr(W_v') = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \sum_{v=1}^{e} \omega(\beta_{W_v'}(t_i)) \times Pr(W_v') = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \Upsilon_{\eta_2}(t_i)$. It follows $\frac{\Upsilon_{\eta_1}(t_i)}{\Upsilon_{\eta_2}(t_i)} = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)}$. From Lemma B.2, we have $Pr(\eta_1) = \prod_{t_u \in \eta_1} Pr(t_u)$ and $Pr(\eta_2) = \prod_{t_u \in \eta_2} Pr(t_u)$. So we have $\frac{\Upsilon_{\eta_1}(t_{x_i})}{\Upsilon_{\eta_2}(t_i)} = \frac{Pr(\eta_1)}{Pr(\eta_2)}$. That is $\frac{\Upsilon_{\eta_1}(t_i)}{Pr(\eta_1)} = \frac{\Upsilon_{\eta_2}(t_i)}{Pr(\eta_2)}$. □

Definition of U_{ij} for Empty S_{ij} under PRF^ω:
We define U_{ij} for empty S_{ij}. Let us first point out some properties of S_{ij}. From the definition of S_{ij}, we know that for a fixed i, the non-empty sets S_{ij} must be consecutive. This means it is impossible that S_{ih} and $S_{i(h+2)}$ $(0 \le h \le l-3)$ are non-empty sets and $S_{i(h+1)}$ is an empty set. From the definition we also know there exists at least one non-empty S_{ij} for a fixed i. Given $t_i \in Q$, let us denote the consecutive sequence of all non-empty sets (w.r.t. t_i) by

$$S_{ij_1}, S_{i(j_1+1)}, ..., S_{ij_2}$$

where $0 \le j_1 \le j_2 \le l-1$. Let us call S_{ij_1} the first non-empty S_{ij} for t_i and S_{ij_2} the last non-empty S_{ij} for t_i. We see that $S_{i0}, S_{i1}, ..., S_{i(j_1-1)}$ and $S_{i(j_2+1)}, S_{i(j_2+2)}, ..., S_{i(l-1)}$ are all empty sets.

For any $1 \le i_1, i_2 \le q$, let $S_{i_1j_1}$ be the first non-empty S_{ij} for t_{i_1}, $S_{i_1j_2}$ be the last non-empty S_{ij} for t_{i_2}, $S_{i_2j_3}$ be the first non-empty S_{ij} for t_{i_2} and $S_{i_2j_4}$ be the last non-empty S_{ij} for t_{i_2}. From the definition of S_{ij}, we know that if $\mathit{score}(t_{i_1}) \ge \mathit{score}(t_{i_2})$, then $j_1 \le j_3$ and $j_2 \le j_4$.

Now we define U_{ij} for empty S_{ij}. For a tuple $t_i \in Q$ $(1 \le i \le q)$, let S_{ij_1} be the first non-empty S_{ij} for t_i and S_{ij_2} be the last non-empty S_{ij} for t_i. For an empty S_{ij}, we define

$$U_{ij} = \begin{cases} U_{ij_1} & \text{if } j < j_1 \\ U_{ij_2} & \text{if } j > j_2 \end{cases}$$

Proof of Theorem 3.2:
We prove part (i). Part (ii) can be proved similarly.

We first prove that for any S_{ij_1}, S_{ij_2}, S_{i_1j}, and S_{i_2j} which are non-empty, where $(1 \le i, i_1, i_2 \le q)$ and $(0 \le j, j_1, j_2 \le l-1)$, the conclusion in part (i) holds.

Similar to the proof in Theorem 3.1, let $\eta_1 \in S_{ij_1}$ and $\eta_2 \in S_{ij_2}$. By definition, we know there exist possible worlds $W_1, W_2, ..., W_e$ which contain all the tuples in η_1 and possible worlds $W_1', W_2', ..., W_e'$ which contain all the tuples in η_2, such that $W_v' = (W_v - \eta_1) \cup \eta_2$ and $W_v = (W_v' - \eta_2) \cup \eta_1$ $(1 \le v \le e)$. As $\eta_1 \in S_{ij_1}$, η_1 contains j_1 tuples which have a higher score than t_i. Since $\eta_2 \in S_{ij_2}$, η_2 contains j_2 tuples which have a higher score than t_i. Since $j_1 \le j_2$, compared with W_v', W_v must contain a less or equal number of tuples which has a higher score than t_i. So t_i has an equal or higher position in W_v than in W_v'. As we have assumed that the weight function is monotonically non-increasing, we get $\omega(\beta_{W_v}(t_i)) \ge \omega(\beta_{W_v'}(t_i))$. Because $\Upsilon_{\eta_1}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i))Pr(W_v)$,
$\Upsilon_{\eta_2}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v'}(t_i))Pr(W_v')$ and
$\frac{Pr(W_v)}{Pr(W_v')} = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)}$ (Lemma B.1), we can get $\Upsilon_{\eta_1}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i)) \times \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times Pr(W_v') = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i)) \times Pr(W_v') \ge \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \sum_{v=1}^{e} \omega(\beta_{W_v'}(t_i)) \times Pr(W_v') = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \Upsilon_{\eta_2}(t_i)$. So $\frac{\Upsilon_{\eta_1}(t_i)}{\prod_{t_u \in \eta_1} Pr(t_u)} \ge \frac{\Upsilon_{\eta_2}(t_i)}{\prod_{t_u \in \eta_2} Pr(t_u)}$. Because $Pr(\eta_1) = \prod_{t_u \in \eta_1} Pr(t_u)$ and $Pr(\eta_2) = \prod_{t_u \in \eta_2} Pr(t_u)$ (from Lemma

B.2), we can get $\frac{\Upsilon_{\eta_1}(t_i)}{Pr(\eta_1)} \geq \frac{\Upsilon_{\eta_2}(t_i)}{Pr(\eta_2)}$. This is $U_{ij_1} \geq U_{ij_2}$. So we have proved that the conclusion in part (i) holds for non-empty S_{ij_1}, S_{ij_2}, S_{i_1j}, and S_{i_2j}.

From the definition of U_{ij} for empty S_{ij}, it is easy to see that the conclusion in part (i) holds for any S_{ij_1}, S_{ij_2}, S_{i_1j}, and S_{i_2j}. We are done. □

Proof of Theorem 3.5:
For notational convenience, let s and t in the theorem be also named as t_a and t_b, respectively. By Theorem 3.2, from $score(t_a) \geq score(t_b)$, we know $U_{aj} \geq U_{bj}$. We construct an assignment θ of c_i w.r.t. Q as follows: set c_i $(i \neq a, b) = 0$ in θ. Then the right hand side of (7) can be written as: $\sum_{j=0}^{l-1} c_a \times Pr(S_{aj}) \times U_{aj} + c_b \times Pr(S_{bj}) \times U_{bj}$. Here $c_a > 0$ and $c_b < 0$. We just need to set c_a large enough and the absolute value of c_b small enough such that $c_a \times Pr(S_{aj}) \geq -c_b \times Pr(S_{bj})$, then it is clear that $\sum_{j=0}^{l-1} c_a \times Pr(S_{aj}) \times U_{aj} + c_b \times Pr(S_{bj}) \times U_{bj} = \sum_{j=0}^{l-1} -c_b \times Pr(S_{bj})(U_{aj} - U_{bj}) + (c_a \times Pr(S_{aj}) + c_b \times Pr(S_{bj}))U_{aj}$. And this is an expression either in the form of (8) or in the form of (9). □

Proof of Theorem 3.6:
If $score(t') < score(t)$, it is easy to see that no assignment w.r.t. Q can induce an upper bound of t. For an assignment w.r.t. Q, if the coefficient of t' is not greater than 0, it is also easy to see that this assignment cannot induce an upper bound of t. So if an assignment w.r.t. Q can induce an upper bound, $score(t') \geq score(t)$ and the coefficient of t' is greater than 0.

Let ϕ be any assignment w.r.t. Q which induces an upper bound u of t. Let the coefficients of t', t in ϕ be c_1, c_2, respectively. We know $c_1 > 0$ and $c_2 < 0$. We thus have

$$\begin{aligned} & c_1\Upsilon(t') + c_2\Upsilon(t) \\ & = \textstyle\sum_{j=0}^{l-1} c_1 Pr(S_{1j})U_{1j} + \sum_{j=0}^{l-1} c_2 Pr(S_{2j})U_{2j} \end{aligned}$$

To make the transformation from the right hand side of the equation above to (8) or (9), the sum of the positive coefficients of U_{ij} must not be smaller than the sum of the absolute value of the negative coefficients of U_{ij}. From equation (6), we have $\sum_{j=0}^{l-1} Pr(S_{1j}) = Pr(t')$ and $\sum_{j=0}^{l-1} Pr(S_{2j}) = Pr(t)$. So we have $c_1 Pr(t') \geq -c_2 Pr(t)$ and this is $c_1 \geq -c_2 \frac{Pr(t)}{Pr(t')}$.

From (2), we know the upper bound u of t induced from ϕ is $u = \frac{c_1\Upsilon(t')}{-c_2}$. So we can get $u \geq \frac{Pr(t)}{Pr(t')}\Upsilon(t')$. □

Proof of Theorem 4.2:
From Theorem 3.2, we know that $U_{10} \geq U_{20}, U_{11} \geq U_{21}, U_{10} \geq U_{11}$. So $U_{10} \geq U_{21}$. From equation (6), we have $Pr(t_1) = Pr(S_{10}) + Pr(S_{11})$ and $Pr(t_2) = Pr(S_{20}) + Pr(S_{21})$. So we can get $\frac{Pr(S_{10})}{Pr(t_1)} - \frac{Pr(S_{20})}{Pr(t_2)} = \frac{Pr(S_{21})}{Pr(t_2)} - \frac{Pr(S_{11})}{Pr(t_1)}$.

Let us set $c_1 = \frac{1}{Pr(t_1)}$ and $c_2 = -\frac{1}{Pr(t_2)}$. We can get $\frac{1}{Pr(t_1)} \times \Upsilon(t_1) - \frac{1}{Pr(t_2)} \times \Upsilon(t_2) =$
$\frac{Pr(S_{10})}{Pr(t_1)} \times U_{10} + \frac{Pr(S_{11})}{Pr(t_1)} \times U_{11} - \frac{Pr(S_{20})}{Pr(t_2)} \times U_{20} - \frac{Pr(S_{21})}{Pr(t_2)} \times U_{21} =$
$\frac{Pr(S_{20})}{Pr(t_2)} \times (U_{10} - U_{20}) + \frac{Pr(S_{11})}{Pr(t_1)} \times (U_{11} - U_{21}) + (\frac{Pr(S_{10})}{Pr(t_1)} - \frac{Pr(S_{20})}{Pr(t_2)}) \times U_{10} - (\frac{Pr(S_{21})}{Pr(t_2)} - \frac{Pr(S_{11})}{Pr(t_1)}) \times U_{21} = \frac{Pr(S_{20})}{Pr(t_2)} \times (U_{10} - U_{20}) + \frac{Pr(S_{11})}{Pr(t_1)} \times (U_{11} - U_{21}) + (\frac{Pr(S_{10})}{Pr(t_1)} - \frac{Pr(S_{20})}{Pr(t_2)}) \times (U_{10} - U_{21})$.
If $\frac{Pr(S_{10})}{Pr(t_1)} \geq \frac{Pr(S_{20})}{Pr(t_2)}$, then $\frac{1}{Pr(t_1)} \times \Upsilon(t_1) \geq \frac{1}{Pr(t_2)} \times \Upsilon(t_2)$. □

Proof of Theorem 4.3:
From equation (6), we have $Pr(t_1) = Pr(S_{10}) + Pr(S_{11})$ and $Pr(t_2) = Pr(S_{20}) + Pr(S_{21})$. Because $\frac{Pr(S_{10})}{Pr(t_1)} < \frac{Pr(S_{20})}{Pr(t_2)}$, we can get $\frac{Pr(S_{10})}{Pr(S_{10})+Pr(S_{11})} < \frac{Pr(S_{20})}{Pr(S_{20})+Pr(S_{21})}$. From this, we have $Pr(S_{20}) \times Pr(S_{11}) - Pr(S_{10}) \times Pr(S_{21}) > 0$. From Theorem 3.2, we know that $U_{10} \geq U_{20}, U_{11} \geq U_{21}$.

Since $score(t_1) \geq score(t_2)$ and the tuples are involved in different generation rules, $Pr(S_{10}) > 0$. Let us set $c_1 = \frac{Pr(S_{20})}{Pr(S_{10}) \times Pr(t_2)}$ and $c_2 = -\frac{1}{Pr(t_2)}$. We get $\frac{Pr(S_{20})}{Pr(S_{10}) \times Pr(t_2)} \times \Upsilon(t_1) - \frac{1}{Pr(t_2)} \times \Upsilon(t_2) =$
$\frac{Pr(S_{20})}{Pr(S_{10}) \times Pr(t_2)}(Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11}) -$
$\frac{1}{Pr(t_2)}(Pr(S_{20}) \times U_{20} + Pr(S_{21}) \times U_{21}) =$
$\frac{Pr(S_{20})}{Pr(t_2)} \times (U_{10} - U_{20}) + \frac{Pr(S_{21})}{Pr(t_2)} \times (U_{11} - U_{21}) +$
$\frac{Pr(S_{20}) \times Pr(S_{11}) - Pr(S_{10}) \times Pr(S_{21})}{Pr(S_{10}) \times Pr(t_2)} \times U_{11} \geq 0$.

So when $\frac{Pr(S_{10})}{Pr(t_1)} < \frac{Pr(S_{20})}{Pr(t_2)}$ and the weight function is non-negative (such that U_{ij} is non-negative), we have $\frac{Pr(S_{20})}{Pr(S_{10})} \times \Upsilon(t_1) \geq \Upsilon(t_2)$. □

To prove Theorem 5.1, we need the following lemma.

LEMMA B.3. *Let T be an uncertain table and $Q = \{t_1, ..., t_q\}$ a set of tuples from T. Suppose the ranking function is PRF^e and the weight function is $\omega(i) = \alpha^i$. Assume α is a real number and $0 < \alpha < 1$. For any non-empty S_{ij} and $S_{i(j+1)}$ $(0 \leq j \leq l-2, l \geq 2)$, we have $U_{i(j+1)} = \alpha \times U_{ij}$.*

PROOF. Suppose the tuples in Q are involved in l $(l \leq q)$ generation rules in R. Assume $score(t_i) \geq score(t_{i+1})$ $(1 \leq i \leq q-1)$.

Similar to the proof of part (i) of Theorem 3.2, let $\eta_1 \in S_{ij}$ and $\eta_2 \in S_{i(j+1)}$. By definition, there exist possible worlds $W_1, W_2, ..., W_e$ which contain all the tuples in η_1 and possible worlds $W'_1, W'_2, ..., W'_e$ which contain all the tuples in η_2, such that $W'_v = (W_v - \eta_1) \cup \eta_2$ and $W_v = (W'_v - \eta_2) \cup \eta_1$ $(1 \leq v \leq e)$. Because $\eta_1 \in S_{ij}$, there are j tuples in η_1 having a higher score than t_i. Since $\eta_2 \in S_{i(j+1)}$, there are $j+1$ tuples in η_2 having a higher score than t_i. So we know that W'_v contains one more tuple than W_v with a higher score than t_i. We thus have $\beta_{W_v}(t_i) = \beta_{W'_v}(t_i) - 1$ and $\omega(\beta_{W_v}(t_i)) = \frac{1}{\alpha} \times \omega(\beta_{W'_v}(t_i))$. From $\Upsilon_{\eta_1}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W_v}(t_i))Pr(W_v)$, $\Upsilon_{\eta_2}(t_i) = \sum_{v=1}^{e} \omega(\beta_{W'_v}(t_i))Pr(W'_v)$, and $\frac{Pr(W_v)}{Pr(W'_v)} = \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)}$, we get $\Upsilon_{\eta_1}(t_i) = \sum_{v=1}^{e} \frac{1}{\alpha} \times \omega(\beta_{W'_v}(t_i)) \times \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times Pr(W'_v) = \frac{1}{\alpha} \times \frac{\prod_{t_u \in \eta_1} Pr(t_u)}{\prod_{t_u \in \eta_2} Pr(t_u)} \times \Upsilon_{\eta_2}(t_i)$. It follows that $\frac{\Upsilon_{\eta_1}(t_i)}{\prod_{t_u \in \eta_1} Pr(t_u)} = \frac{1}{\alpha} \times \frac{\Upsilon_{\eta_2}(t_i)}{\prod_{t_u \in \eta_2} Pr(t_u)}$. As $Pr(\eta_1) = \prod_{t_u \in \eta_1} Pr(t_u)$ and $Pr(\eta_2) = \prod_{t_u \in \eta_2} Pr(t_u)$, we have $\frac{\Upsilon_{\eta_1}(t_i)}{Pr(\eta_1)} = \frac{1}{\alpha} \times \frac{\Upsilon_{\eta_2}(t_i)}{Pr(\eta_2)}$. This is $U_{i(j+1)} = \alpha \times U_{ij}$. □

Recall that in the definition of U_{ij} (see equation (4)), we assume that the corresponding S_{ij} are non-empty. Now, for technical reasons we would like to deal with empty S_{ij} as well. We did this for PRF^ω. Due to the special property of PRF^e (as stated in Lemma B.3), we need to deal with empty S_{ij} differently from PRF^ω.

Recall also that for a fixed i, non-empty S_{ij} appears consecutively. For a tuple $t_i \in Q$, we assume S_{ij_1} is the first non-empty S_{ij} for t_i and S_{ij_2} is the last non-empty S_{ij} for

t_i. Now, for empty S_{ij} we define the corresponding U_{ij} as:

$$U_{ij} = \begin{cases} U_{ij_1} \times \alpha^{j-j_1} & \text{if } j < j_1 \\ U_{ij_2} \times \alpha^{j-j_2} & \text{if } j > j_2 \end{cases}$$

After the setup of U_{ij} for empty S_{ij}, it is easy to check that for any S_{ij} and $S_{i(j+1)}$ $(0 \le j \le l-2, l \ge 2)$, we have $U_{i(j+1)} = \alpha \times U_{ij}$. So we can relax the assumption that given S_{ij} be non-empty in Lemma B.3.

It is easy to see that after the setup of U_{ij} for empty S_{ij} for PRF^e, the conclusions in Theorem 3.2 still hold.

Proof of Theorem 5.1:
As indices are an important part of our notation, let us use t_1 and t_2 for t and t' in the theorem, respectively. There are two cases: the tuples t_1 and t_2 are in two different generation rules or both are in the same generation rule. Let us assume the former first. By the new representation in (5), we have

$$\Upsilon(t_1) = Pr(S_{10}) \times U_{10} + Pr(S_{11}) \times U_{11}$$
$$\Upsilon(t_2) = Pr(S_{20}) \times U_{20} + Pr(S_{21}) \times U_{21}$$

By Lemma B.3, we have $U_{11} = \alpha \times U_{10}$ and $U_{21} = \alpha \times U_{20}$. From equation (6), we have

$$Pr(t_1) = Pr(S_{10}) + Pr(S_{11}) \quad (10)$$
$$Pr(t_2) = Pr(S_{20}) + Pr(S_{21}) \quad (11)$$

It follows that

$$\frac{\Upsilon(t_1)}{Pr(t_1)} = \frac{Pr(S_{10})}{Pr(t_1)} \times U_{10} + (1 - \frac{Pr(S_{10})}{Pr(t_1)}) \times U_{10} \times \alpha$$
$$\frac{\Upsilon(t_2)}{Pr(t_2)} = \frac{Pr(S_{20})}{Pr(t_2)} \times U_{20} + (1 - \frac{Pr(S_{20})}{Pr(t_2)}) \times U_{20} \times \alpha$$

Now, let us divide the two equations above, i.e.,

$$\frac{\frac{\Upsilon(t_1)}{Pr(t_1)}}{\frac{\Upsilon(t_2)}{Pr(t_2)}} = \frac{U_{10}((1-\alpha)\frac{Pr(S_{10})}{Pr(t_1)} + \alpha)}{U_{20}((1-\alpha)\frac{Pr(S_{20})}{Pr(t_2)} + \alpha)}$$

From equation (10), it is clear that $0 \le \frac{Pr(S_{10})}{Pr(t_1)} \le 1$. Similarly, from (11) we have $0 \le \frac{Pr(S_{20})}{Pr(t_2)} \le 1$. Thus, we have $\alpha \le (1-\alpha)\frac{Pr(S_{10})}{Pr(t_1)} + \alpha \le 1$ and $\alpha \le (1-\alpha)\frac{Pr(S_{20})}{Pr(t_2)} + \alpha \le 1$. So we have $\frac{(1-\alpha)\frac{Pr(S_{10})}{Pr(t_1)}+\alpha}{(1-\alpha)\frac{Pr(S_{20})}{Pr(t_2)}+\alpha} \ge \alpha$. Because $U_{10} \ge U_{20}$, we get $\frac{\frac{\Upsilon(t_1)}{Pr(t_1)}}{\frac{\Upsilon(t_2)}{Pr(t_2)}} \ge \alpha$. It follows that $\Upsilon(t_2) \le \frac{1}{\alpha} \times \frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$. As $0 \le Pr(t_2) \le 1$, we have $\Upsilon(t_2) \le \frac{1}{\alpha} \times \frac{1}{Pr(t_1)}\Upsilon(t_1)$ which is the conclusion in the theorem.

Now we consider the second case, namely t_1 and t_2 are involved in the same generation rule. In this case, by the new representation of $\Upsilon(t)$, we have $\Upsilon(t_1) = Pr(S_{10}) \times U_{10}$ and $\Upsilon(t_2) = Pr(S_{20}) \times U_{20}$. From equation (6), we know that $Pr(t_1) = Pr(S_{10})$ and $Pr(t_2) = Pr(S_{20})$. So we have $\frac{\Upsilon(t_1)}{Pr(t_1)} = U_{10}$ and $\frac{\Upsilon(t_2)}{Pr(t_2)} = U_{20}$. Because $U_{10} \ge U_{20}$, we divide the two equations above to get $\frac{\frac{\Upsilon(t_1)}{Pr(t_1)}}{\frac{\Upsilon(t_2)}{Pr(t_2)}} = \frac{U_{10}}{U_{20}} \ge 1 \ge \alpha$. So we have $\Upsilon(t_2) \le \frac{1}{\alpha} \times \frac{Pr(t_2)}{Pr(t_1)}\Upsilon(t_1)$. Since $0 \le Pr(t_2) \le 1$, we have $\Upsilon(t_2) \le \frac{1}{\alpha} \times \frac{1}{Pr(t_1)}\Upsilon(t_1)$.

Therefore, we conclude that no matter whether t_1 and t_2 are in the same generation rule or in different generation rules, we always have $\Upsilon(t_2) \le \frac{1}{\alpha} \times \frac{1}{Pr(t_1)}\Upsilon(t_1)$. This completes the proof. □

C. COMPUTATION OF $PR(S_{IJ})$

We adopt the idea of generating function in [7] for the computation of $Pr(S_{ij})$. Suppose tuple t_i is involved in the generation rule r_d. For each of the generation rules $r_1, r_2, ..., r_{d-1}, r_{d+1}, ..., r_l$, it divides the tuples involved in it into two parts. Some tuples have higher scores than t_i and others have lower or equal scores than t_i. We define b_{ih} as the sum of the probabilities of the tuples involved in $r_h (1 \le h \le l$ and $h \ne d)$ which have higher scores than t_i. Here we assume a virtual tuple has lower score than any real tuple. From the definition of b_{ih}, we know that $1 - b_{ih}$ is the sum of the probabilities of the tuples involved in r_h which have equal or lower scores than t_i. It is easy to see that we need $O(l\tau)$ time to compute all the b_{ih} for t_i. For each tuple t_i, we define a set $\theta_i = \{b_{ih}\}$ $(1 \le h \le l$ and $h \ne d)$.

Let us consider $\eta \in \Delta_i$. We assume $\eta = \{t_{s_1}, t_{s_2}, ..., t_{s_{d-1}}, t_i, t_{s_{d+1}}, ..., t_{s_l}\}$ where $t_{s_h} \in r_h$. We construct a generating vector $\gamma = \langle \gamma_1, \gamma_2, ..., \gamma_{d-1}, \gamma_{d+1}, ..., \gamma_l \rangle$ for η. If $score(t_{s_h}) > score(t_i)$, $\gamma_h = 1$; otherwise, $\gamma_h = 0$. Some tuple sets in Δ_i may have the same generating vector.

We assume γ is a generating vector for some tuple sets in Δ_i. We define Φ_γ as the set of all the tuple sets in Δ_i that have the generating vector γ. Let $Pr(\Phi_\gamma)$ be the sum of the probabilities of all the tuple sets in Φ_γ. It is easy to see that

$$Pr(\Phi_\gamma) = Pr(t_i) \prod_{h:\gamma_h=1} b_{ih} \prod_{h:\gamma_h=0} (1 - b_{ih})$$

We notice that all the tuple sets in Φ_γ belong to the same S_{ij}. S_{ij} is the union of some Φ_γ. Let $|\gamma|$ be the number of 1 in γ. The condition that Φ_γ belongs to S_{ij} is $|\gamma| = j$. We can compute $Pr(S_{ij})$ as follows.

$$Pr(S_{ij}) = Pr(t_i) \sum_{|\gamma|=j} \prod_{h:\gamma_h=1} b_{ih} \prod_{h:\gamma_h=0} (1 - b_{ih})$$

.

Let us see a function: $F(x) = \prod_{i=1}^{n}(a_i + b_i x)$. The coefficient of x^j in $F(x)$ is given by: $\sum_{|\beta|=j} \prod_{i:\beta_i=0} a_i \prod_{i:\beta_i=1} b_i$ where $\beta = \langle \beta_1, ..., \beta_n \rangle$ is a boolean vector.

Now consider the following generating function: $F^i(x) = Pr(t_i) \times \prod_{b \in \theta_i}(1 - b + b \times x) = \sum_{j=0}^{l-1} a_j x^j$.

We can see that the coefficient a_j of x^j in the expansion of $F^i(x)$ is $Pr(S_{ij})$. We can extend $F^i(x)$ to get c_j in $O(l^2)$ time. Therefore, for each tuple t_i, the complexity of computing $Pr(S_{ij})$ is $O(l^2 + l\tau)$. □

D. COMPLEXITY FOR PRF^ω

Recall from Section 4.2 that the time cost to compute the upper bound of a tuple is $O(\tau)$, where τ is the maximum number of real tuples in a generation rule.

Assume there are n tuples in T, and t_i is the i-th tuple, ordered by tuples' scores. In [7], the time cost of computing the PRF^ω value of t_i is $O(i\ log^2(i))$. The total time cost to find top-k tuples is $O(n^2 log^2 n)$, where n is the number of tuples in T. We can see that the cost of pruning for a tuple is much smaller than that of computing its PRF^ω value.

In the worst case in which no tuple is pruned, the time complexity of our (combined) algorithm is $O(n^2 log^2 n + n\tau)$ (since the complexity of finding top-k tuples is $O(n^2 log^2 n)$ and the cost of pruning for each tuple is $O(\tau)$). As the maximum value of τ is n, the time complexity of the combined algorithm remains the same as the one without pruning,

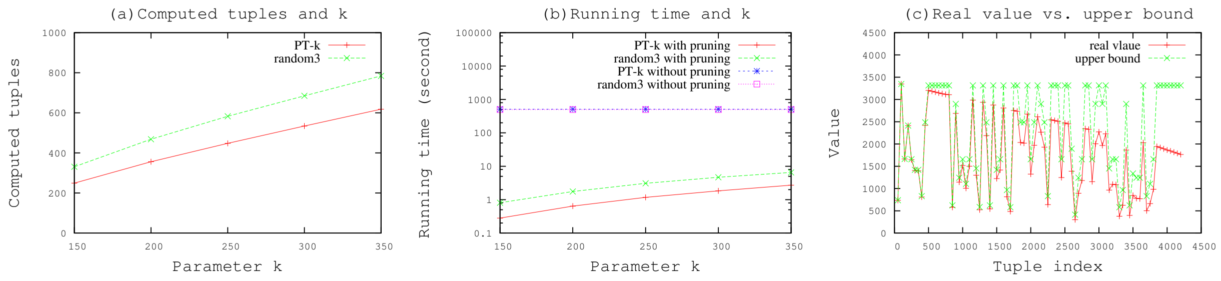

Figure 7: Experiments with real data

namely $O(n^2 log^2 n)$. So pruning does not reduce or increase the complexity of the given algorithm.

E. THEORETICAL COMPARISON

In [4], the concept of *probability threshold top-k query* (PT-k query) is introduced. It defines the top-k probability of a tuple t to be the sum of the probabilities of all the possible worlds in which t is one of the top-k tuples. Given a probability threshold p, the answer set of a PT-k query is the set of all tuples whose top-k probability values are at least p. Here let us consider the answer set of a PT-k query to be the set of k tuples with highest top-k probabilities.

In [4], some pruning techniques are proposed. PT-k query answer can be thought of as a special case of PRF^{ω} by writing the weight function of the PT-k query answer as follows:

$$\omega(i) = \begin{cases} 1 & \text{if } i \le k \\ 0 & \text{if } i > k \end{cases}$$

Therefore, our methods are applicable to PT-k query. In [4] two theorems (Theorems 3 and 4) are given for pruning. It can be shown that these theorems are special cases of our Theorems 4.1 and 4.2 but not conversely.

In fact, our upper bound theorems are much stronger than those in [4]. For the condition of conclusion (1) in Theorem 3 in [4] (the same condition appears in Theorem 4 in [4]), our theorems weaken the condition from $Pr^k(t) < p$ to $Pr^k(t) < \frac{Pr(t)}{Pr(t')} \times p$. Since $\frac{Pr(t)}{Pr(t')} > 1$, our condition covers more cases. Here $Pr^k(t)$ is the top-k probability of tuple t. One can think of $Pr^k(t)$ as $\Upsilon(t)$ of this paper, and take threshold p as the k-th highest top-k probability found so far in the combined algorithm of this paper. For conclusion (2) in Theorem 3 in [4], our theorems allow a tuple in a multi-tuple generation rule to compare directly with an independent tuple. This means that we can weaken the condition from $Pr(R) \le Pr(t)$ to $Pr(t'') < Pr(t)$, for any $t'' \in R$. We can also weaken the condition from $Pr^k(t) < p$ to $Pr^k(t) < \frac{Pr(t)}{Pr(t'')} \times p$. The new condition covers more cases.

Generally speaking, the theorems in [4] must satisfy two conditions for any two given tuples t and t': (1) $score(t) > score(t')$, and (2) $Pr(t) \ge Pr(t')$. But our theorems only require the first condition. As a result, our theorems cover more cases. Even when both conditions are satisfied, as we have already shown in the last paragraph, our theorems can prune better than the theorems in [4].

In [4], a global constraint is introduced for pruning. But the global constraint cannot be used for PRF^{ω} or PRF^{e}, due to the difference between the definition of PRF^{ω} and that of PT-k query answer.

F. EXPERIMENTS WITH REAL DATA

We use the data from International Ice Patrol (IIP) Iceberg Sighting Databases (http://nsidc.org/data/g00807.html). A real data set is generated from the iceberg sighting databases where each tuple contains the number of days drifted (score) and a confidence value (membership probability). The generated data set consists of 4232 tuples and 826 multi-tuple generation rules.

In Figure 7 the left two graphs show the computed tuples and running times for a RGWF random3 and PT-k query answer. We vary k from 150 to 350. The improvement is similar to the ones with the synthetic normal data sets.

We also conducted experiments to show the closeness of the PRF^{ω} values of tuples and their computed upper bounds using the method of this paper, for the real data set described in this appendix. We set $k = 50$ and use the weight function $\omega(i) = n - i$. To plot the graph (the last one in Figure 7), we pick one tuple from every 50 tuples (i.e., 51st, 101st,..., and so on; the upper bounds of the first k tuples need not be computed).

From the figure, we can see that for early retrieved tuples, the upper bounds are very close to their PRF^{ω} values. This shows that our pruning method finds very good upper bounds at the beginning. For later retrieved tuples, the distance between the upper bounds and the PRF^{ω} values becomes larger. Question arises as why the distance is larger while most later tuples are still pruned?

A short answer is that later tuples are easier to be pruned. In more detail, there are three factors to be considered. First, when more tuples are retrieved, the lowest PRF^{ω} value in the current top-k tuples becomes larger. With the bar raised it is easier to be lower. If an upper bound of a tuple is lower than this value, the tuple is pruned. Second, the graph in Figure 7 shows the trend of the PRF^{ω} values generally decreasing with the decrease of scores, for the data set being tested. So the later tuples in this case generally have lower PRF^{ω} values. (Note that this is not necessarily true for all data sets, e.g., for the special data sets described in Section 6.) This is why the distance becomes larger. Third, for later tuples, because most of the tuples retrieved before have been pruned and their PRF^{ω} values are not computed, the maintained t_{lowest}'s ratio between its PRF^{ω} value and membership probability is distanced from the possible lowest ratio among all retrieved tuples. So the computed upper bounds are not as tight. Observe that this is to say that when pruning is effective, the distance of the real values and computed upper bounds tends to become larger, and when pruning is ineffective, more PRF^{ω} values are computed and better t_{lowest}'s ratios are generated so to make later pruning more likely. This is an interesting self-adapting process.