Efficient Rank Based KNN Query Processing Over Uncertain Data

Ying Zhang, Xuemin Lin, Gaoping Zhu, Wenjie Zhang, Qianlu Lin

The University Of New South Wales

{yingz, lxue, gzhu, zhangw, qlin}@cse.unsw.edu.au

***Abstract*— Uncertain data are inherent in many applications such as environmental surveillance and quantitative economics research. As an important problem in many applications, KNN query has been extensively investigated in the literature. In this paper, we study the problem of processing rank based KNN query against uncertain data. Besides applying the *expected rank* semantic to compute KNN, we also introduce the *median rank* which is less sensitive to the outliers. We show both ranking methods satisfy nice top-k properties such as *exact-k*, *containment*, *unique ranking*, *value invariance*, *stability* and *fairfulness*. For given query q, IO and CPU efficient algorithms are proposed in the paper to compute KNN based on expected (median) ranks of the uncertain objects. To tackle the correlations of the uncertain objects and high IO cost caused by large number of instances of the uncertain objects, randomized algorithms are proposed to approximately compute KNN with theoretical guarantees. Comprehensive experiments are conducted on both real and synthetic data to demonstrate the efficiency of our techniques.**

I. Introduction

Managing uncertain data has been extensively studied since 1980s by the database community. Recently, it has drawn more research attention with many emerging applications where the underlying datasets are uncertain in nature. The causes of uncertain data vary greatly from one application to another. In moving objects tracking [1], the location information of objects collected by a GPS system may not be exact due to the delay on data updates. When predicting habitat trends [2], collected sensor data such as temperature, humidity are imprecise due to the limitation of measuring devices. Besides, data randomness and incompleteness, even privacy preservation can also be the causes of data uncertainty. It is an important task for the database community to efficiently manage uncertain data. A number of issues have already been addressed which include modeling [3] and querying uncertainty [4].

The KNN (K Nearest Neighbor) query has been widely studied and used in applications including location based services, traffic network analysis, sensor network and facial pattern recognition. However, the natural data uncertainty in these applications leads to imprecise results. For example, in anti-terrorist missions where team members are equipped with GPS devices, the location of each member is periodically transmitted to the server. When one team member or more are ambushed by terrorists, assistance is called by sending request to the server which may issue a query for the 10 nearest members to the ambushed member(s). As location information is not exact, a team member may satisfy the query partially. As mentioned in [5], another application exists in locating accident witnesses. Assuming all witnesses are mobile users, as their locations collected from the mobile devices are uncertain, it is natural to find a set of k witnesses who are most likely to be the k nearest ones to a query point representing an accident location.

As shown above, the KNN query over uncertain data is very important in many real life applications. Even so, only a few work has been reported on this problem. In [5], [6], [7], the NN query over uncertain data is defined as a PNN (Probabilistic Nearest Neighbor) based query, which ranks uncertain objects based on their probabilities of being the nearest neighbor of a query q. Although k uncertain objects with the highest probabilities are returned as top-k answers, the semantic is inherently different from that of the traditional KNN query. For instance, suppose there is an object whose possible locations are very close the to the query point but it never appears as NN under all possible worlds [8], it will never be returned as top-k answer regardless the value of k.

The KNN query over uncertain data is studied in [9], [10], in which the ranking of uncertain objects follows the semantic of *expected score* [11] and U-top$k$ [8] respectively. Specifically, in [9], the expected distance (under L1 norm) is employed to find KNN. And in [10], following the semantic of U-topk, authors aim to find sets of uncertain objects such that in each set there are k uncertain objects and the probability of these uncertain objects being k nearest neighbor is larger than a given probabilistic threshold.

In [11], five important properties including *exact-k*, *containment*, *unique ranking*, *value invariance* and *stability* are proposed to evaluate the "goodness" of the top-k semantics. We will introduce these properties in Section II. Since the KNN query over uncertain objects can be naturally regarded as a top-k query on attribute level where the possible scores of an object are distances from its possible locations to the given query point, the semantic used to compute KNN should satisfy those properties. As shown in [11], ranking on *expected score* is sensitive to the values thus violates the *value invariance* property, while U-top$k$ does not meet the *exact-k* and *containment* properties.

This motivates us to apply the *expected rank* semantic proposed in [11] to rank uncertain objects for KNN queries as it is the only existing top-k semantic which meets all those properties. As discussed in Section III, directly applying the techniques in [11] does not lead to efficient solutions for the KNN query over uncertain data. So in this paper we aim to develop IO and CPU time efficient algorithm to find *expected rank* based KNN. As discussed in [11], compared with mean, median is less sensitive to outliers. In this paper, we also investigate the use of *median rank* for the KNN query which is shown to be more IO efficient than *expected rank*. Note that the semantic of our KNN query is different with that of [9] and [10]. Moreover, since [9] only considers the L1 norm distance and [10] assumes the distribution of the distance between a query point and uncertain object is available, their techniques can not be applied in our work.

As a promising approach to tackle the complicated correlations among the uncertain data, the Monte-Carlo simulation has been applied in recent works [12], [13], [14]. Assuming the samples based on popular Markov Chain Monte-Carlo (MCMC) method are available, we develop efficient randomized algorithm to approximately compute KNN. Even in applications without data correlations, our randomized algorithm can achieve a significant performance improvement on the KNN query, especially when possible locations of an object are massive or the query is uncertain.

Contributions. Our contributions in this paper can be summarized as follows.

- We formally introduce the problem of rank based KNN query over uncertain data in which the *expected rank* and *median rank* are considered. We further show that ranking on *expected rank* or *median rank* for KNN query satisfies important top-k properties.
- Two important techniques [15], [16] in general selection problem are thoroughly compared in Section II.
- We develop efficient exact and randomized algorithms for processing both *expected rank* and *median rank* based KNN query.
- Comprehensive experiments are conducted on both real and synthetic data to demonstrate the efficiency of our techniques.

Organization of the paper. The rest of the paper is organized as follows. Section II formally defines the problem of rank based KNN query over uncertain objects, while preliminary knowledge is also presented in details. Section III and Section IV present the exact and randomized algorithms for rank based KNN query respectively. The experimental results are reported in Section VI. This is followed by the related work presented in Section VII. We conclude our paper in Section VIII.

II. Preliminary

In this section, we formally define the problem of rank based KNN query. Then we show both *expected rank* and *median rank* based KNN query stratify the *exact-k*, *containment*, *unique ranking* and *value invariance* properties. With independence assumption, they also satisfy *stability* and *fairfulness* properties. This is followed by a comparison between two important techniques [15], [16] used for selection problem and some theoretical foundations used in the paper. Table I below summarizes the mathematical notations used throughout this paper.

A. Problem Definition

Suppose the possible location of an uncertain object or a query, by default, is in a d-dimensional numerical space, the distance between two points x and y is denoted by $\parallel x - y \parallel$.

TABLE I
THE SUMMARY OF NOTATIONS.

Notation	Definition
U, V	uncertain objects
u, v	instances of uncertain objects
n	number of uncertain objects
m	number of instances in an uncertain object
$e(I)$	the aR-tree entry associated with interval I
l (r)	lower(upper) bound for validating(pruning) objects
$er(U)$ ($mr(U)$)	*expected* (*median*) rank of object U
I ($\mathcal{I}$)	interval (set of intervals)
$d(U)$ ($d(u)$,$d(I)$)	distance of uncertain object(instance, interval)
$r(u)$ ($r(I)$)	rank value of instance(interval)
$U_{rmin}(U_{rmax})$	minimal(maximal) possible rank of object U

The *Euclidean Distance* is employed as the distance metric in this paper. Nevertheless, our technique can be easily extended to other distance metrics.

An uncertain object can be described either *continuously* or *discretely*. In the *continuous* case, an uncertain object U is described by its PDF (Probability Density Function) $U.pdf$ and uncertain region U_r. The appearance probability of an instance $x \in U_r$ is $U.pdf(x)$ and $\int_{x \in U_r} U.pdf(x)dx = 1$. In the *discrete* case, an uncertain object U consists of a set of instances $\{u_1, \ldots, u_m\}$ where u_i appears with a probability p_{u_i} and $\sum_{u \in U} p_u = 1$. Practically, $U.pdf$ is often explicitly unavailable and approximated by a set of sampled instances. Although this paper mainly discusses the *discrete* case, we briefly discuss how to apply our techniques to continuous case in Section V. We assume the uncertain objects are independent of each other, if not explicitly specified.

Remark 1: The query in this paper might be a point or an uncertain object. we assume query is a point in the sequel and Section V discusses how to tackle the rank based KNN problem with uncertain query locations. Meanwhile, we use "object" to represent "uncertain object" whenever there is no ambiguity.

Given a set of objects $\mathcal{U} = \{U_1, U_2, \ldots, U_n\}$, a possible world $W = \{u_1, u_2, \ldots, u_n\}$ is a set of instances sequentially sampled for each object. The probability of W to appear is $Pr(W) = \prod_{i=1}^{n} p_{u_i}$. Let $\mathcal{W}$ be the set of all possible worlds, then $\sum_{W \in \mathcal{W}} Pr(W) = 1$. For a given W, the rank of U in W regarding a query q, denoted by $Rank_q^W(U)$, is the number of other objects which are closer to q than U. Let U^W denote the instance of U in W, following is a formal definition.

$$Rank_q^W(U) = |\{U_j| \parallel U_j^W - q \parallel < \parallel U^W - q \parallel, \ U \neq U_j\}| \quad (1)$$

Then we define the rank of an instance $u \in U$ as follows,

$$r_q(u) = \frac{1}{p_u} \sum_{U^W = u} Rank_q^W(U) \times Pr(W) \quad (2)$$

Based on the definition of $Rank_q^W(U)$, we have the following definition of *expected rank*.

Definition 1 (expected rank): Given a set of objects $\mathcal{U}$ and a query q, the *expected rank* of an object U, denoted by $er_q(U)$, is defined as follows.

$$\begin{aligned} er_q(U) &= \sum_{W \in \mathcal{W}} Rank_q^W(U) \times Pr(W) \quad (3) \\ &= \sum_{u \in U} r_q(u) \times p_u. \quad (4) \end{aligned}$$

Before the definition of *median rank*, we first define the *median* of a set of weighted elements.

Definition 2 (median): Given a set of n elements $\{e_i | i \in [1, n]\}$ each of which is associated with a value $v(e_i)$ and a weight $w(e_i)$ where $\sum_{i=1}^{n} w(e_i) = w$, suppose the elements are sorted on their values with non-decreasing order, then the median($\{e_i | i \in [1, n]\}$) is the value of element e_j such that j is the smallest integer such that $\sum_{i=1}^{j} w(e_i) \geq \lfloor \frac{w}{2} \rfloor$.

Then based on Equation 4, we have the definition of *median rank* of object U, denoted by $mr_q(U)$.

Definition 3 (median rank): Given a set of objects $\mathcal{U}$ and a query q, for each $U \in \mathcal{U}$:

$$mr_q(U) \quad = \quad median(u_j | u_j \in U\}) \tag{5}$$

where $v(u_j) = r_q(u_j)$ and $w(u_j) = p_{u_j}$.

Remark 2: For presentation simplicity, whenever there is no ambiguity, we use $r(u)$, $er(U)$ and $mr(U)$ to denote the rank of an instance, the *expected rank* and *median rank* of an object regarding a query q respectively. Similarly, $d(u)$ is employed to denote the distance $\parallel u - q \parallel$.

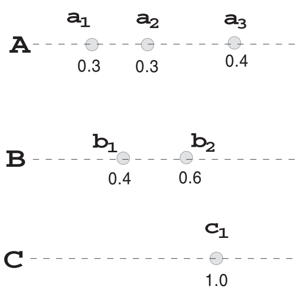


Fig. 1. Example

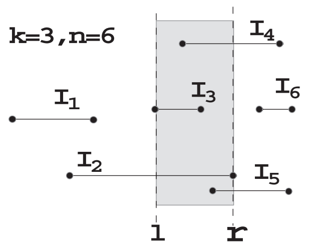


Fig. 2. Bound Based Approach

Example 1: In Figure 1, the instances of three objects A, B and C are sorted on their distances to the query q, the appearance probability of each instance is presented as well. There are 6 possible worlds : $W_1(a_1, b_1, c_1)$, $W_2(a_1, b_2, c_1)$, $W_3(b_1, a_2, c_1)$, $W_4(a_2, b_2, c_1)$, $W_5(b_1, c_1, a_3)$, $W_6(b_2, c_1, a_3)$ with probability 0.12, 0.18, 0.12, 0.18, 0.16 and 0.24 respectively. For A, we have $r(a_1) = 0, r(a_2) = 0.4$ and $r(a_3) = 2$, then $er(A) = 0.4 \times 0.3 + 2 \times 0.4 = 0.92$ and $mr(A) = 0.4$. Similarly, we have $er(B) = 0.48$, $mr(B) = 0.6$, $er(C) = 1.6$ and $mr(C) = 1.6$.

Problem Statement. In this paper, we investigate the problem of finding k nearest neighbors for a given query q based on the *expected (median) ranks* of n objects. Without loss of generality, we assume $n > k$ and ties are broken by object ids.

B. Top-k Properties

In this subsection, we first briefly introduce the *exact-k*, *containment*, *unique ranking*, *value invariance*, *stability* and *fairfulness* properties [17], [11] in the context of KNN query. Then Theorem 1 shows both *expected rank* and *median rank* meet those properties.

- *exact-k*. KNN query returns exactly k objects.
- *containment*. (K+1)NN should contain all objects in KNN.
- *unique ranking*. The same object should not be listed multiple times in KNN.
- *value invariance*. The distance only determines the relative behavior of the object: changing the distance of an instance to q without altering the relative order should not change the KNN.
- *stability*. For object X in KNN, becoming "probabilistically no further" will not turn X into a non-KNN objects. Similarly, for object Y which is not in KNN, Y will not be KNN if it becomes "probabilistically no closer".
- *fairfulness*. If an object X is "probabilistically closer" than another object Y, $X \in$ KNN if $Y \in$ KNN.

Following the concept of "stochastic order"[18], "probabilistically closer" is defined as follows.

Definition 4: Given two objects X, Y and a query q, X is probabilistically closer than Y, denoted by $X \prec_q Y$, if and only if for any value $t \in (-\infty, \infty)$, $Pr(d(X) < t) > Pr(d(Y) < t)$, where $Pr(d(X) < t) = \sum_{u \in X \ and \ d(u) < t} p_u$. Similarly, we have "$X$ is probabilistically no further than Y" and "X is probabilistically no closer than Y", denoted by $X \preceq_q Y$ and $Y \preceq_q X$ respectively.

With independence assumption, for an instance u of an object U, we have

$$r(u) = \sum_{v \in \mathcal{U} - U, d(v) < d(u)} p_v \tag{6}$$

Following lemma immediately holds based on Equation 6. Note that it might not hold for instances from different objects.

Lemma 1: Given an object U and a query q, for any two instances $u_1, u_2 \in U$, we have $r(u_1) \leq r(u_2)$ if $d(u_1) < d(u_2)$ and $r(u_1) = r(u_2)$ if $d(u_1) = d(u_2)$.

Similar with the definition of *median rank*, we define the *median distance* of an object U, denoted by $md(U)$, as follows.

Definition 5 (median distance): $md(U) = median$ $(\{u_j | u_j \in U\})$, where $v(u_j) = d(u_j)$ and $w(u_j) = p_{u_j}$.

Based on Definition 5 and Lemma 1 we have the following lemma, which implies that the instance contributes to the *median rank* of an object must have the same rank value as the instance contributes to the *median distance*.

Lemma 2: Let u_i and u_j be the instances which determine the *median rank* and *median distance* of U respectively, we have $r(u_i) = r(u_j)$.

Following theorem shows that the *expected rank* and *median rank* satisfy the *exact-k*, *containment*, *unique ranking*, *value invariance* and *stability* properties.

Theorem 1: Both *expected rank* and *median rank* satisfy *exact-k*, *containment*, *unique ranking* and *value invariance*. With independence assumption, they also satisfy *stability* and *fairfulness*.

Proof: For presentation simplicity, we assume the *expected (median) rank* is unique (ties are broken arbitrarily) so that the ranking forms a total ordering. For a given query q, an object U can be regarded as a random variable with possible values $\{d(u) | u \in U\}$ where u happens with probability p_u. Following the proof of Theorem 1 in [11], it is immediate that the *expected rank* satisfies *exact-k*, *containment*, *unique ranking*, *value invariance* and *stability*. We also have that the *expected rank* meets *fairfulness* with independence assumption.

For the *median rank*, the correctness of *exact-k*, *containment* and *unique ranking* are immediate based on its definition and the unique assumption. As the *median rank* of U is computed based on the ranks of its instances which naturally support the *value invariance* property, the *value invariance* follows. For an object U in KNN, suppose another object $U^{\uparrow}$ is "probabilistically no further" than U. Let ζ be the smallest value such that $Pr(U < \zeta) \geq 0.5$, then we have $Pr(U^{\uparrow} < \zeta) \geq 0.5$ according to Definition 4, which implies $md(U) \geq md(U^{\uparrow})$. Let u and u' be the instances in U and $U^{\uparrow}$ with $d(u) = md(U)$ and $d(u') = md(U^{\uparrow})$, we have $d(u') \leq d(u)$. According to Lemma 2, we have $r(u) = mr(U)$ and $r(u') = mr(U^{\uparrow})$. If we use $U^{\uparrow}$ to **replace** U, then $r(u') \leq r(u)$ since $\mathcal{U} - U = \mathcal{U} - U^{\uparrow}$ in Equation 6. So the *median rank* satisfies the *stability*. With similar rationale, we have that the *median rank* also satisfies *fairfulness* under independence assumption. ■

C. Finding Minimal Set for Selection Problem

The selection problem is to find the k-th smallest element among a set of n elements, where each element e is associated with an interval $I_e = [l(I_e), r(I_e)]$ to represent its minimal/maximal possible value. A cost $c(e)$ is required to retrieve the exact value of e. One of the most important subroutines of the problem is to find the minimal set of intervals which might contain the k-th smallest value. Currently, there are two kinds of approaches to find the minimal set.

In [15], Khanna *et. al* proposed an *interval graph* based approach to find the minimal set by partitioning all the intervals into t groups such that any two intervals among the same group do not *overlap* each other, where t is the largest number of intervals that pairwise overlap with one another. A nice property of this approach is that the candidate set only consists of at most $2 \times t$ intervals. Although there is a polynomial-time algorithm [19] to partition intervals into minimal t groups, the time cost of this approach is high especially when intervals are dynamically "refined".

Another approach is called the bound based approach which is described by the following lemma [16].

Lemma 3: Given an interval set $\{I = [l(I), r(I)]\}$, the k-th smallest element must lie in the interval $I_c = [l, r]$, where l is the k-th smallest $l(I)$ and r is the $(n-k+1)$-th largest $r(I)$. Then any interval I with $l(I) > r$ or $r(I) < l$ can not contain the k-th smallest element.

As shown in Figure 2 where $l = l(I_3)$ and $r = r(I_2)$, only intervals I_2, I_3, I_4 and I_5 might contain the 3-rd smallest element. Similar to the computation of median [20], l and r can be computed in $O(n)$ time where n is the number of intervals. Moreover, with two priority queues, l and r can be efficiently maintained with cost $O(\log k)$ when the left or right point of an interval is updated.

In the following theorem, we show that the bound based approach also outperforms the *interval graph* based one in terms of candidate set size. Consequently, we can safely use the bound based approach in this paper.

Theorem 2: Given a set $\mathcal{I}$ of n intervals and k, the candidate set obtained by applying the bound based approach always has smaller or equal size compared with that of the *interval graph* based approach.

Proof: We first show the candidate set has at most $2 \times t$ intervals where t is the largest number of intervals that pairwisely *overlap* one another. Let L, M and R denote intervals $(-\infty, l)$, $[l, r]$ and $(r, +\infty)$ respectively. L_l and L_r are employed to represent the number of left and right end points of intervals falling in L respectively. Similar definitions go to M_l, M_r, R_l and R_r. Regarding to the example in Figure 2, we have $L_l = 2$, $L_r = 1$, $M_l = 3$, $M_r = 2$, $R_l = 1$ and $R_r = 3$. According to Lemma 3, we have $L_l = k-1$, $R_r = n-k$ and $L_r + M_r = k$. Let $\mathcal{J}$ denote the intervals with $l(I) < l$ and $r(I) \geq l$. Clearly, we have $|\mathcal{J}| \leq t-1$ since all $I \in \mathcal{J}$ *overlap* each other. This implies $L_r \geq k-t$ as each left point in L must match a right point. Since $L_r + M_r = k$, we have $M_r \leq t$. Let $\mathcal{S}$ and $\mathcal{R}$ denote the interval sets with $r(I) \leq r$ and $r(I) > r$ respectively. It is immediate that $|\mathcal{S}| + |\mathcal{R}| = n$. For each interval $I \in \mathcal{S}$, I is a candidate if and only if $r(I) \geq l$. So there are at most t of them included since $M_r \leq t$. For each interval $I \in \mathcal{R}$, I is a candidate if and only if $l(I) \leq r$. Consequently there are at most t of them since all $I \in \mathcal{R}$ being candidates *overlap* each other. So the bound based approach has the same worst case guarantee as that of the *interval graph* based approach.

Let $\bar{\mathcal{S}}$ and $\bar{\mathcal{R}}$ denote the intervals in the candidate set with $r(I) \leq r$ and $r(I) > r$ respectively. With similar rationale in the above proof, we have $|\bar{\mathcal{S}}| \leq |\mathcal{G}|$ and $|\bar{\mathcal{R}}| \leq |\mathcal{F}|$, where $|\mathcal{G} \cup \mathcal{F}|$ is the candidate set defined in [15]. Due to space limitation, we omit the detailed proof. ■

Similar to [21], Lemma 3 can be easily applied to find the minimal candidate set for the ϕ-quantile [22] computation [1] where each element is represented by an interval.

Lemma 4: Given $0 \leq \phi \leq 1$ and a set of intervals $\mathcal{I}$ with $\sum_{I \in \mathcal{I}} w(I) = 1.0$. Suppose all intervals are firstly sorted on $l(I)$ with non-decreasing order, let I_j be the first interval with $\sum_{i=1}^{j} w(I_i) \geq \phi$, we have $l = l(I_j)$. Similarly, we have $r = r(I_j)$ where I_j is the first interval with $\sum_{i=1}^{j} w(I_i) \geq 1 - \phi$ when all intervals are sorted on $r(I)$ with non-increasing order. Then any interval I with $r(I) < l$ or $l(I) > r$ could not contain the ϕ-quantile element. The size of candidate set is at most $(1 + \frac{w_u}{w_l}) \times t$, where w_l and w_u are the minimal and maximal w of the intervals.

The correctness of the Lemma 4 is immediate based on Lemma 3 and Theorem 2 by considering each interval I as a set of intervals with the same left and right points.

During the KNN computation, we only need to compute the *expected rank* or *median rank* for the objects which can not be *validated* or *pruned* based on their possible maximal and minimal ranks. According to [12] and Theorem 2, we use the following lemma to *prune* or *validate* objects for KNN query.

Lemma 5: For given intervals $\{I\}$ and value k, let l be the $(k+1)$-th smallest $l(I)$ and r denote the $(n-k+1)$-th largest $r(I)$. Then any interval I with $r(I) < l$ must contain element in the top-k list and hence can be *validated*. If $l(I) > r$, the

[1] Median in Definition 2 is a special case of ϕ-quantile with $\phi = 0.5$

element associated with the interval can be safely *pruned* from the top-k list. The number of objects left is at most $2 \times t$ where t is the largest number of intervals that pairwisely *overlap* one another.

III. Exact Algorithm

This Section presents efficient algorithms to retrieve k objects with the highest *expected (median) ranks* for a given query q. We first introduce the motivation of our techniques in Section III-A. Section III-B presents the details of our KNN algorithms, followed by detailed performance analysis in Section III-C.

A. Motivation

For an object U and a query q, let $d^-(U)$ and $d^+(U)$ denote the minimal and maximal possible distance between instances of U and q respectively, which can be computed based on the uncertain region of U. Following lemma indicates that we can prune some objects without exploring their individual instances. The correctness of the lemma is immediate based on Lemma 5 and the definition of the *expected (median) rank*.

Lemma 6: For a given query q, let d_k be the k-th largest d^+ value for all the objects. Any object U with $d^-(U) > d_k$ can not be KNN of q, while the other objects form a candidate set. Let d_f denote the largest d^+ value for objects in the candidate set, then any object U with $d^-(U) > d_f$ can be excluded from the KNN computation.

For a given query q, we can prune objects based on Lemma 6, then apply the exact computation algorithm proposed in [11] to compute the *expected rank* based KNN [2], in which the scores of an object are distances between its instances and query q. However, this approach might suffer from the expensive IO cost when the number of instances or the size of uncertain region is large. The following lemma provides a detailed analysis under 2-dimensional space with similar argument in [23].

Lemma 7: Assume the domain sizes of all dimensions are between $[0, 1]$ and the uncertain regions are circles with radius r_u, if the centres of objects and their instances follow the uniform distribution, the expected candidate size is $n \times \pi \times (\sqrt{\frac{k}{\pi n}} + 4 \times r_u)^2$.

Proof: Under the uniform assumption, the object U with the k-th smallest maximal distance has $d^+(U) = d_k = \sqrt{\frac{k}{\pi n}} + r_u$; then we have $d_f = \sqrt{\frac{k}{\pi n}} + 3 \times r_u$. Any object U with $d_c(U) > d_f + r_u$ will be pruned where $d_c(U)$ is the distance from its centre to query. So the expected number of objects left is $n \times \pi \times (\sqrt{\frac{k}{\pi n}} + 4 \times r_u)^2$. ■

Although [11] provides a pruning technique based on the pre-computed expected scores of objects to reduce the number of objects accessed, it can not be applied to KNN computation as the expected score of an object can not be pre-computed since it might change upon different queries. Moreover, only the approximate KNN are returned by the pruning technique.

This motivates us to develop new CPU and IO efficient algorithms to compute the rank based KNN for a given query q. We assume the uncertain regions of objects are indexed by a R-tree, named *global* R-tree, which is denoted by gR. As there is no assumption on the distribution of the instances, we can not derive the distribution of $\{d(u)\}$ for object U where $u \in U$, except that $d^-(U) \leq d(u) \leq d^+(U)$. To avoid loading all instances of an object during the computation, we assume its instances are organized by an aggregate R-tree, named *local* aR-tree. As illustrated in Figure 3(a), 5 objects are indexed by 5 *local* aR-tree and the roots of these aR-trees (uncertain regions) are organized by a *global* R-tree. For a query q, an entry $e(I)$ of the *global* or *local* aR-tree can be treated as an interval I within which each instance has a possible distance from q between $[d^-(I), d^+(I)]$. Note that $d^-(I) = \| e - q \|_{min}$ and $d^+(I) = \| e - q \|_{max}$. Each interval I can be further represented by several child intervals each of which are created for a corresponding child entry of e. Note that all intervals in Figure 3(b) and (c) are ordered on their distances to q.

In the KNN computation, since an interval I can be regarded as the representative of a group of instances, the weight of I, denoted by $p(I)$, is the aggregate value of appearance probabilities of these instances. Following Equation 6, we define the minimal/maximal rank of an interval I of object U, denoted by $r^-(I)$ and $r^+(I)$ respectively, as follows:

$$r^-(I) = \sum_{I' \in \mathcal{U} - U,\ d^+(I') < d^-(I)} p(I') \tag{7}$$

$$r^+(I) = \sum_{I' \in \mathcal{U} - U,\ d^-(I') \leq d^+(I)} p(I') \tag{8}$$

As shown in Figure 3(c), r^- and r^+ of $I_{B,3}$ is 1.2 and 2.5 respectively, where $I_{B,3}$ represents the interval of B with id 3. Let $\mathcal{I}_U$ be the set of all intervals of U. For any instance $u \in U$, u belongs to one and only one interval $I \in \mathcal{I}_{\mathcal{U}}$. Clearly, we have $\sum_{I \in \mathcal{I}_U} p(I) = 1$. Then the minimal and maximal *expected rank* of U, denoted by $er^-(U)$ and $er^+(U)$ respectively, can be computed as follows:

$$er^-(U) = \sum_{I \in \mathcal{I}_U} r^-(I) p(I) \tag{9}$$

$$er^+(U) = \sum_{I \in \mathcal{I}_U} r^+(I) p(I) \tag{10}$$

Similarly, we have

$$mr^-(U) = median^-(\{I' = [r^-(I), r^+(I)] | I \in \mathcal{I}_U\}) \tag{11}$$

$$mr^+(U) = median^+(\{I' = [r^-(I), r^+(I)] | I \in \mathcal{I}_U\}) \tag{12}$$

where $median^-$ and $median^+$ denote the minimal and maximal possible value of $mr(U)$ which is computed based on Lemma 4 with $\phi = 0.5$.

For any instance $u \in U$, u belongs to one and only one $I \in \mathcal{I}_U$. As $r^-(I) \leq r(u) \leq r^+(I)$, the correctness of Equations 9, 10, 11 and 12 is immediate.

Remark 3: For object U, we use U_{rmin} to denote $er^-(U)$ and $mr^-(U)$ whenever there is no ambiguity. U_{rmax} is defined in a similar way.

The following example demonstrates how to compute the lower and upper bounds for the *expected rank* and *median rank* of an object.

Example 2: According to above definitions, for object B in Figure 3(c), $\mathcal{I}_B = \{I_{B,1}, I_{B,2}, I_{B,3}\}$, and we have $er^-(B) =$

[2]It can be easily extended to compute *median rank* based on its definition

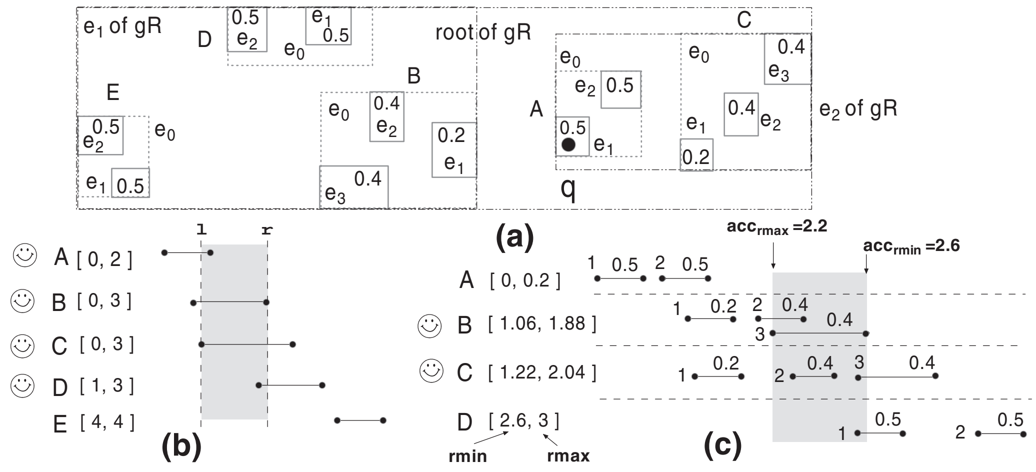


Fig. 3. KNN Example (k=2)

$\sum_{i=1}^{3} r^-(I_{B,i}) \times p(I_{B,i}) = 0.5 \times 0.2 + 1.2 \times 0.4 + 1.2 \times 0.4 = 1.06$ and $er^+(B) = 1.2 \times 0.2 + 1.6 \times 0.4 + 2.5 \times 0.4 = 1.88$. As $median^-(\mathcal{I}_B) = r^-(I_{B,2})$ and $median^+(\mathcal{I}_B) = r^+(I_{B,2})$, we have $mr^-(B) = 1.2$ and $mr^+(B) = 1.6$.

According to the above Equations, we can refine the *expected rank* and *median rank* of objects by carefully visiting entries of their *local* aR tree from high levels to low levels.

B. KNN Algorithm

In this subsection, we introduce an efficient KNN algorithm whose main body is presented in Algorithm 1. The procedure **Initialize** is employed to compute the candidate objects based on Lemma 6, and our implementation is based on the *global* R-tree and a min heap. Due to space limitation, we omit the details. The candidate set $\mathcal{I}$ consists of intervals created for the candidate objects. The procedure **Sweepline** is invoked to efficiently compute the ranks of objects and update l and r. The objects might be *pruned* or *validated* during the computation, while the procedure **ProbeIntervals** is called to probe the intervals in a *level-by-level* fashion. Followings are the details of **Sweepline** and **ProbeIntervals**.

Algorithm 1: ***KNN***(gR, q, k)

```
Input  : $gR$ : the global $R$-Tree, $q$ : query, $k$
Output : $\mathcal{O}$ : KNN of $q$
1 $\mathcal{O} = \emptyset$; $\mathcal{I}$ := Initialize ($gR$, $q$, $k$);
2 while $|\mathcal{O}| < k$ do
3     Sweepline($\mathcal{I}$);
4     if $|\mathcal{O}| < k$ then
5         $\mathcal{I}$ := ProbeIntervals($\mathcal{I}$);
6 return $\mathcal{O}$
```

Sweepline Algorithm Algorithm 2 illustrates the details of the procedure **Sweepline**. For presentation simplicity, we only consider *expected rank* in Algorithm 2. The computation of *median rank* will be discussed later.

For each object, there is a flag to indicate whether it is *active*. An object is *active* if and only if it is not *pruned* or *validated* by l and r according to Lemma 5. Obviously, we only need to compute the ranks for *active* objects.

Algorithm 2: ***Sweepline***($\mathcal{I}$, q, k) [*expected rank* version]

```
Input : $\mathcal{I}$ : interval set; $q$ : query; $k$
1 $acc_{rmin}$ := 0; $acc_{rmax}$ := 0; ;
2 for all active object $U$ do
3     $U_{rmin} := U_{rmax} := U_{ar_{min}} := U_{ar_{max}} := 0$;
4 $\mathcal{P}$ := start and end points of all $I \in \mathcal{I}$ ;
5 sort $\mathcal{P}$ by $d(p)$ with non-decreasing order;
6 for each $p \in P$ accessed in sequence do
7     $U := p.obj$; $I := p.I$;
8     if $p$ is an end point then
9         $acc_{rmin} := acc_{rmin} + p(I)$;
10        if $U$ is active then
11            $U_{rmax} := U_{rmax} + (acc_{rmax} - U_{ar_{max}}) \times p(I)$ ;
12            $U_{ar_{min}} := U_{ar_{min}} += p(I)$;
13            if $U_{ar_{min}} = 1.0$ then
14                Update $l$ and $r$;
15                if $U_{rmax} < l$ or $U_{rmin} > r$ then
16                    set $U$ inactive ;
17                    $\mathcal{O} := \mathcal{O} \cup U$ if $U_{rmax} \leq r$;
18    else
19        $acc_{rmax} := acc_{rmax} + p(I)$ ;
20        if $U$ is active then
21            $U_{rmin} := U_{rmin} + (acc_{rmin} - U_{ar_{min}}) \times p(I)$ ;
22            $U_{ar_{max}} := U_{ar_{max}} + p(I)$;
```

Let $acc_{rmin}(d)$ denote the accumulation of the probability values of the intervals $\{I \in \mathcal{I}\}$ with $d^+(I) \leq d$ and $U_{ar_{min}}(d)$ represent the accumulation of the probability values of intervals $\{I \in \mathcal{I}_U\}$ with $d^+(I) \leq d$. Recall that $\mathcal{I}_U$ is a set of intervals representing instances of U, based on Equation 7, we have $r^-(I) = acc_{rmin}(d) - U_{ar_{min}}(d)$ where $d = d^-(I)$. Similar definitions go to acc_{rmax} and $U_{ar_{max}}$, then we have $r^+(I) = acc_{rmax}(d) - U_{ar_{max}}(d)$ where $d = d^+(I)$.

In order to efficiently compute the ranks for *active* objects, we use a *sweep line paradigm*. As shown in Line 4 and 5 of Algorithm 2, we sort the start and end points of the intervals in $\mathcal{I}$ where start(end) point has distance value $d^-(I)$ ($d^+(I)$). For a point p, its related object and interval are denoted by $p.obj$ and $p.I$ respectively. Then the rank of *active* objects can be accumulatively computed and the details are presented in

Line 6-22. As shown in Line 13-17, once we finish computing the rank of an object, the l and r will be updated and the flag of the object is set to *inactive* if it is *pruned* or *validated*.

Algorithm 2 can be easily modified to support the *median rank*. At Line 11, we have $U_{rmax} = acc_{rmax} - U_{ar_{max}}$ if $U_{ar_{min}} < 0.5$ and $U_{ar_{min}} + p(I) \geq 0.5$. At Line 21, $U_{rmin} = acc_{rmin} - U_{ar_{min}}$ if $U_{ar_{max}} < 0.5$ and $U_{ar_{max}} + p(I) \geq 0.5$. The correctness is immediate based on Lemma 4.

The following example shows some details of the computation of *expected rank* and *median rank*.

Example 3: Figure 3(b) and (c) illustrate the results of **Sweepline** procedure at the first and second rounds. The objects with happy face are *active* objects. Particularly, when $d = d^-(I_{B,3})$ we have $acc_{rmax} = 2.2$, $acc_{rmin} = 1.4$, $B_{ar_{max}} = 1.0$ and $B_{ar_{min}} = 0.2$. So $r^-(I_{B,3})$=$acc_{rmin} - B_{ar_{min}} = 1.2$. Similarly, we have $acc_{rmax} = 3.5$, $acc_{rmin} = 2.6$, $B_{ar_{max}} = 1.0$, $B_{ar_{min}} = 1.0$ and $r^+(I_{B,3}) = 2.5$ when $d = r^+(I_{B,3})$. To compute the *median rank* of B, in Figure 3(b) we have $mr^-(B) = r^-(I_{B,2}) = 1.2$ and $mr^+(B) = r^+(I_{B,2}) = 1.6$.

The time complexity of Algorithm 2 is $O(n_{\mathcal{I}} \log n_{\mathcal{I}})$. This is because the sorting at Line 5 takes time $O(n_{\mathcal{I}} \log n_{\mathcal{I}})$, while the update cost of l and r is $O(n_{\mathcal{I}} log(k))$ as two priority queues of size k are employed in our implementation. Clearly, the time cost is the same as the modified Algorithm 2 for *median rank* computation.

Probe Intervals Algorithm 3 illustrates the details of the procedure **ProbeInterval**.

According to Equation 7 and 8, only the intervals *overlapping* at least one interval of an active object need to be probed. For example, in Figure 3(c), the intervals *overlapping* $I_{B,2}$ (the shaded area) will be probed since object B is active. To speed up the checking process, the intervals of active objects are merged into a set of *non-overlapping* intervals. Note that if interval I_1 *overlaps* interval I_2, the merged result is an interval I with $d^-(I) = min(d^-(I_1), d^-(I_2))$ and $d^+(I) = max(d^+(I_1), d^+(I_2))$. As shown at Line 9 in Algorithm 3, although some intervals are not probed, they remain for future computation because they contribute to the accumulative computation in Algorithm 2.

Algorithm 3: ***ProbeIntervals***($\mathcal{I}$) [***expected rank*** **version**]

```
   Input  : $\mathcal{I}$ : interval set
   Output : $\mathcal{I}'$ : new interval set
1  $\mathcal{I}' := \emptyset$;
2  $C$ := Merging intervals from all active objects ;
3  $\ell$ := furthest $d^+(I)$ for all $I \in C$ ;
4  for all $I \in \mathcal{I}$ do
5      if $I$ overlaps any $I_c \in C$ then
6          for each child interval $I'$ of $I$  * do
7              if $d^-(I') \leq \ell$ then
8                  $\mathcal{I}' := \mathcal{I}' \cup I'$
9      else
10         $\mathcal{I}' := \mathcal{I}' \cup I$ if $d^-(I') \leq \ell$ ;
11 return $\mathcal{I}'$;          * : if $e(I)$ is a data entry, $I' = I$
```

As to the computation of *median rank*, for active object U, only the intervals not *pruned* by Lemma 4 with $\phi = 0.5$ are merged into C. This implies the computation of *median rank* might be more IO efficient than that of *expected rank*, which is confirmed in our performance analysis and experiments.

The following example shows more details of interval probing.

Example 4: For intervals in Figure 3(c), 8 intervals ($I_{A,2}, I_{B,1}, I_{B,2}, I_{B,3}, I_{C,1}, I_{C,2}, I_{C,3}, I_{D,1}$) are probed for the computation of *expected rank*, while only 5 intervals ($I_{B,2}$, $I_{B,3}$, $I_{C,2}$, $I_{C,3}$, $I_{D,1}$) need to be probed to compute *median rank*.

By applying the sweepline paradigm, which is similar to Algorithm 2, merging process at Line 1 takes time $O(n_{\mathcal{I}} \log n_{\mathcal{I}})$, and Lines $3 - 9$ uses time $n_{\mathcal{I}'} + n_{\mathcal{I}} + n_p \times c_{io}$ where n_p is the number of intervals probed and c_{io} is the unit cost for IO operation.

C. Performance Analysis

We first analyze the time efficiency of our rank based KNN algorithm. The initial procedure takes a cost of at most $O(n \log n + n_{p_0} \times c_{io})$ as the heap structure is employed, where n is the number of objects and n_{p_0} is the number of IO operations invoked. Let h be the maximal height of *local* aR-trees, then there are at most h iterations in Algorithm 1.

For each round, as $n_{\mathcal{I}} \leq n \times m$, the cost is bounded by $O(n \times m \log(n \times m)) + n_{p_i} \times c_{io}$ based on the analysis of procedure **Sweepline** and **ProbeIntervals**, where n_{p_i} denotes the number of IO in i-th round. Consequently, the total time cost is $T = O(h \times n \times m \log(n \times m)) + \sum_{i=0}^{h} n_{p_i} \times c_{io}$.

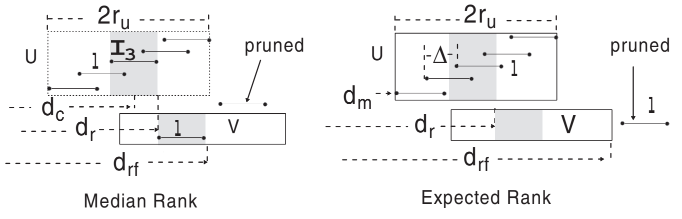


Fig. 4. IO cost Estimation

In the following part, we estimate the IO cost for the KNN query over 2D data under uniform assumption. For presentation simplicity, we only estimate the number of IO on leaf node level for all *local* aR-trees as it is the dominant cost for KNN query. Nevertheless, our analysis can be easily extended to internal node level or higher dimensions.

Theorem 3: Assume the domain sizes of all dimensions are between $[0, 1]$ and uncertain regions are circles with radius r_u. The centres of n objects and their instances follow the uniform distribution, while the instances of an object are organized by an aR-tree with fanout f. Then the expected number of leaf IO for all local aR-trees in Algorithm 1 is bounded by $\frac{nm\pi}{f} \times ((d_{rf} + \frac{\ell}{2})^2 - (d_{lf} - \frac{\ell}{2})^2)$, where $\ell = r_u\sqrt{2\pi f/m}$. Let $d_c = \sqrt{k/(\pi n)}$ and $d_m = d_c - r_u$. For *median rank*, we have $d_{lf} = max(\frac{\ell}{2}, d_c - \frac{3\times\ell}{2})$ and $d_{rf} = d_c + \frac{3\times\ell}{2}$. For *expected rank*, we have $d_{lf} = max(\frac{\ell}{2}, d_m - \ell)$ and $d_{rf} = d_m + 2r_u + \ell$.

Proof: According to Equation 6 and uniform assumption, for any instance u we can use $n\pi \times d(u)^2$ to estimate $r(u)$. In the rest of the proof, let U denotes the object whose centre is the k-th closest one to q and we assume $d^-(U) > 0$; Based on uniform assumption, we have $r = U_{rmax}$ where r is the upper bound used to *prune* objects for KNN query. As shown in Figure 4, for each object we assume c intervals $\{I\}$ with length ℓ are associated with its leaf nodes. They are uniformly distributed and sorted on $d^-(I)$. For *median rank* computation, without loss of generality, we assume c is an odd number. Then it is immediate that the interval ranked at the middle (e.g. interval I_3 in Figure 4) will determine the minimal/maximal *median rank* of the object. Let d_c denote the distance between the centre of U to q, then we have $d_c = \sqrt{k/(\pi n)}$ and $d_r = d_c + \frac{\ell}{2}$ where d_r is the distance such that $r = mr^+(U) = \pi d_r^2$. Let V be the **candidate** object with $mr^-(V) = r$ and $d_{rf} = d_c + \frac{3\times\ell}{2}$, with uniform assumption it is immediate that any interval I with $d^-(I) > d_{rf}$ will not be probed, thus invokes no leaf IO.

For *expected rank* computation, as shown in Figure 4 we have $r = er^+(U) = \alpha_1$ where $\alpha_1 = n\pi \times (\sum_{i=0}^{c-1}(d_m + \ell + i*\Delta)^2)/c$, $\Delta = \frac{2r_u - \ell}{c-1}$ and $d_m = d_c - r_u$ which is the minimal distance from U to q. Let V be the **candidate** object with $er^-(V) = er^+(U) = r$, since $er^-(V) = n\pi \times (\sum_{i=0}^{c-1}(d'_m + i*\Delta)^2)/c$ where d'_m is the minimal distance from V to q, we have $d'_m = d_m + \ell$. Consequently, any interval I with $d^-(I) > d_{rf}$, where $d_{rf} = d_m + \ell + 2\times r_u$, will not be probed. With similar rationale, we have $d_{lf} = d_c - \frac{3\times\ell}{2}$ and $d_{lf} = d_m - \ell$ for *median rank* and *expected rank* respectively. According to the uniform assumption, $\ell \leq r_u\sqrt{2\pi f/m}$ where f is the fanout of the *local* aR-tree. Based on the argument of [24], the total number of *local* aR-tree leaf node accessed is bounded by $\frac{nm\pi}{f} \times ((d_{rf} + \frac{\ell}{2})^2 - (d_{lf} - \frac{\ell}{2})^2)$. So the theorem holds. ■

As ℓ is smaller than r_u especially when the number of instances in an object is large, the computation of *median rank* has better chance to more IO efficient than that of *expected rank*. This is confirmed in our experiment.

IV. Randomized Algorithm

In this section, we present the randomized algorithm for rank based KNN query. The basic idea of the randomized algorithm is to sample the possible world such that the *expected rank* and *median rank* can be approximately computed in an efficient way. Suppose s possible worlds S_1, S_2, ..., S_s are sampled and each S_i has n sampled instances. As the sampling of possible worlds is independent of the KNN query and the processing might be time consuming for the MCMC [25] approach, we assume possible worlds are sampled in advance and instances of each S_i are organized by a R-tree, denoted by sR_i. In the following part, we present randomized algorithms for *expected rank* and *median rank* computation.

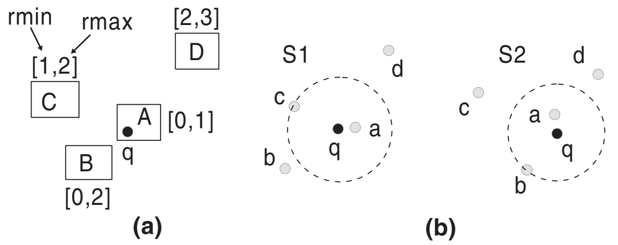


Fig. 5. A-KNN Example(k=2)

Algorithm 4: ***A-KNN***(gR, q, $s\mathcal{R}$, k)

```
Input  : $gR$ : the global $R$-tree, $q$ : query,$s\mathcal{R}$ : sample $R$-trees, $k$
Output : $\mathcal{O}$ : Approximate KNN of $q$
1  $\mathcal{O} = \emptyset$; $rank := 0$;
2  $\mathcal{C} :=$ Initialize($gR$, $q$, $k$);
3  Compute $U_{rmin}$ and $U_{rmax}$ for each $U \in \mathcal{C}$; Compute $l$ and $r$;
4  while $|\mathcal{O}| < k$ do
5      for each $sR \in s\mathcal{R}$ do
6          $U :=$ nextNN( $sR$ );
7          if $U \in \mathcal{C}$ and $U_{nv} \geq 0$ then
8              $U_{rmin} := U_{rmin} + U_{nv} \times (rank - U_{rmin})/s$;
9              $U_{rmax} := U_{rmax} - (U_{oldRmax} - rank)/s$;
10             $U_{nv} := U_{nv} - 1$;
11             Update $l$ and $r$;
12             for $V \in \mathcal{C}$ can be validated or pruned do
13                 $\mathcal{C} := \mathcal{C} - V$;
14                 $\mathcal{O} := \mathcal{O} \cup V$ if $V$ is validated;
15     $rank := rank + 1$;
16 return $\mathcal{O}$
```

A. Expected Rank Computation

For given sample set $\mathcal{S}$, we can use $\tilde{er}(U) = \sum_{i=1}^{s} r_i(U)/s$ to estimate the *expected rank* of an object U where $r_i(U)$ is the rank of U in sample S_i.

Example 5: As shown in Figure 5, there are two samples S_1 and S_2 for 4 objects. And we have $\tilde{er}(A) = 0$, $\tilde{er}(B) = 1.5$, $\tilde{er}(C) = 2$ and $\tilde{er}(D) = 2.5$.

Clearly, it will be much time and IO expensive to find the KNN by computing the estimated *expected rank* of each object and then ranking them. Algorithm 4 presents a CPU and IO efficient algorithm based on the incremental nearest neighbor technique [23]. Same as the exact algorithms, we first find the candidate objects $\mathcal{C}$ for the KNN query based on the *global* R-tree. Then the minimal and maximal *expected rank* of each object can be computed based on Sweepline algorithm proposed in Section III. As shown in Figure 5(a), U_{rmin} and U_{rmax} of each objects are first computed based on their uncertain regions. Clearly, for any object U, $U_{rmin} \leq r_i(U) \leq U_{rmax}$ when $1 \leq i \leq s$. In Line 6 and 14, we iteratively update U_{rmin} and U_{rmax} of objects by fetching the next nearest neighbor in all samples. Note that U_{nv} denotes the number of sample instances "unseen" so far from U, which is initially set to s. If U does not appear in S_i until current iteration, it implies $U_{rmin} \leq r_i(U) \leq U_{oldRmax}$ where the $U_{oldRmax}$ records the U_{rmax} value before the iteration process. Same as the exact algorithms, l and r values are dynamically maintained to *prune* or *validate* objects for the KNN query. As to the example in Figure 5(b), after the second iteration we have $\tilde{er}(A) = [0, 0]$, $\tilde{er}(B) = [1, 1.5]$, $\tilde{er}(C) = [1, 2]$ and $\tilde{er}(D) = [2, 3]$.

As discussed in Section III, the CPU cost at Line 2 and 3 is $O(n \times \log n)$. In our implementation, two priority queues are employed to dynamically maintain l and r with cost $O(\log k)$ per update. We use two AVL-trees to maintain U_{rmin} and U_{rmax} values of the candidate objects with time cost $O(\log n)$ per update. Then the amortized cost for *validation* and *pruning* at Line 12 is $O(1)$ in each iteration. The update CPU cost at Line 6 might be $O(\log n)$ in the worst case. Consequently, the time complexity of the A-KNN algorithm is $O(n \times \log n + s \times$

$n' \times \log n + n_1 \times c_{io})$ in the worst case where n' is the number of iterations at Line 4 and n_1 is the number of IO at Line 2 and 6. With similar rationale with exact algorithm, the expected number of leaf IO is bounded by $s \times n \times \pi(\sqrt{\frac{n'}{\pi n}} + \sqrt{\frac{f}{2n}})^2$ with uniform assumption where f is the fanout of the aR-tree. According to Lemma 6, $n' = n \times \pi \times (\sqrt{\frac{k}{\pi n}} + 4 \times r_u)^2$ in the worst case where r_u is the radius of the object.

Accuracy Evaluation For object U_j, let $u_{i,j}$ denote the sampled instance of U_j in S_i where $1 \leq i \leq s$ and $1 \leq j \leq n$. The event that whether $d(u_{i,l}) < d(u_{i,j})$ in S_i can be described by the following random variable

$$Z_{i,l} = \begin{cases} 1 & \text{if } j \neq l \text{ and } d(u_{i,l}) < d(u_{i,j}) \text{ in } S_i \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

Let random variable $X_j = \sum_{i=1}^{s} Y_i/s$ where $Y_i = \sum_{l=1}^{n} Z_{i,l}$, then $E(Y_i) = \sum_{l=1}^{n} Pr(d(U_l) < d(U_j))$ where $l \neq j$ since $Pr(Z_{i,l} = 1) = Pr(d(U_l) < d(U_j))$. According to the definition of $er(U_j)$, we have $E(Y_i) = er(U_j)$ and hence $E(X_j) = er(U_j)$. Clearly $\tilde{er}(U_j) = X_j$ since $\tilde{er}(U_j) = \sum_{i=1}^{s} r_i(U_j)/s = \sum_{i=1}^{s}(\sum_{l=1}^{n} Z_{i,l})/s$. Then the following theorem is immediate based on the Hoeffding bound [26].

Theorem 4: Given an ϵ $(0 < \epsilon < 1)$, a δ $(0 < \delta < 1)$, and n objects, if $s = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$, then

$$Pr(|X_j - er(U_j)| \leq \epsilon n) \geq 1 - \delta.$$

The theorem shows that the $\tilde{er}(U)$ is close to $er(U)$ with high probability if there are sufficient samples. Nevertheless, we are more interested in the *precision* and *recall* of our approximate answer. Since we only return k objects, the *precision* and *recall* are always the same. The following theorem indicates that when the sample size s is sufficiently large, the *precision* and *recall* can be guaranteed with probability at least $1 - \delta$.

To prove Theorem 5, we need the following Lemma.

Lemma 8: For two objects U_i and U_j, suppose that $er(U_i) < er(U_j)$. Then, $Pr(X_j \leq X_i) \leq exp(-s(er(U_j) - er(U_i))^2/(2 \times n^2))$.

Proof: Let $Z = X_i - X_j$. We have $Pr(X_i \geq X_j) = Pr(Z - E(Z) > er(U_j) - er(U_i))$. By Hoeffding inequality (Theorem 2 in [26]), the lemma is immediate. ■

Theorem 5: For given precision or recall α, suppose the objects are sorted on their *expected rank* with non-decreasing order as shown in Figure 6. Let λ equal to $er(U_{k+1}) - er(U_\beta)$ where $\beta = \lceil \alpha k \rceil$ and assume $\lambda \geq \epsilon n$. Then when the number of sample $s = O(\frac{2}{\epsilon^2} \log \frac{k \times n}{\delta})$, we have $Pr(k^* < \alpha \times k) \leq \delta$ where k^* is the number of *true* KNN objects returned by Algorithm 4.

Proof: As shown in Figure 6, if none of the objects in $U_k + 1, \ldots, U_n$ reverse order with any objects in $U_1, \ldots U_\beta$, we have $k^* > \alpha \times k$. Since the probability of $X_{k+1} < X_\beta$ is bounded by $exp(-s\lambda^2/n^2)$ based on Lemma 8 and for all $l < \beta$ and $j > k + 1$, $Pr(X_j < X_l) \leq Pr(X_{k+1} < X_\beta)$, we have $Pr(k^* < \alpha \times k) < k \times n \times exp(-s \times \lambda^2/(2n^2)) \leq \delta$. ■

B. Median Rank computation

According to the definition of *median rank*, if a sampled instance u appears more than once in all the samples, the rank of u is estimated by averaging the ranks of all its "copies"

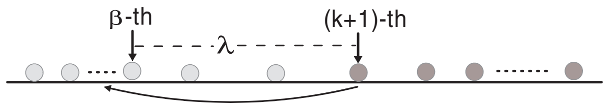


Fig. 6. Example of *reverse order*

throughout all the samples. Then the median of all the instance rank values is the estimation of $mr(U)$.

Although we can come up with similar accuracy guarantee as Theorem 4 to estimate $r(u)$ where u is a sampled instance, there is no theoretical accuracy guarantee for the estimation of $mr(U)$. Nevertheless, our experiments show the randomized algorithm for *median rank* estimation achieves similar approximate quality as that of *expected rank*. Algorithm 4 can be easily extended to tackle *median rank* computation according to its definition.

V. Extensions

A. Query Uncertainty

In some scenarios, the location of the query might be imprecise. So it is worthwhile investigating the rank based KNN query where the location of the query and objects are uncertain. Suppose the query q is an uncertain object $q = \{q_1, q_2, \ldots, q_m\}$, the definitions of *expected rank* and *median rank* are described as follows: $er_q(U) = \sum_{i=1}^{m} er_{q_i}(U) \times p_{q_i}$ and $mr_q(U) = \sum_{i=1}^{m} mr_{q_i}(U) \times p_{q_i}$.

Following the exact and randomized KNN algorithms in Section III and Section IV, the KNN algorithms for uncertain query are briefly presented below. Due to space limitation, we only introduce the *expected rank* based KNN query.

Exact Algorithm A straightforward extension is to apply Algorithm 1 on each instance of query q to compute the k' nearest neighbor where $k' > k$, then compute the KNN based on the outcome of Algorithm 1. It is not efficient as the entries of the objects might be loaded many times and the value of k' is not easy to set.

Alternatively, we can share the *ProbeIntervals* procedure during the computation: that is, for all query instances, we only use one set of intervals $\mathcal{I}$. At each iteration in Algorithm 1, the **Sweepline** procedure is called m times to compute $er^+_{q_i}(U)$ and $er^-_{q_i}(U)$ for each query instance q_i as $d^+(I)$ and $d^-(I)$ vary with different query instances. Then U_{rmin} and U_{rmax} of U can be computed. In the **ProbeIntervals** procedure, the intervals contributing to the future computation of any q_i will be probed.

Suppose the query q is organized by a *local* aR-tree, then we can further improve the CPU cost of above algorithm by synchronically traversing the tree with other local aR-trees of the objects in Algorithm 1. For the aR-tree of q, an entry can be regarded as a set of query instances whose locations are bounded by MBR of the entry. So we can compute U_{rmin} and U_{rmax} values for the objects at each level of q.

Randomized Algorithm Since the occurrence of query instances is independent of that of the object instances, we can sample the instances of q on the fly for given sample S_i. So in Algorithm 4, we only need to randomly select s instances q_1, q_2, ... , q_s from q and use q_i as the query point of the incremental nearest neighbor algorithm at S_i. With the same rationale in Section IV, Theorem 4 and 5 hold for uncertain query as well.

B. Continuous Distribution

In some applications, the uncertainty of the data might be described by continuous probabilistic functions (e.g. Gaussian). Our randomized algorithm can be directly applied as the samples of continuous distribution are same as those from discrete case. As to the exact KNN algorithm, we can discretize the distributions to an approximate level of granularity [11], [5]. For instance, various multidimensional histogram technique can be employed to partition uncertain regions. Although the numerical evaluation of the integral can not be totally avoided, same as other index techniques on uncertain data, our algorithm can significant reduce this cost.

VI. Performance Evaluation

We present results of a comprehensive performance study to evaluate the efficiency and scalability of proposed techniques in this paper. Following algorithms are evaluated.

NA the exact *expected rank* computation technique proposed in [11] and applied on the objects not *pruned* by Lemma 6 in Section III. As the modified technique for *median rank* computation has the same performance, we only evaluate *expected rank* based algorithm.

***ER* (*MR*)** the exact KNN Algorithm introduced in Section III for *expected (median) rank* computation. To efficiently process uncertain queries, we implement the techniques proposed in Section V, denoted by *UER* (*UMR*).

***R-ER* (*R-MR*)** the randomized KNN algorithm proposed in Section IV for approximated *expected (median) rank* computation. And *UR-ER* (*UR-MR*) represents the extension of *R-ER* (*R-MR*) to support uncertain queries.

All algorithms proposed in this paper are implemented in standard C++ with STL library support and compiled with GNU GCC. Experiments are run on a PC with Intel Xeon 2.40GHz dual CPU and 4G memory running Debian Linux. The disk page size is fixed to 2048 bytes.

In the experiments, the uncertain region of objects is a circle or sphere with radius r_u varying from 50 to 400 with default value 100. Suppose the PDF of an object is described by m instances which follow three popular distributions *Uniform*, *Normal* and *Zipf*. In the first two distributions, instances of an uncertain object follow *Uniform* or *Constrained Normal* distribution within the uncertain region. The standard deviation σ of *Constrained Normal* is set to $0.2 \times r_u$. For *Zipf* distribution, we first randomly choose an instance u inside the uncertain region, then the distances of other instances to u follow the *Zipf* distribution with $z = 0.5$. In this paper, *Zipf* serves as the default distribution and all instances of an object have the same appearance probability.

Two real spatial datasets, *LB* and *US*, are employed to represent the centres of uncertain regions. They contain $62K$ and $500K$ 2-dimensional points representing locations in Los Angeles and the United States respectively[3]. Synthetic datasets *2D* and *3D* are generated with size $100K$, in which the centres of uncertain regions are uniformly distributed. All dimensions are normalized to domain $[0, 10000]$ and *LB* with *Zipf* distribution is employed as the default dataset, denoted by LB_Z. Similarly, we have datasets LB_U, LB_N, US_Z, $3D_Z$, etc.

[3]Available at http://www.census.gov/geo/www/tiger/

In the experiments, a workload consists of $1,000$ query points following *Uniform* distribution in the domain. Extension for uncertain queries are also evaluated.

Table II below lists parameters which may potentially have an impact on our performance study. In the experiments, all parameters use default values unless otherwise specified.

TABLE II
System Parameters

Notation	Definition (Default Values)
r_u	the radius of uncertain object region(100)
n	number of objects ($63k$)
k	k value in KNN query (40)
m	number of instances of each object (2000)
s	number of samples (200)

A. Evaluating Efficiency

In this subsection, the average KNN query response time and the average number of pages accessed are recorded to measure the efficiency of the techniques against different settings.

We evaluate the performance of *NA*, *ER*, *MR*, *R-ER* and *R-MR* against datasets LB_U, LB_N, LB_Z, $3D_U$, $3D_N$ and $3D_Z$ in the first set of experiments. In Figure 7, empty bars on top of each algorithm represent the CPU cost, while the bar below them stand for the IO cost. Clearly, IO cost is the dominant cost for all algorithms. As expected, *NA* Algorithm is much slower than other techniques because of its large number of IO invoked. The randomized algorithms always have the best performance on various datasets and *R-MR* Algorithm slightly outperforms *R-ER* Algorithm. Although having similar CPU cost, *MR* Algorithm significantly outperforms *ER* Algorithm due to its IO efficiency as analyzed in Section III. As both *ER* and *MR* Algorithms are much faster than *NA* Algorithm and they provide exact KNN as well, we exclude *NA* in the following experiments.

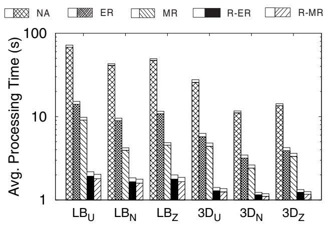


Fig. 7. Diff. datasets

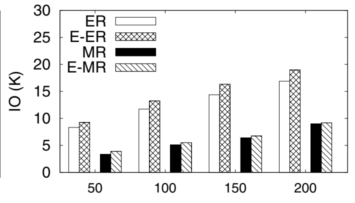
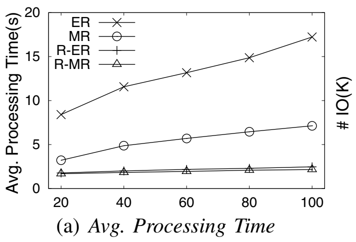


Fig. 8. IO Cost Estimation(diff. k)

We evaluate the effectiveness of the IO cost model proposed in Section III. The experiments are conducted on $2D_U$ dataset with $n = 100K$ and $m = 4000$, where k varies from 50 to 200. Figure 8 validates the fact that *MR* Algorithm is much more IO efficient than *ER* Algorithm. Moreover, the estimations are close to the real leaf IO cost, especially for *MR* Algorithm.

Figure 9 investigates the impact of k on the time and IO efficiency of the algorithms in the third set of experiments. It is shown that randomized algorithms are much less sensitive to the growth of k. Compared with *MR* Algorithm, the time and IO cost of *ER* grows faster but still scalable. As shown in Figure 9(b), when k grows from 20 to 100, the number of IO is only doubled.


(a) *Avg. Processing Time*

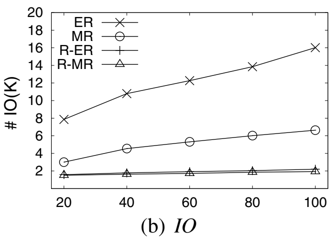

(b) *IO*

Fig. 9. Performance vs diff. k

The fourth set of experiments evaluates the time efficiency of the algorithms against the US_Z dataset where the number of objects varies from $100K$ to $500K$. As shown in Figure 10, the performance of all algorithms degrades slowly against the growth of n.

Figure 11 evaluates the impact of the number of instances m on the algorithms where m grows from 500 to $8,000$. Clearly, the performance of the randomized algorithms is not affected by m for given sample rate s, while the performance of exact algorithms drops with m. Specifically, the average processing time of *ER* (*MR*) Algorithm grows from $5.7(3.0)$ seconds to $30.8(9.5)$ seconds, when m increases from 500 to $8,000$.

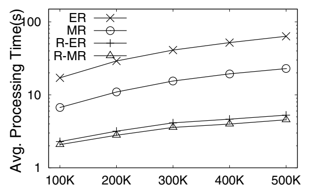


Fig. 10. Diff. n on US_Z

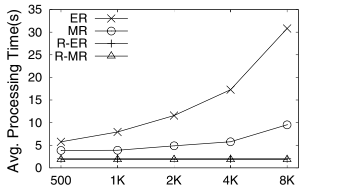


Fig. 11. Diff. m

Figure 12 investigates the impact of uncertain region size where r_u varying from 50 to 400. As expected, the larger the uncertain region size, the poorer the performance of the algorithms. Nevertheless, the randomized algorithms are less sensitive to the growth of r_u.

In the last set of experiments, we evaluate randomized algorithms against sampling rate s varying from 20 to 400. As shown in Figure 13, the time cost increases linearly with s, since the performance of the randomized algorithms is similar in each sample.

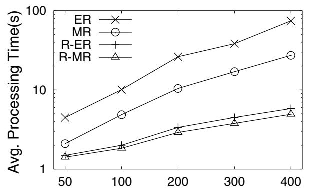


Fig. 12. Diff. r_u

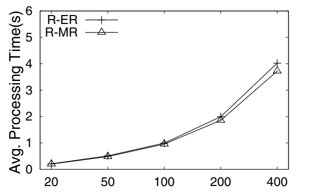


Fig. 13. Diff. s

B. Evaluating Accuracy

In this subsection, we evaluate the accuracy of randomized algorithms by *precision* and *recall* of the KNN results. As we output exactly k objects for KNN query, *precision* and *recall* of the results are always the same.

Figure 14 reports the impact of sampling rate s on randomized algorithms *R-ER* and *R-MR* on datasets LB_U and LB_Z. As expected, the approximation quality improves when s grows. With only 200 rounds of trials, the randomized algorithms can achieve a *precision* and *recall* value of at least 0.95 for both algorithms.

We also evaluate the impact of k on randomized algorithms with LB_U and LB_N datasets. Figure 15 shows the accuracy of the algorithms increases with k.

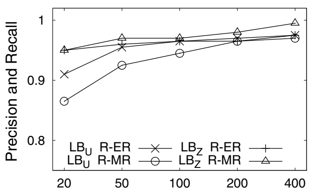


Fig. 14. Diff. s

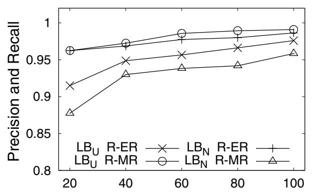


Fig. 15. Diff. k

C. Uncertain Query Evaluation

We evaluate the performance of exact and randomized algorithms against uncertain queries in this subsection. The workload consists of 200 uncertain queries which are randomly chosen from objects within LB_Z with $r_u = 50$ and $m = 300$.

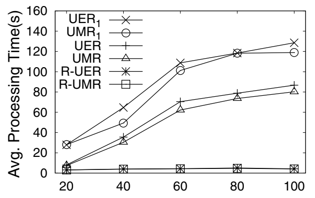


Fig. 16. Diff. k

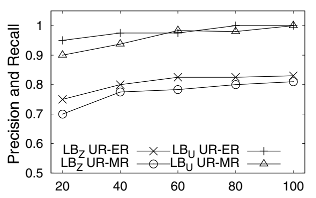


Fig. 17. Diff. k

Figure 16 reports the time efficiency of the algorithms when k varies from 20 to 100. It is not surprising that randomized algorithms on uncertain query perform much faster than exact algorithms since they have similar performance with randomized algorithms over query point according to the algorithm description in Section V-A. As to exact algorithms, *UER*$_1$ and *UER* represent the modified exact algorithms based on Algorithm 1 where *UER*$_1$ does not apply the synchronously traversing technique against q while *UER* is the full implementation of the techniques described in Section V-A. *UMR*$_1$ and *UMR* are defined in the same way. As shown in Figure 16, the synchronously traversing technique significantly improves the performance of exact algorithm. Nevertheless, the performance of all exact algorithms significantly degrades when k grows.

Figure 17 illustrates the accuracy of the randomized algorithms against datasets LB_U and LB_Z, where k varies from 20 to 100. With the same sampling rate $s = 200$, the approximation quality of the randomized algorithms over uncertain query is not as good as the one over query point. Nevertheless, we can still achieve a *precision* (*recall*) of 70% in the worst case of this experiment.

VII. Related Work

In the last two decades, the inherent uncertainty of data in many applications leads to the emergence of many uncertain database models ([2], [27], [28], [29], [30]) most of which are developed based on either tuple-level or attribute-level uncertainty. In this paper, we model the uncertain data on attribute-level.

On uncertain data, a large amount of work has been dedicated to top-k queries with different semantics such as U-top$k$[8], U-$k$ranks [8], PT-k [31], Global-top$k$ [17], *expected rank* top$k$ [11] and c-Typical-Top$k$ [32].

As a natural extension of top-k query, the problem of k nearest neighbor query over uncertain data has been studied in many recent works due to the natural data uncertainty in the applications such as location based services, traffic network analysis and sensor network. However, most of the current work [33], [34], [35], [5], [6] focus on the probabilistic NN query which is a special case of KNN query with $k = 1$. Particularly, in [33], Cheng *et al.* compute the qualification probabilities of objects satisfying probabilistic NN by first transforming the uncertainty of each object to PDF and CDF based on its distance to q. In [6], Kriegel *et al.* propose an approach to represent an object by multiple points sampled from its distance based PDF. The problem of efficient retrieval of data objects which have the minimum aggregate distance from multiple queries is addressed in [35]. Existential probability, which represents the probability of an object's existence in the database is used in [5] to derive lower and upper bounds for object pruning.

Although above works can output the top-k objects based on their probability of being the nearest neighbor of the query, the semantic is inherently different with the traditional KNN query. As to our best knowledge, the KNN query over uncertain data is only studied in [9], [10]. Ljosa *et al.* [9] propose an efficient index structure, called APLA-tree, to efficiently rank objects based on their expected distances(under L1-norm) to the query. Very recently, Cheng *et al.* [10] study the T-k-PNN query which finds multiple sets of k objects satisfying the query with probabilities higher than a given threshold T. An indexing approach called k-bound filtering is developed, while they also show that the computation of K-PNN can also be performed by using distance based PDFs and CDFs.

VIII. Conclusions

In this paper, rank based KNN query on uncertain data is studied where *expected (median) rank* satisfying important top-k properties are adopted as ranking criteria. Exact and randomized algorithms integrating efficient object pruning and IO accessing techniques are developed to process queries modeled by either query points or uncertain regions. Comprehensive experiments are conducted on both real and synthetic data to demonstrate the efficiency of our techniques. As a possible future work, we will investigate the incremental k nearest neighbor problem on spatial uncertain data.

Acknowledgement. The work was supported by ARC Grant (DP0987557, DP0881035 and DP0666428) and Google Research Award. The second and fourth authors' work was also partially supported by NICTA.

References

[1] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Querying the uncertain position of moving objects," in *Temporal Databases*, 1997.
[2] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004.
[3] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working models for uncertain data." in *ICDE*, 2005.
[4] J. Pei, M. Hua, Y. Tao, and X. Lin, "Mining uncertain and probabilistic data: problems, challenges, methods, and applications," in *KDD*, 2008.
[5] G. Beskales, M. A. Soliman, and I. F. Ilyas, "Efficient search for the top-k probable nearest neighbors in uncertain databases," *PVLDB*, vol. 1, no. 1, 2008.
[6] H.-P. Kriegel, P. Kunath, and M. Renz, "Probabilistic nearest-neighbor query on uncertain objects," in *DASFAA*, 2007, pp. 337–348.
[7] R. Cheng, J. Chen, M. F. Mokbel, and C.-Y. Chow, "Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data," in *ICDE*, 2008.
[8] M. A. Soliman, I. F. Ilyas, and K. C. Chang., "Top-k query processing in uncertain databases." in *ICDE 2007.*
[9] V. Ljosa and A. K. Singh, "Apla: Indexing arbitrary probability distributions," in *ICDE*, 2007.
[10] R. Cheng, L. Chen, J. Chen, and X. Xie, "Evaluating probability threshold k-nearest-neighbor queries over uncertain data," in *EDBT*, 2009.
[11] G. Cormode, F. Li, and K. Yi, "Semantics of ranking queries for probabilistic data and expected ranks," in *ICDE*, 2009.
[12] C. Re, N. N. Dalvi, and D. Suciu, "Efficient top-k query evaluation on probabilistic data," in *ICDE*, 2007.
[13] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas, "Mcdb: a monte carlo approach to managing uncertain data," in *SIGMOD*, 2008.
[14] M. A. Soliman and I. F. Ilyas, "Ranking with uncertain scores," in *ICDE*, 2009.
[15] S. Khanna and W. C. Tan, "On computing functions with uncertainty," in *PODS*, 2001.
[16] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom, "Computing the median with uncertainty," in *STOC*, 2000.
[17] X. Zhang and J. Chomicki, "On the semantics and evaluation of top-k queries in probabilistic databases," in *DBRank*, 2008.
[18] J. G. Shanthikumar and M. Shaked, *Stochastic Orders and Their Applications*. Academic Press, 1997.
[19] A. M. Gibbons, *Algorithmic Graph Theory*, 1980.
[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (second edition)*, 2001.
[21] M. L. Yiu, N. Mamoulis, and Y. Tao, "Efficient quantile retrieval on multi-dimensional data," in *EDBT*, 2006.
[22] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate medians and other quantiles in one pass and with limited memory," in *SIGMOD*, 1998.
[23] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases." *TODS*, vol. 24, no. 2, pp. 265–318, 1999.
[24] Y. Theodoridis and T. K. Sellis, "A model for the prediction of r-tree performance," in *PODS*, 1996.
[25] W. Gilks, S. Richardson, and D. Spiegelhalter, *Markov chain Monte Carlo in practice*. Chapman & Hall, 1996.
[26] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *JASA 1963*.
[27] N. N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," in *VLDB*, 2004, pp. 864–875.
[28] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom, "Uldbs: Databases with uncertainty and lineage," in *VLDB*, 2006.
[29] L. Antova, C. Koch, and D. Olteanu, "Query language support for incomplete information in the maybms system," in *VLDB*, 2007.
[30] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah, "Orion 2.0: native support for uncertain data," in *SIGMOD Conference*, 2008.
[31] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: A probabilistic threshold approach." in *SIGMOD*, 2008.
[32] T. Ge, S. Zdonik, and S. Madden, "Top-k queries on uncertain data: On score distribution and typical answers," in *SIGMOD*, 2009.
[33] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *SIGMOD Conference*, 2003.
[34] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Querying imprecise data in moving object environments," *TKDE*, vol. 16, no. 9, 2004.
[35] X. Lian and L. Chen, "Probabilistic group nearest neighbor queries in uncertain databases," *TKDE*, vol. 20, no. 6, 2008.