# Quantile-Based KNN Over Multi-Valued Objects

Wenjie Zhang, Xuemin Lin, Muhammad Aamir Cheema, Ying Zhang, Wei Wang

*The University of New South Wales, Australia*

{zhangw, lxue, macheema, yingz, weiw}@cse.unsw.edu.au

***Abstract*— $K$ Nearest Neighbor search has many applications including data mining, multi-media, image processing, and monitoring moving objects. In this paper, we study the problem of $K$NN over multi-valued objects. We aim to provide effective and efficient techniques to identify KNN sensitive to relative distributions of objects. We propose to use quantiles to summarize relative-distribution-sensitive $K$ nearest neighbors. Given a query $Q$ and a quantile $\phi \in (0, 1]$, we firstly study the problem of efficiently computing $K$ nearest objects based on a $\phi$-quantile distance (e.g. median distance) from each object to $Q$. The second problem is to retrieve the $K$ nearest objects to $Q$ based on overall distances in the "best population" (with a given size specified by $\phi$-quantile) for each object. While the first problem can be solved in polynomial time, we show that the 2nd problem is NP-hard. A set of efficient, novel algorithms have been proposed to give an exact solution for the first problem and an approximate solution for the second problem with the approximation ratio 2. Extensive experiment demonstrates that our techniques are very efficient and effective.**

## I. Introduction

Given a set $\mathcal{D}$ of objects (points) in a $d$-dimensional metric space and a $d$-dimensional query object (point) $q$, the $K$ nearest neighbor search retrieves the $K$ closest objects to $q$ from $\mathcal{D}$. The conventional KNN search has been extensively studied [15], [23] with a wide spectrum of applications including data mining, contents-based image retrieval, and location based services. In this paper, we study the problem of $K$ nearest neighbor search over objects each of which has a collection of values (instances) without temporal constraints specified; that is, we do not deal with sequence databases [1], [18].

The existing model, probabilistic KNN, is to apply the uncertain semantics to each object by treating the collection of instances of each object mutually exclusive. It aims to catch relative distributions among objects with multi-instances. The two semantics of ranking top-$k$ uncertain tuples are employed in a probabilistic KNN model: 1) retrieving $k$ tuples that can co-exist in a possible world (e.g. U-top$k$) [24], and 2) retrieving tuples according to the probability that a tuple is top-$k$ or at a specific rank in all possible worlds (e.g. U-$k$Ranks and PT-$k$) [24], [17] [1]. While various probabilistic NN models are proposed in [3], [5], [20], a probabilistic KNN model over uncertain data has been proposed following U-top$k$ ranking semantics [6]. In these probabilistic KNN models, the probability for an object to be KNN to a query object is calculated to define the result of a KNN. Nevertheless, below we show that the probabilistic KNN models may provide results *insensitive* to relative distributions of instances of objects.

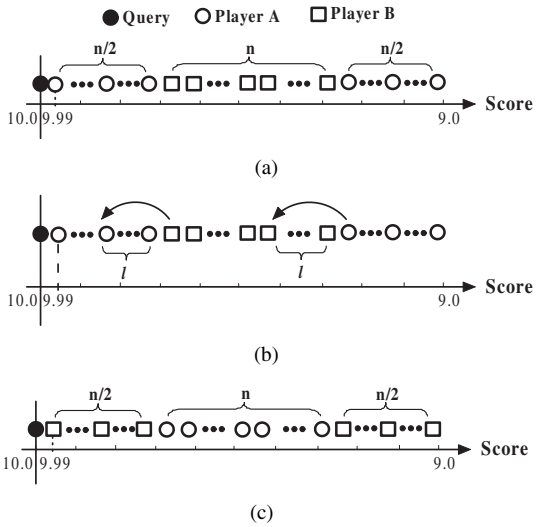



Fig. 1. Motivating Example

**Motivating Example.** Let $k = 1$. In gymnastics, suppose that we want to select the "best" balance-beam player among all candidates to participate a world championship. The scores of two players A and B, based on the most recent $n$ games/attempts, are depicted in Figure 1(a), respectively. Assume that the $2n$ scores of A and B ($n$ for A and $n$ for B) are distributed from 9.99 to 9.0 as depicted in Figure 1(a).

Assume that we approximately treat each player as an uncertain object and the score of an attempt as an instance with the equal occurrence probability. It can be immediately verified that based on the existing probabilistic NN models, player A and player B have the same probability, $\frac{1}{2}$, to be the nearest neighbor of the query point $q$ (i.e. the score 10) if $|10 - score|$ is used as the distance metric. We permutate the distribution in Figure 1(a) by swapping the two pairs of instances of A and B as depicted in Figure 1(b). It is immediate that A and B still have the same probability, $\frac{1}{2}$, to be the nearest neighbor regarding the score distribution after these two permutations. Choosing $l = \frac{n}{2}$, the score distribution in Figure 1(a) is eventually modified to the score distribution in 1(c) after $\frac{n}{2}$ such pairs of permutations; consequently, the nearest neighbor probabilities of A and B, respectively, remain unchanged, $\frac{1}{2}$, regarding the distribution in Figure 1(c).

**Quantile-Based KNN.** The examples in Figures 1 (a)-(c) demonstrate that the existing probabilistic KNN models may

[1]When $k = 1$, these two models are the same.

be insensitive to relative distributions of object instances. Very recently, in [10] a novel model based on the *expected* rank for ranking top-$k$ uncertain objects has been proposed. Regarding the distributions and the permuted intermediate distributions as depicted above in Figures 1 (a)-(c), player A and B always have the same expected rank. Moreover, in the above application we do not need to enforce the uncertain semantics among multi-instances of each player by treating them mutually exclusive. Motivated by these, we treat each player as a multi-valued object.

Quantiles [26] may provide a succinct summary of data distributions. In this paper, we investigate the KNN problem over multi-valued objects based on a $\phi$-quantile distance ($\phi \in (0, 1]$) from a multi-valued object to a query $Q$; for example, the median is the $0.5$-quantile. We extend our investigation to the KNN problem over multi-valued objects based on overall distances in the "best population" (with a given size specified by $\phi$-quantile) regarding each object; such overall distances are called a $\phi$-quantile group-base distance.

Regarding the above example, our KNN problem based on $0.5$-quantile distances is to rank players based on their median performances, respectively. The KNN problem based on a $0.5$-quantile group-base distance is to rank players based on their overall performances of the top-$50\%$ of scores, respectively.

The above example contains multi-valued objects in a 1-dimensional space and the query is a single-valued point. Nevertheless, our investigation covers the applications where data objects consist of multiple instances in a $d$-dimensional space and a query object may also consist of multiple instances in a $d$-dimensional space. For instance, in NBA the performance of a player per game may be measured by his statistics (scores, assists, rebounds, steals, blocks) and may be treated as an instance of the player; consequently, each player has a set of instances. Suppose that a team wants to sign a contract with player A and wants to find his market value. The team may want to find out the top-$k$ "similar" NBA players, with existing contracts, to A against their recent game statistics. Then, the team can use the salaries information of these $k$-players to project the salary level of A.

**Contributions.** To the best of our knowledge, this is the first paper to study KNN problems regarding quantiles over multi-valued objects. Yiu *et al* [26] develop efficient techniques to compute quantile-distances among data points; nevertheless the techniques are not applicable to our problem due to the following reasons. Firstly, the query object in our problem setting may have multiple instances and we count all pair combinations between an object and a given query object, while the computation of multi-source-based quantile-distances in [26] is to compute the distance of an instance to its nearest given source. Secondly, the quantile group-base distance problem studied in this paper is NP-hard. Our contribution may be summarized as follows.

- We make the first attempt to identify KNN sensitive to the relative distributions among multi-valued objects.
- Efficient, novel techniques are proposed for computing quantile distance based KNN against a set of multi-valued objects and a given query object that is also multi-valued.
- We show that the problem of KNN against the quantile group-base distance is NP-hard. Novel and efficient algorithms are proposed with the approximation ratio 2.

As a byproduct, our techniques to compute a $\phi$-quantile distance is $O(n)$ if single-valued object is involved while the technique in [26] is $O(n \log n)$ where $n$ is the number of instances. Besides the theoretical analysis, an extensive performance evaluation demonstrates that the proposed techniques are both efficient and effective.

The rest of the paper is organized as follows. In Section II, we formally define the problems and provide some necessary background information. In Section III, we present the framework of our algorithms to conduct KNN against these 2 quantile-based KNN problems. Section IV and Section V present query processing techniques for these two KNN problems, respectively. In Section VI, we report our experiment results. Related work is summarized in Section VII. This is followed by conclusions.

## II. Background Information

We present problem definition and necessary preliminaries. For reference, notations frequently used in the paper are summarized in Table I.

| Notation | Definition |
|---|---|
| $\mathcal{U}$ | set of of objects |
| $U$ ($Q$) | multi-valued (query) object |
| $E$ | entry of R-tree |
| $u$ ($q$) | instance of $U$ ($Q$) - a point in $d$-dimensional space |
| $w(u)$ ($w(S)$) | (total) weight of $u$ (the set $S$) |
| $d(q, u)$ | Euclidean distance between $q$ and $u$ |
| $d^{lo}(E, E')$ | distance lower-bound between $E$ and $E'$ |
| $d^{up}(E, E')$ | distance upper-bound between $E$ and $E'$ |
| $d_\phi(Q, U)$ | $\phi$-quantile distance of $Q$ and $U$ |
| $gbd_\phi(Q, U)$ | $\phi$-quantile group-base distance of $Q$ and $U$ |
| $Q \times U$ | Cartesian product of instances from $Q$ to $U$ |

TABLE I
THE SUMMARY OF NOTATIONS.

### A. Problem Definition

Given a collection $S$ of $m$ elements, each element $s_i$ has a weight $w(s_i)$ where $0 < w(s_i) \leq 1$ and $\sum_{i=1}^{m} w(s_i) = 1$. Let $S$ be sorted increasingly on a search key $f$ - a function; that is, $f(s_i) \leq f(s_j)$ if $i < j$. Without loss of generality, in this paper a *sorted* collection $S$ of data elements, thereafter, always means sorting $S$ *increasingly* on a given search key unless otherwise specified.

*Definition 1 ($\phi$-quantile of $S$):* Given a $\phi$ ($0 < \phi \leq 1$), the $\phi$-quantile $S_\phi$ of $S$ is the **first** element $s_i$ in the **sorted** $S$ on the search key such that $\sum_{j=1}^{i} w(s_j) \geq \phi$.

In our problem definition, an instance of an object $U$ (or $Q$) is weighted - weight gives the *representativeness* of an instance in $U$. For instance, in the examples in Section I a game statistic may appear multiple times; consequently a normalized weight

(the occurrence of an instance over the total occurrences of all instances) may be used to indicate the representativeness of an instance. Note that the total of such weights in $U$ (or $Q$) is 1.

A multi-valued object $U$ is represented as $\{(u_i, w(u_i))|1 \leq i \leq m\}$ where $u_i$ is a point in a $d$-dimensional space, $0 < w(u_i) \leq 1$ $(1 \leq i \leq m)$, and $\sum_{i=1}^{m} w(u_i) = 1$. A query object $Q$ is also a multi-valued object. We use $\mathcal{U}$ to denote a set of multi-valued objects.

For a given $Q$ and each $U \in \mathcal{U}$, there are totally $(|Q| \times |U|)$ pairs of instances in $Q \times U$ where each pair $(q_i, u_j)$ ($q_i \in Q$ and $u_j \in U$) has the weight $w(q_i) \times w(u_j)$, namely $w(q_i, u_j)$. Clearly, $\sum_{q_i \in Q, u_j \in U} w(q_i) \times w(u_j) = 1$. The *Euclidean distance* $d(q_i, u_j)$[2] between $q_i$ and $u_j$ is called the distance of $(q_i, u_j)$. Let $Q \times U = \{((q_i, u_j), w(q_i, u_j)) \mid q_i \in Q \ \& \ u_j \in U\}$.

*Definition 2 ($\phi$-quantile distance of $Q$ and $U$):* Given a $\phi \in (0, 1]$, let $Q \times U$ be sorted increasingly on the search key - the distance $d(q_i, u_j)$ of each element $(q_i, u_j)$. Then, the distance of the $\phi$-quantile of $Q \times U$ is called the *$\phi$-quantile distance* of $Q \times U$, denoted by $d_\phi(Q, U)$.

Definition 2 states that if $(q, u)$ is the $\phi$-quantile of $Q \times U$ (i.e., $(Q \times U)_\phi = (q, u)$) then $d(q, u)$ is $d_\phi(Q, U)$.

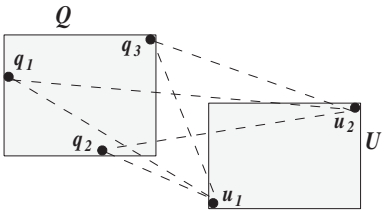



Fig. 2. Distances between 2 Multi-Valued Objects

*Example 1:* Regarding the example in Figure 2, $|Q|$ = 3 and $|U|$ = 2. Assume that $w(q_1) = \frac{1}{2}$, $w(q_2) = w(q_3) = \frac{1}{4}$; $w(u_1) = w(u_2) = \frac{1}{2}$. Consequently, $Q \times U$ consists of the following six pairs sorted on their distances increasingly: $Q \times U = \{((q_2, u_1), \frac{1}{8}), ((q_3, u_1), \frac{1}{8}), ((q_3, u_2), \frac{1}{8}), ((q_1, u_1), \frac{1}{4}), ((q_2, u_2), \frac{1}{8}), ((q_1, u_2), \frac{1}{4})\}$.

Note that the $\frac{2.5}{8}$-quantile and the $\frac{3}{8}$-quantile of $Q \times U$ are the same $(q_3, u_2)$. The 0.2-quantile distance $d_{0.2}(Q, U)$ of $Q$ and $U$ is $d(q_3, u_1)$, $d_{0.5}(Q, U)$ is $d(q_1, u_1)$, $d_{0.6}(Q, U)$ is also $d(q_1, u_1)$. □

Below, we specify a measure based on aggregates to define the *top/best* quantile-population of $S$.

*Definition 3 ($\phi$-quantile population of $S$):* Given a $S$ and a $\phi \in (0, 1]$, a $\phi$-quantile population $S_{\phi,P}$ of $S$ is a sub-collection $S'$ of $S$ such that the total weights of the elements in $S'$ is not smaller than $\phi$ and removing any element from $S'$ makes the total weights in the remaining sub-collection smaller than $\phi$.

*Definition 4 ($\phi$-quantile group-base distance):* Given a $\phi \in (0, 1]$, the $\phi$-quantile *group-base* distance of $Q$ and $U$ is the minimum total weighted distance among $\phi$-quantile populations of $Q \times U$; that is, the minimum value of $\sum_{(q,u) \in S'} w(q)w(u)d(q, u)$ with the constraint that $S'$ is a $\phi$-quantile population of $Q \times U$.

The $\phi$-quantile *group-base* distance between $Q$ and $U$ is denoted by $gbd_\phi(Q, U)$. Note that for a $\phi \in (0, 1]$, the example below shows that $gbd_\phi(Q, U)$ is not always defined on the set of the "consecutive" smallest distances. In fact, we will show in Section 5 that the computation of $gbd_\phi(Q, U)$ is NP-hard.

*Example 2:* Regarding Example 1, let $\phi = 0.5$. $gbd_{0.5}(Q, U) = \frac{1}{8}d(q_2, u_1) + \frac{1}{8}d(q_3, u_1) + \frac{1}{8}d(q_3, u_2) + \frac{1}{8}d(q_2, u_2)$ instead of $\frac{1}{8}d(q_2, u_1) + \frac{1}{8}d(q_3, u_1) + \frac{1}{8}d(q_3, u_2) + \frac{1}{4}d(q_1, u_1)$. Here, $\{((q_2, u_1), \frac{1}{8}), ((q_3, u_1), \frac{1}{8}), ((q_3, u_2), \frac{1}{8}), ((q_1, u_1), \frac{1}{4})\}$ is not even a 0.5-quantile population of $Q \times U$.

In fact, there are several 0.5-quantile populations of $Q \times U$, including $\{((q_3, u_1), \frac{1}{8}), ((q_2, u_2), \frac{1}{8}), ((q_1, u_1), \frac{1}{4})\}$, $\{((q_2, u_1), \frac{1}{8}), ((q_2, u_2), \frac{1}{8}), ((q_1, u_1), \frac{1}{4})\}$, etc. □

*Definition 5 ($\phi$-Quantile KNN):* Given a $\phi \in (0, 1]$, a set $\mathcal{U}$ of multi-valued objects in a $d$-dimensional space, and a multi-valued query object $Q$, the $\phi$-quantile KNN problem is to retrieve the set $\Phi_K$ of $K$ objects from $\mathcal{U}$ such that for each $U \in \Phi_K$ and each $U' \in \mathcal{U} - \Phi_K$, $d_\phi(Q, U) \leq d_\phi(Q, U')$.

*Definition 6 ($\phi$-Quantile Group-base KNN):* Given a $\phi \in (0, 1]$, a set $\mathcal{U}$ of multi-valued objects in a $d$-dimensional space, and a multi-valued query object $Q$, the $\phi$-quantile group-base KNN problem is to retrieve the set $\Phi_K$ of $K$ objects from $\mathcal{U}$ such that for each $U \in \Phi_K$ and each $U' \in \mathcal{U} - \Phi_K$, $gbd_\phi(Q, U) \leq gbd_\phi(Q, U')$.

**Problem Statement.** In this paper, for a given $\phi \in (0, 1]$, we study the problems of efficiently computing the $\phi$-Quantile KNN and the $\phi$-Quantile Group-base KNN.

### B. Preliminaries

Given a collection $S$ of $m$ elements, each element $s_i$ has a weight $w(s_i)$ where $0 < w(s_i) \leq 1$ and $\sum_{i=1}^{m} w(s_i) \leq 1$. A naive way to compute the $\phi$-quantile is to firstly sort $S$ regarding a given search key $f$, and then scan the sorted list to obtain the $\phi$-quantile of $S$. Clearly, the naive algorithm runs in $O(m \log m)$.

In [8], an efficient and effective partitioning technique PARTITIONING ($S$) is proposed to find an element $s \in S$ to divide $S$ into two sub-collections $S_1$ and $S_2$ with the following properties:

1) for each $s' \in S_1$, $f(s') \leq f(s)$; and for each $s' \in S_2$, $f(s') \geq f(s)$.
2) $|S_1| \geq \frac{3}{10}m - 6$ and $|S_2| \geq \frac{3}{10}m - 6$.

Using the partitioning technique, in Algorithm 1 we present an iteration-based algorithm to compute a $\phi$-quantile when $S$ is not sorted.

In Algorithm 1, $w(S_1)$ denotes the total weights of the elements in $S_1$. When $S$ has only one element, $S_1 = S_2 = \emptyset$.

It is shown in [8] that the time complexity of PARTITIONING ($S$) is linear - $O(|S|)$. Consequently, each iteration runs in linear time regarding the current sub-collection size. Recall the property 2 above in PARTITIONING ($S$). It is immediate that the sizes of sub-collections involved in the

[2]Note that our techniques developed in this paper is based on Euclidean distance; nevertheless they can be immediately extended to cover other distance metrics.

**Algorithm 1**: QUANTILE ($S$, $\phi$)

```
Input  : S: a collection of m elements; φ: 0 < φ ≤ Σ_{i=1}^{m} w(s_i);
         f: specify a search key;
Output : φ-quantile of S
(s, S1, S2) ⟵ PARTITIONING (S);
if φ ≤ w(S1) then
    call QUANTILE (S1, φ);
else
    if φ > w(S1) + w(s) then
        call QUANTILE (S2, φ − w(S1) − w(s));
    else
        return s;
```

iterations in Algorithm 1 are exponentially reduced - at the $i$th iteration bounded by $((\frac{7}{10})^{i-1}m + c)$ where $c$ is a constant; consequently, the time complexity of Algorithm 1 is **linear** - $O(m)$. The correctness of Algorithm 1 immediately follows from the property 1 of PARTITIONING ($S$).

## III. Framework Overview

Our techniques for solving the $\phi$-quantile KNN and and the $\phi$-quantile group-base KNN for a given $\phi \in (0, 1]$ follow a standard seeding-refinement paradigm outlined in Algorithm 2.

**Algorithm 2**: Framework

- **Phase 1 - Seeding:** Compute the $\phi$-quantile (or $\phi$-quantile group-base) distance from each of the $K$ chosen objects to $Q$.
- **Phase 2 - Refinement:** Determine the final solution for $\phi$-quantile KNN (or $\phi$-quantile group-base KNN).

In the seeding phase, we choose $K$ objects and compute their $\phi$-quantile distances (or $\phi$-quantile group-base distances); assume that $\gamma_k$ ($\lambda_K$) is the maximal of these $K$ $\phi$-quantile distances (or the $\phi$-quantile group-base distances). In the refinement phase, we use $\gamma_K$ ($\lambda_K$) and $\phi$ to effectively prune objects and iteratively update $\gamma_K$ ($\lambda_K$) (if necessary).

**Selecting $K$ Objects in the Seeding Phase.** Any $K$ multi-valued objects from $\mathcal{U}$ could be used for the seeding phase. Clearly, the smaller $\gamma_K$ is, the more powerful $\gamma_K$ may be used in the refinement for pruning. In our algorithm, we use the mean $a_U$ of the multiple instances for each multi-valued object $U$, and we use the mean $a_Q$ of the multiple instances of $Q$. Then we apply the KNN algorithm in [16] to obtain the $K$ nearest neighbors to $a_Q$ from $\{a_U \mid U \in \mathcal{U}\}$. Subsequently, we use the $K$ objects corresponding to these $K$ nearest means to $a_Q$ as the $K$ objects in the seeding phase for both $\phi$-quantile KNN and $\phi$-quantile group-base KNN, respectively.

**Data Structures.** Below are the data structures used in the seeding and refinement phases in our techniques. For each multi-valued object $U \in \mathcal{U}$, a *local* aR-tree [21] is built to organize its multiple instances. The aggregate information kept on each intermediate entry is the sum of weights of instances

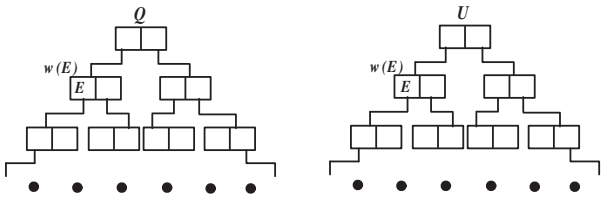



Fig. 3. Local aR-trees for Multi-Valued Objects

indexed by the entry. Namely, for every intermediate entry $E$ in the local aR-tree, we record the weight of $E$ as the sum of weights (total weights) of instances having $E$ as an ancestor. Local aR-trees will facilitate the efficient computation of $\phi$-quantile (or $\phi$-quantile group-base) KNN and support effective pruning techniques. Figure 3 shows the local aR-trees for $Q$ and $U$ where each intermediate entry records $w(E)$.

Besides, we maintain an R-tree on the MBBs of multiple instances of objects. That is, for each object we first obtain the MBB of its multiple instances. Then we build an R-tree on these MBBs. This R-tree is called the *global* R-tree. Note in the global R-tree, each leaf is an MBB of an object.

**Distances between MBBs.** Given two MBBs $E$ and $E'$, we compute the minimal and maximal distances $d^{lo}(E, E')$ and $d^{up}(E, E')$ between them as follows. For each dimension $i$ ($1 \leq i \leq d$), let $I_{E,i}$ and $I_{E',i}$ denote the intervals on which $E$ and $E'$ are projected, respectively. The minimum distance $mindist_i(E, E')$ between $I_{E,i}$ and $I_{E',i}$ is defined as follows. If $I_{E,i}$ overlaps with $I_{E',i}$ then $mindist_i(E, E') = 0$ otherwise $mindist_i(E, E')$ is the minimal value among the distances of 4 pairs of the ends of $I_{E,i}$ and $I_{E',i}$. $maxdist_i(E, E')$ is the the maximal value among the distances of 4 pairs of the ends of $I_{E,i}$ and $I_{E',i}$.

$$d^{lo}(E, E') = \sqrt{\sum_{i=1}^{d} (mindist_i(E, E'))^2}$$

$$d^{up}(E, E') = \sqrt{\sum_{i=1}^{d} (maxdist_i(E, E'))^2}$$

Figure 4 below shows representative examples. Note that we can immediately verify that for each pair of instances $(u, u')$ where $u$ is contained by $E$ and $u'$ is contained by $E'$, $d^{lo}(E, E') \leq d(u, u') \leq d^{up}(E, E')$. Thus, $d^{lo}(E, E')$ and $d^{up}(E, E')$ can be used as a lower- and upper- bound of the distance of any pair in $E \times E'$. Immediately, computing minimal/maximal distance between the two $d$-dimensional MBBs requires $O(d)$ time.

## IV. $\phi$-Quantile KNN

We present our techniques for conducting $\phi$-quantile KNN for a given $\phi \in (0, 1]$. We first present an efficient algorithm to compute a $\phi$-quantile distance between $Q$ and $U$ instead of a brute-force computation; this will be used in the two phases. Then, we present a set of novel pruning techniques in the refinement phase, as well as the refinement algorithm.

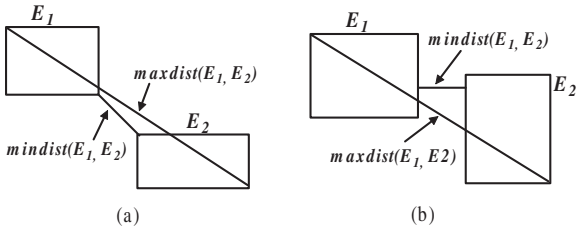



Fig. 4. Minimal/Maximal Distance between 2 MBBs

### A. Efficiently Computing $\phi$-Quantile Distances.

Given $Q$, $U$, and a $\phi \in (0, 1]$, we present an efficient algorithm to compute $d_\phi(Q, U)$ in this section.

**Naive Linear Algorithm.** Firstly, for each $(q_i, u_j)$ in $Q \times U$, we calculate its distance $d(q_i, u_j)$ and its weight $w(q_i, u_j)$ ($= w(q_i)w(u_j)$). Then call Algorithm 1 to produce the $\phi$-quantile $(q, u)$ of $Q \times U$ regarding the search key value (distance) of each $(q_i, u_j)$ and the weight $w(q_i, u_j)$ of each $(q_i, u_j)$. Clearly, $d_\phi(Q, U) = d(q, u)$ and the naive algorithm runs in linear time regarding $|Q \times U|$; that is, $O(|Q \times U|)$.

**Pruning-based Linear Algorithm.** While it is costly to enumerate all pairs of instances in $Q \times U$, intuitively most pairs in $Q \times U$ are possible to be removed without enumerating them. This may be done by using the local aR-trees of $Q$ and $U$, respectively. Our algorithm is based on level-by-level synchronous traversal on the local aR-trres of $Q$ and $U$. The example below gives the basic idea of the algorithm.

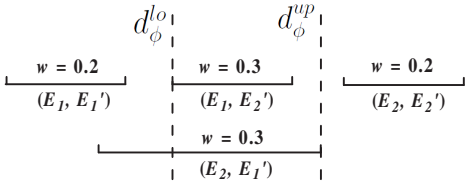



Fig. 5. Prune Entries at the Current Level

*Basic Idea.* Suppose that the root $R_Q$ of local aR-tree of $Q$ has 2 entries $(E_1, E_2)$, and the root $R_U$ of loca aR-tree of $U$ has 2 entries $(E_1', E_2')$. Totally, the 4 pairs of entries are depicted in Figure 5 where the two ends of each interval, corresponding to $(E_i, E_j')$, are $d^{lo}(E_i, E_j')$ and $d^{up}(E_i, E_j')$, respectively, and $w(E_i, E_j')$ is also shown as $w$. Assume that a *lower-bound* $d_\phi^{lo}$ and an *upper-bound* $d_\phi^{up}$ of $d_\phi(Q, U)$ are as what are depicted in Figure 5, respectively.

Since $d^{up}(E_1, E_1')$ is smaller than $d_\phi^{lo}$, $(E_1, E_1')$ can be removed. The pair of $(E_2, E_2')$ can also be removed because $d^{lo}(E_2, E_2')$ is larger than $d_\phi^{up}$. Consequently, we only focus on 2 pairs of entries, $(E_1, E_2')$ and $(E2, E_1')$, in the next level iteration. As the distance of any pair of instances in $(E_1, E_1')$ is guaranteed to be smaller than $d_\phi(Q, U)$ and the total weight of $(E_1, E_1')$ is 0.2, from the next level we only need to find the $(\phi - 0.2)$-quantile distances in the remaining pairs of instances. This is the basic idea of our algorithm.

*Algorithm Description.* We outline our algorithm in Algorithm 3 below in a recursive fashion. Note that the input of the algorithm is a collection of pairs of entries - initially, $R_Q \times R_U$ where $R_Q$ ($R_U$) is the set of the entries in the root of the local aR-tree of $Q$ ($U$).

**Algorithm 3**: QUANTILE-DISTANCE ($R_Q \times R_U$, $\phi$)

```
   Input  : R_Q × R_U; φ: 0 < φ ≤ 1;
   Output : d_φ(Q, U)
1  θ := 0; T_1 := ∅; T_2 := ∅;
2  if R_Q × R_U only contains leaf entries then
3      call Algorithm 1 on R_Q × R_U and φ
4  else
5      Calculating d_φ^lo and d_φ^up regarding R_Q × R_U;
6      for each (E, E') ∈ R_Q × R_U do
7          if d^up(E, E') ≥ d_φ^lo and d^lo(E, E') ≤ d_φ^up then
8              T_1 := {(E, E')} ∪ T_1
9          else if d^up(E, E') < d_φ^lo then
10             θ := θ + w(E) × w(E')
11     for each (E, E') ∈ T_1 do
12         T_2 := T_2 ∪ ENUMERATING(E, E')
13     QUANTILE-DISTANCE (T_2, φ − θ);
```

In Algorithm 3, lines 7 and 8 remove the pairs, with their maximum distances smaller than $d_\phi^{lo}$ or minimum distances larger than $d_\phi^{up}$, from $T_1$ (i.e. no further exploring). Lines 9 and 10 cumulatively record the total weights $\theta$ of removed pairs of entries with the maximum distance smaller than $d_\phi^{lo}$. Lines 11 and 12 enumerate all the remaining pairs of entries in the next level for the next iteration; this will be shown in the example below. To ensure the correctness, in the next iteration, we compute the $(\phi - \theta)$-quantile distance from remaining pairs of instances.

*Example 3:* Continue the example (Figure 5) in the part of Basic Idea. Using Algorithm 3, $T_1 = \{(E_1, E_2'), (E_2, E_1')\}$ in the 1st iteration and $\theta = 0.2$.

Assume that the child node, NODE($E_1$), of $R_Q$ corresponding to the entry $E_1$ has two entries $\{E_{1,1}, E_{1,2}\}$, NODE($E_2$) contains $\{E_{2,1}, E_{2,2}\}$, NODE($E_1'$) contains $\{E_{1,1}', E_{1,2}'\}$, and NODE($E_2'$) contains $\{E_{2,1}', E_{2,2}'\}$. In Algorithm 3, ENUMERATING($E_1, E_2'$) generates the 4 pairs of entries, $\{(E_{1,1}, E_{2,1}')$, $(E_{1,1}, E_{2,2}')$, $(E_{1,2}, E_{2,1}')$, $(E_{1,2}, E_{2,2}')\}$. Similarly, ENUMERATING($E_2, E_1'$) generates the 4 pairs of entries, $\{(E_{2,1}, E_{1,1}')$, $(E_{2,1}, E_{1,2}')$, $(E_{2,2}, E_{1,1}')$, $(E_{2,2}, E_{1,2}')\}$. These 8 pairs (in $T_2$) together with $(\phi - 0.2)$ are sent to the next iteration - QUANTILE-DISTANCE ($T_2$, $\phi - 0.2$). □

*Remark 1:* If $\{E_1, E_2\}$ are the leafs (i.e. points), then they have no corresponding children nodes. In this case, ENUMERATING($E_1, E_2'$) and ENUMERATING($E_2, E_1'$) generate 4 pairs of entries in total: $\{(E_1, E_{2,1}')$, $(E_1, E_{2,2}')$, $(E_2, E_{1,1}')$, $(E_2, E_{1,2}')\}$. Similarly, if $\{E_1', E_2'\}$ are the leafs, then the following 4 pairs of entires are generated for the next iteration: $\{(E_{1,1}, E_2')$, $(E_{1,2}, E_2')$, $(E_{2,1}, E_1')$, $(E_{2,2}, E_1')\}$. □

*Calculating $d_\phi^{lo}$ and $d_\phi^{up}$.* In Algorithm 3, at each iteration we need to calculate $d_\phi^{lo}$ and $d_\phi^{up}$ (line 5) except that all entries are at the leaf level. Assume that at the $j$th iteration, there

are $l$ pairs of entries left; that is, $T_2 = \{t_i \mid 1 \leq i \leq l\}$ - each $t_i$ with the form $(E, E')$ where $E$ is an entry from the local aR-tree of $Q$ and $E'$ is an entry from the local ar-tree of $U$. Recall that the maximum distance $d^{up}(t_i)$ and the minimum distance $d^{lo}(t_i)$ are defined on a pair $t_i$ of entries in Section III. Suppose that in the current iteration, we want to compute the $\phi'$-quantile distance $d_{\phi'}(\mathcal{T})$ in $\mathcal{T}$ where $\mathcal{T}$ denotes the collection of pairs of instances each of which has an element in $T_2$ as the ancestor; that is, for each pair of instances $(q, u) \in \mathcal{T}$, $\exists (E, E') \in T_2$ such that $E$ contains $q$ and $E'$ contains $u$.

Let the $\phi'$-quantile of $T_2$, regarding the search key $d^{lo}(t_i)$, be $(EL, EL')$, and the $\phi'$-quantile of $T_2$, regarding the search key $d^{up}(t_i)$, be denoted by $(EU, EU')$.

*Theorem 1:* $d^{lo}(EL, EL') \leq d_{\phi'}(\mathcal{T}) \leq d^{up}(EU, EU')$.

*Proof:* According to the definition of the $\phi'$-quantile distance, it is immediate that $\sum_{d^{lo}(t) \leq d_{\phi'}(\mathcal{T}), t \in T_2} w(t) \geq \phi'$; consequently, $d^{lo}(EL, EL') \leq d_{\phi'}(\mathcal{T})$.

Similarly, according to the definition of $\phi'$-quantile distance, $\sum_{d^{up}(t) < d_{\phi'}(\mathcal{T}), t \in T_2} w(t) < \phi'$. Therefore, $d_{\phi'}(\mathcal{T}) \leq d^{up}(EU, EU')$. ■

Theorem 1 implies that $d^{lo}(EL, EL')$ and $d^{up}(EU, EU')$ are a lower-bound and an upper-bound, respectively, of $d_{\phi'}(\mathcal{T})$. Thus, in line 5 of Algorithm 3, we calculate $d^{lo}(EL, EL')$ and $d^{up}(EU, EU')$. Clearly, this can be done by Algorithm 1 in linear time.

*Time Complexity.* In each iteration, our algorithm is linear regarding $|T_2|$; that is, $O(|T_2|)$. Since the total entries in the local aR-trees of $Q$ and $U$ are $(|Q|)$ and $O(|U|)$, respectively, Algorithm 3 runes in linear time regarding $|Q \times U|$; that is, $O(|Q \times U|)$.

*Correctness.* The following theorem can be immediately verified based on the definition of $\phi$-quantile distance.

*Theorem 2:* Let $\theta$ denote the total weights of the pairs of entries so far pruned by $d_\phi^{lo}$ at each iteration. Then, $d_{\phi-\theta}(\mathcal{T}) = d_\phi(Q, U)$ where $\mathcal{T}$ consists of all remaining pairs of instances after the current iteration.

Theorem 1 and Theorem 2 imply that Algorithm 3 is correct; that is, it can produce $d_\phi(Q, U)$.

*Filtering while Enumerating.* Algorithm 3 can be improved when enumerating children pairs in line 12 - ENUMERATING($E, E'$). For every enumerated pair $t$ of children entries, before adding to $T_2$ we check if it can be pruned by the current distance lower and upper bounds. Then $T_2$ keeps only the remaining pairs of children entries for the next iteration. Note that in $\theta$, we also include the total weights of children pairs pruned by the current lower bound $d_\phi^{lo}$.

*Example 4:* Continue Example 3. The enumerated 8 pairs are depicted in Figure 6. The pair $(E_{2,1}, E'_{1,1})$ with the weight 0.1 is pruned by the current $d_\phi^{lo}$. Consequently, the remaining 7 pairs (put in $T_2$) and $(\phi - 0.2 - 0.1)$ are used to call QUANTILE-DISTANCE () for the next iteration.

Note that the time complexity of Algorithm 3, by adding the technique "Filtering while Enumerating", remains the same $O(|Q| \times |U|)$.

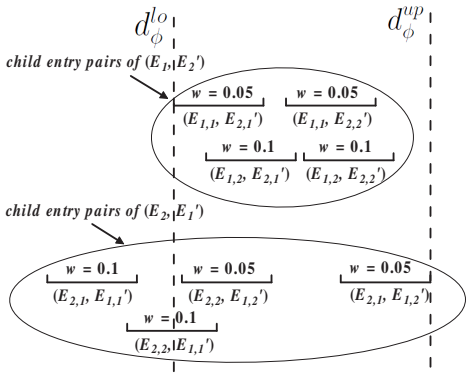



Fig. 6. Filtering while Enumerating

### B. Refinement Algorithm

In the seeding phase, after $d_\phi(Q, U)$ is calculated for each $U$ of chosen $K$ objects. A max_heap maintains the $K$ objects based on their $\phi$-quantile distances; $\gamma_K$ is the maximum of these $K$ $\phi$-quantile distances and sits on the top. Our refinement algorithm for generating the final result for $\phi$-quantile KNN is outlined below in Algorithm 4. In the algorithm, we effectively use $\gamma_K$ and $\phi$ to prune as many entries, in the global R-tree on MBBs of objects and local aR-trees, as possible. Since "closer" objects have a better chance to be in the final result of KNN (thus a better chance to reduce $\gamma_K$), we traverse the global R-tree based on the priority that an entry $E$ with the smallest minimum distance to the MBB of $Q$ will be visited first. This can be done by maintaining a heap $H$ on the currently extended entries.

**Algorithm 4**: Refinement

```
1  while H ≠ ∅ do
2      E := deheap(H);
3      if not PRUNED1(Q, E) then
4          if E is an intermediate entry then
5              add MBBs of the child entries to H
6          else    /* E corresponds to an object U */
7              if not PRUNED2(Q, E) then
8                  call Algorithm 3 to compute d_φ(E, Q);
9                  if d_φ(Q, E) < γ_K then
10                     update current KNN
11
```

In Algorithm 4, we initially load to $H$ the entries MBBs in the root node of the global R-tree. Then we iteratively apply PRUNED1($E, Q$) to the heap top $E$ by using the pruning rules below in Section IV-C for the global R-tree. If $E$ cannot be pruned (i.e. PRUNED1($Q, E$) returns FALSE), then we add to $H$ the entries of the child node with $E$ as the MBB when $E$ is an intermediate entry. When $E$ cannot be pruned and $E$ is the MBB of an object $U$, we apply the pruning rules below in Section IV-C on an individual object, PRUNED2($Q, E$), to

prune $U$ (PRUNED2($Q, E$) returns TRUE if pruned) or "trim" the entries in the local aR-tree of $U$.

### C. Pruning Rules

Pruning Rules 1 and 2 attempt to prune an entry in the global R-tree, while Pruning Rule 3 is to further examine the "details" of a remaining object using its local aR-tree.

**Pruning an Entry $E$ of Global R-tree.** Pruning Rules 1 and 2 below use $Q$ to prune an entry $E$ of the global R-tree. The correctness of Pruning Rule 1 is immediate.

*Pruning Rule 1.* (*Distance based:*) If $d^{lo}(E_Q, E) \geq \gamma_K$, then $E$ can be pruned where $E_Q$ is the MBB of $Q$ and $E$ is an entry of the global R-tree. (E is pruned means that all objects indexed by $E$ can be pruned).

Note that the minimum distance between two MBBs is defined in Section III and can be calculated in constant time. Next, we present Pruning Rule 2.

Regarding the local aR-tree $aR_Q$ of $Q$, a set $\Gamma$ of entries in $aR_Q$ is a *$\gamma_K$-cover* of $aR_Q$ if 1) there are no 2 entries in $\Gamma$ with the descendent relationship, 2) for each $E_i \in \Gamma$, $d^{lo}(E_i, E) \leq \gamma_K$, and 3) for each entry $E'$ which is not an ancestor nor a descendent of any entry in $\Gamma$, $d^{lo}(E', E) > \gamma_K$. The following theorem is immediate from the definition of $\phi$-quantile distance.

*Theorem 3:* Let $\Gamma$ be a $\gamma_K$-cover of $R_Q$. If $\sum_{E' \in \Gamma} w(E') \leq \phi$, then for each $U \in E$, $d_\phi(Q, U) \geq \gamma_K$.

Clearly, if we can find a $\gamma_K$-cover satisfying the condition in Theorem 3, then $E$ can be pruned.

*Pruning Rule 2.* (*Weights based:*) If there is a $\gamma_K$-cover $\Gamma$ with $\sum_{E' \in \Gamma} w(E') \leq \phi$, then $E$ can be pruned.

Note that there could be many $\gamma_K$-covers as shown in Example 5.

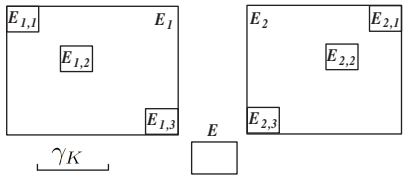



Fig. 7. $\gamma_K$-Cover

*Example 5:* As depicted in Figure 7, the $\gamma_K$-covers can be $\{E_1, E_2\}$, $\{E_1, E_{2,3}\}$, $\{E_{1,3}, E_2\}$, $\{E_{1,3}, E_{2,3}\}$. If $E_{1,3}$ and $E_{2,3}$ have child entries, more alternatives could be enumerated. They possibly have different total weights.

A $\gamma_K$-cover $\Gamma$ is *minimum* if $\sum_{E' \in \Gamma} w(E')$ is minimized. In example 5, $\{E_{1,3}, E_{2,3}\}$ has the smallest weight among those 4 covers. Clearly, the minimum $\gamma_K$ has the maximal pruning power since $\sum_{E' \in \Gamma} w(E')$ is minimized.

*Executing Pruning Rule 2.* Although a minimum $\gamma_K$-cover can be computed by traversing $aR_Q$ level-by-level from the root, we will not always try to get a minimum $\gamma_K$-cover if $E$ can be pruned earlier. We visit $aR_Q$ level-by-level from the root. At each level $i$, we generate a todo list $TD_i$ (initially $\emptyset$), and remove/trim the child entries $E'$ of the entries in $TD_{i-1}$, if $d^{lo}(E', E) > \gamma_K$. For each remaining child entry $E'$ (not trimmed), $E'$ with $d^{up}(E', E) \leq \gamma_K$ will not be extended at the next level since all its decedent entries always have their minimum (and maximum) distances not greater that $\gamma_K$ - we cumulate $w(E')$ in $\Delta$; $E'$ with $d^{up}(E', E) > \gamma_K$ will be extended in the next level for further trimming (thus, it is put into $TD_i$). $E$ is pruned and we terminate the execution of Pruning Rule 2 if the value of $\Delta$ plus the total weights of the entries in $TD_i$ is not greater than $\phi$. Note that if $E$ cannot be pruned, then the execution terminates if either the the current $TD_i$ is empty or at the leaf level. Moreover, at the root level (i.e. $i = 1$), we assume that $TD_0$ consists of the MBB $E_Q$ of $Q$.

Clearly, the execution of Pruning Rule 2 terminates if $E$ is pruned or the minimum value of $\gamma_K$-cover is obtained ($\Delta$ + the total weights in current $TD$). If $E$ is the MBB of an object $U$ (i.e. corresponds to $U$) and $U$ cannot be removed, then we record the obtained total weights of the minimum $\gamma_K$-cover in $\Delta_Q$ and record its trimmed aR-local tree by $aR_{Q,trim}$. $\Delta_Q$ will be used in the next pruning rule, and $aR_{Q,trim}$ will be used in line 8 to call Algorithm 3 if $U$ cannot be pruned by the next pruning rule.

*Example 6:* Continue Example 5 regarding Figure 7. Suppose that the root of $aR_Q$ contains entires $E_1$ and $E_2$. $TD_0 = \{E_Q\}$. At the root level, we obtain $TD_1 = \{E_1, E_2\}$ regarding the depicted $\gamma_k$. At the next level, $E_{1,1}$, $E_{1,2}$, $E_{2,2}$, $E_{2,1}$ are trimmed; consequently, $TD_2 = \{E_{1,3}, E_{2,3}\}$ and $\Delta = 0$ if $d^{up}(E_{1,3}, E) > \gamma_K$ and $d^{up}(E_{2,3}, E) > \gamma_K$. If $w(TD_2) < \phi$, then $E$ will be pruned; otherwise we go to the next level for further exploring.

In case that $d^{up}(E_{1,3}, E) \leq \gamma_K$ and $d^{up}(E_{2,3}, E) \leq \gamma_K$, $TD_2$ remains $\emptyset$ and $\Delta = w(E_{1,3}) + w(E_{2,3})$. If $\Delta > \phi$ then $E$ cannot be pruned. Since $TD_2 = \emptyset$, the execution of Pruning Rule 2 terminates. If $E$ is an object, then we record $\Delta_Q$ and $aR_{Q,trim}$. Here, $\Delta_Q = w(E_{1,3}) + w(E_{2,3})$, and in $aR_{Q,trim}$, $E_{1,1}$, $E_{1,2}$, $E_{2,1}$, $E_{2,3}$ are pruned/trimmed. □

*Remark 2:* When we trim/remove entries of R-trees, we do a "logic" removal by commenting them out.

*PRUNED1($Q, E$).* For each entry, we first check Pruning Rule 1 - PRUNED1($Q, E$) returns TRUE if $E$ is pruned. If $E$ cannot be pruned by Pruning Rule 1, then we invoke the above execution of Pruning Rule 2; PRUNED1($Q, E$) returns TRUE if $E$ is pruned.

**Trimming the Local aR-Tree of $U$.** Before conducting the computation of $\phi$-quantile distance by Algorithm 3, we first trim the entries of the local aR-tree by $\gamma_K$. We conduct this in a level-by-level fashion from the root of the local aR-tree in the same way as the execution of Pruning Rule 2 except that we swap the role $Q$ with $U$; that is, $Q$ becomes $E$, and $U$ becomes $Q$ in the execution of Pruning Rule 2. At each level $i$ of aR-tree of $U$, we check the flowing pruning rule.

*Pruning Rule 3.* (*Using Local aR-tree:*) If $(\Delta + w(TD_i)) \times$

$\Delta_Q \leq \phi$, then $U$ can be pruned. [3]

*Proof:* From the definition of $\phi$-quantile distance, it is immediate that if $(\Delta + w(TD_i)) \times \Delta_Q \leq \phi$, then $d_\phi(Q, U) \geq \gamma_K$. ■

*PRUNED2($Q, E$).* As described above, the execution of Pruning Rule 3 is the same as the execution of Pruning Rule 2 except that we swap the roles of $Q$ and $U$ and check Pruning Rule 3 instead of Pruning 2 at each level. PRUNED2($Q, E$) terminates and returns TRUE if $E$ is pruned; otherwise, PRUNED2($Q, E$) terminates at the leaf-level (or $TD_i = \emptyset$) and returns FALSE. When PRUNED1($Q, E$) returns FALSE, $R_{Q,trim} \times R_{U,trim}$ is used as the input of Algorithm 3 for computing $d_\phi(Q, U)$ instead of using $R_Q \times R_U$. Here, $R_{Q,trim}$ ($R_{U,trim}$) consists of the untrimmed entries at the root of $aR_{Q,trim}$ ($aR_{U,trim}$). Note that in Algorithm 3, level-by-level we use only untrimmed entries from both $aR_{Q,trim}$ and $aR_{U,trim}$. We can further speed-up the computation by visiting only the intermediate nodes, in $aR_{Q,trim}$ and $aR_{Q,trim}$, respectively, with more than one child.

The correctness of Algorithm 4 immediately follows from the theorems and pruning rules. Note that when Algorithm 3 is invoked, at each iteration we use the minimum value of $\gamma_K$ and the obtained upper-bound $d_\phi^{up}$ as an upper-bound.

## V. $\phi$-QUANTILE GROUP-BASE KNN

Our algorithm for solving the $\phi$-quantile group-base KNN ($\phi \in (0, 1]$) (defined in Section II-A) also follows the seeding-refinement framework, Algorithm 2. For the seeding-phase, firstly we show that computing a $\phi$-quantile group-base distance $gbd_\phi(Q, U)$ between $Q$ and $U$ is NP-hard, and then an existing algorithm is employed with the approximation factor 2 to approximately compute $gbd_\phi(Q, U)$. In the refinement phase, 2 novel, effective pruning techniques are developed.

### A. Computing $\phi$-Quantile Group-base Distances

We first show that the *Knapsack Problem* can be converted to a special case of our problem.

**Knapsack Problem.** It is NP-complete and can be formally described below [11].

INSTANCE: Finite set $S$, for each element $s \in S$, an integer size $c(s)$, and an integer value $v(s)$, and positive integers $X$ and $Y$.

QUESTION: Is there a subset $S'$ of $S$ such that $\sum_{s \in S'} c(s) \leq X$ and $\sum_{s \in S'} v(s) \geq Y$.

**NP-hardness.** As defined in Section II-A, the problem of computing $gbd_\phi(Q, U)$ can be stated below. Find a subset $S'$ from $Q \times U$ such that $\sum_{(q,u) \in S'} w(q, u) \geq \phi$ and $\sum_{(q,u) \in S'} w(q, u) d(q, u)$ is minimized.

A special case of the problem of computing $gbd_\phi(Q, U)$ is that $Q$ is a point with weight 1. In this case, each $w(u)$ ($= w(Q, u)$) may be arbitrarily assigned with the constraint $\sum_{u \in U} w(u) = 1$, and the location of each $u$ can be chosen so that $w(Q, u)d(Q, u)$ equals any integer.

If we normalize the above Knapsack Problem by normalizing each $v(s)$ by $\frac{v(s)}{\sum_{s' \in S} v(s')}$. Then the normalized version of Knapsack is also NP-complete. The *decision* problem of the above spacial case of computing $gbd_\phi(Q, U)$ is the same as the normalized Knapsack Problem. Consequently, the problem of computing $gbd_\phi(Q, U)$ is NP-hard.

*Theorem 4:* The problem of computing $gbd_\phi(Q, U)$ is NP-hard.

**Approximately Computing $gbd_\phi(Q, U)$.** If we want to maximize $\sum_{s \in S'} v(s)$ with respect to a given $X$ in the Knapsack Problem, then there is PTAS; that is, a polynomial-time approximation scheme giving an approximate factor arbitrarily closer to 1. Nevertheless, there is no PTAS to approximately minimize $\sum_{s \in S'} c(s)$ regarding a given $Y$.

We adopt the approximate algorithm in [13] for Knapsack Problem. It runs in time $O(m \log m)$, where $m$ is the number of elements in $S$, with the approximation factor 2 for minimizing $\sum_{s \in S'} c(s)$ for a given $Y$. The algorithm can be immediately used to approximately compute $gbd_\phi(Q, U)$ if we treat $Q \times U$ as $S$; and for each $(q, u) \in Q \times U$, treat $w(q, u)$ as a $v$ value and treat $w(q, u)d(q, u)$ as a $c$ value in the Knapsack Problem. Let $aproxgbd_\phi(Q, U)$ denote the group distance output by the approximation algorithm. The following theorem is shown [13].

*Theorem 5:* $1 \leq \frac{aproxgbd_\phi(Q,U)}{gbd_\phi(Q,U)} \leq 2$.

We briefly present the basic idea of the algorithm in [13] while applying it to computing $gbd_\phi$. It iteratively conducts 2 phases: Completion and Growing Seed-Set $ST$ - initially $\emptyset$ ($w(ST)$ is always smaller than $\phi$). We firstly sort $Q \times U$ increasingly based on $d(q, u)$. In the Completion phase, for each element $(q, u)$ in the remaining $Q \times U$ with $w(q, u) + w(ST) \geq \phi$, 1) replace the current feasible solution $S'$ if the total weighted distance in $ST \cup \{(q, u)\}$ is smaller than that in $S'$, and 2) remove $(q, u)$ from $Q \times U$. In Growing Seed-Set $ST$, move the 1st element from the remaining $Q \times U$ to $ST$. In each iteration, we first conduct Completion and then Growing Seed-Set; the algorithm terminates and outputs the total weighted distance in $S'$ if there is no element left in the remaining $Q \times U$.

*Example 7:* Suppose that $\phi = 0.5$ and $Q \times U$ contains 4 elements. To simplify the presentation, we present these 4 elements only by its ($distance$, $weight$): $\{(1, 0.28), (2, 0.12), (3, 0.48), (4, 0.12)\}$. In our algorithm, we first sort the list increasingly based on the value of $\frac{weight \times distance}{weight} = distance$.

In the 1st iteration, nothing is chosen in the Completion phase since all elements with weight less than 0.5; $ST$ becomes $\{(1, 0.28)\}$ and $(1, 0.28)$ is removed from $Q \times U$ in the Growing Seed-Set phase. In the 2nd iteration, $S' = \{(1, 0.28), (3, 0.48)\}$ is chosen as a feasible solution and $(3, 0.48)$ is removed $Q \times U$ in the Completion phase; $ST$ grows to $\{(1, 0.28), (2, 0.12)\}$ and $(2, 0.12)$ is removed from $Q \times U$ since $(2, 0.12)$ was the 1st element. In the 3rd iteration, regarding Completion phase, $(4, 0.12)$ is removed from $Q \times U$ as $w(ST) + 0.13 = 0.52 > 0.5$ and $\{(1, 0.28), (2, 0.12),$

[3] Here, $\Delta_Q$ is obtained as the weight of the minimum $\gamma_K$-cover of the local aR-tree of $Q$. $\Delta$ and $TD_i$ are recorded when execute the Pruning Rule 2 at level $i$ and swap the roles of $Q$ and $U$ as described above.

$(4, 0.12)\}$ becomes $S'$ as its total weighted distance (1) smaller than that (1.72) in $S' = \{(1, 0.28), (3, 0.48)\}$. Consequently, 1 is output as $approxgbd_{0.5}(Q, U)$; in this example it happens $approxgbd_{0.5}(Q, U) = gbd(Q, U)$. □

Note that this approximate algorithm does not accommodate a pruning-based level-by-level computation of $gbd_\phi(Q, U)$ because it requires to access all elements.

### B. Refinement

In the seeding phase, we use the above approximate algorithm to approximately compute $gbd_\phi(Q, U)$ between $Q$ and each of the chosen $K$ objects. The largest obtained $aproxgbd_\phi$ value is denoted as $\lambda_K$. The refinement algorithm follows the similar framework outlined in Algorithm 4 in Section IV-B except that:

- In PRUNDE1($Q, E$) we will use the pruning rules below.
- remove line 7.
- call the above algorithm to (approximately) compute $gbd_\phi(Q, U)$ instead of Algorithm 3.
- use $aproxgbd_\phi$ generated by the above approximate algorithm and $\lambda_K$ to replace $d_\phi$ and $\gamma_K$, respectively.

In the group with its total weighted distance $gbd_\phi(Q, U)$, instances may be from many different entries of the local aR-tree of $U$. Consequently, it is not always possible to trim many entries (subtrees) from the local aR-tree as what we do for computing $\phi$-quantile KNN. Thus, in our refinement algorithm we only develop pruning rules to prune entries in the global R-tree.

*Pruning Rule 4.* Suppose that $E_Q$ is the MBB of $Q$. If $\phi \times d^L(E_Q, E) \geq \lambda_K$, then $E$ is pruned from the global R-tree.

The next pruning rule is used at each level. Suppose that $L_k = \{E_i \mid 1 \leq i \leq l\}$ consists of all the entries at the level $k$ of the local aR-tree of $Q$. Without loss of generality, we assume that $L_k$ is sorted in the increasing order based on $d^L(E_i, E)$; that is, $d^L(E_{i1}, E) \leq d^L(E_{i2}, E)$ if $i1 < i2$. Let $E_j$ denote the $\phi$-quantile of $L_k$ according to the search key $d^L(E_i, E)$ and the weight $w(E_i)$ of each element $E_i \in L_k$.

*Pruning Rule 5.* $E$ is pruned if:

$$(\phi - \sum_{i=1}^{j-1} w(E_i)) d^L(E_j, E) + \sum_{i=1}^{j-1} (w(E_i) \times d^L(E_i, E)) \geq \lambda_K.$$

**Executing PRUNDE1($Q, E$).** For an $E$ in the global R-tree, we first check Pruning Rule 4; this is done by constant time. If $E$ cannot be pruned, then we traverse the local aR-tree of $Q$ level-by-level from the root to test Pruning Rule 5. To test Pruning Rule 5 at each level $k$, we first need to sort $L_k$. The total time complexity for traversing the local aR-tree of $Q$ to test Pruning Rule 5 is thus $O(|Q|)$.

**Accuracy Guarantee.** Our algorithm for solving $\phi$-quantile group-base KNN has the following accuracy guarantee.

*Theorem 6:* Suppose that for $1 \leq i \leq k$, $U_i$ is ranked the top-$i$th in the exact $\phi$-quantile group-base KNN, and $U'_i$ is ranked the top-$i$th by our algorithms. Then for $1 \leq i \leq k$, $gbd_\phi(Q, U_i) \leq aproxgbd_\phi(Q, U'_i) \leq 2gbd_\phi(Q, U_i)$.

*Proof:* First, it can be immediately verified that the object $U$ pruned (i.e., the entry $E$ containing $U$ is pruned) by Pruning Rule 4 or Pruning 5 has the property that $gbd_\phi(Q, U) \geq \lambda_K$. From Theorem 6, it follows that for $1 \leq i \leq k$, $gbd_\phi(Q, U_i) \leq aproxgbd_\phi(Q, U'_i) \leq 2gbd_\phi(Q, U_i)$. ■

Theorem 6 states that every $i$th group-base distance ($i \in [1, K]$) output by our algorithm is between $gbd_\phi(Q, U_i)$ and $2gbd_\phi(Q, U_i)$. Our experiment, nevertheless, indicates the error could be much smaller in practice.

## VI. Experimental Study

We report a thorough performance evaluation on the efficiency and effectiveness of our algorithms. In particular, we implement and evaluate the following techniques.

- Q-KNN: Techniques presented in Section IV to compute KNN based on a $\phi$-quantile distance ($\phi \in (0, 1]$).
- Naive Q-KNN: Remove the pruning rules from Q-KNN.
- G-KNN: Techniques in Section V to compute KNN based on $\phi$-quantile group-base distances.
- Naive G-KNN: Remove the pruning rules from G-KNN.

All algorithms are implemented in C++ and compiled by GNU GCC. Experiments are conducted on PCs with Intel Xeon 2.4GHz dual CPU and 4G memory under Debian Linux. Our experiments are conducted on both real and synthetic datasets.

**Real dataset** is extracted from NBA players' game-by-game statistics (http://www.nba.com), containing 339,721 records of 1,313 players. Each player is treated as a multi-valued object where the statistics (score, assistance, rebound) of a player per game is treated as an instance with the equal weight (normalized).

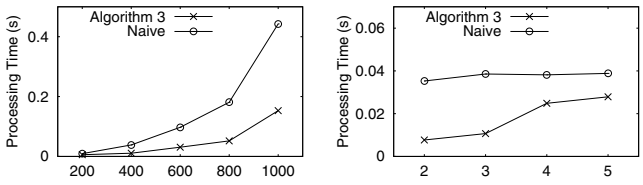
**Synthetic datasets** are generated using the methodologies in [4] regarding the following parameters. Dimensionality $d$ varies from 2 to 5 with default value 3. Data domain in each dimension is $[0, 1]$. Number $n$ of objects varies from $10,000$ to $50,000$ with default value $10,000$. Number $m$ of instances per object follows a uniform distribution in [1, $\mathcal{M}$] where $\mathcal{M}$ varies from $400$ to $2,000$ with the default value $400$. The value $K$ varies among 5, 10, 20, 30 and 40 with default value 10. The average length of object MBBs follows a *uniform* or *normal* distribution. In normal distribution, the length of MBB lies in the range $[0, h]$ with the expectation value $h/2$ and standard deviation $0.025$; in uniform distribution, the length of MBBs uniformly spreads over [0, $h$] where $h$ varies from $0.05$ to $0.25$ with default value $0.05$ (i.e., $5\%$ of the edge length of the whole data space). With the default setting, the total number of instances is about 2 millions.

Centers of objects (objects' MBBs) follow either *uniform*, *normal* or *anti-correlated* distribution. Locations of instances in an object follow *uniform* or *normal* distribution. Weights assigned to each instance follow *uniform* or *normal* distribution. Table II summarizes the parameters used in our experiment where the default values are in **bold** font. *For each experiment, we randomly choose* 100 *objects from datasets as query objects and record the average performance. Note that*

*default values will be used in our experiment unless otherwise specified.*

| | |
|---|---|
| dimensionality $d$ | 2, **3**, 4, 5 |
| number of objects $\mathcal{N}$ | **10k**, 20k, 30k, 40k, 50k |
| edge length $h$ | **0.05**, 0.1, 0.15, 0.2, 0.25 |
| number of instances $m$ | **400**, 600, 800, 1k, 2k |
| $K$ | 5, **10**, 15, 20, 30 |
| $\phi$ | 0.1, 0.3, **0.5**, 0.7, 0.9 |
| object location | uniform, normal, **anti-correlated** |
| instance location | **uniform**, normal |
| weight distribution | uniform, **normal** |
| $h$ distribution | **uniform**, normal |

TABLE II
PARAMETER VALUES.




(a) Varying $\mathcal{M}$ (b) Varying $d$

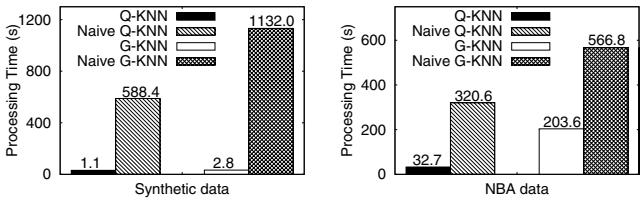
Fig. 8. Time for Computing $d_\phi$

### A. Computing $\phi$-Quantile Distance

Figure 8 evaluates the efficiency of our technique, Algorithm 3, for computing a $\phi$-quantile distance, against the naive algorithm described in Section IV-A. In our experiment, we randomly select 1000 pairs of objects from the datasets to test these 2 algorithms and report the average time by seconds. Figure 8(a) shows that our technique has more advantages when the number of instances increases. Figure 8(b) shows that the advantage of using Algorithm 3 gets lower when dimensionality increases. This is because that the pruning costs in Algorithm 3 are proportional to the dimensionality. When dimensionality increases, more pruning overheads are involved. Nevertheless, Figure 8 indicates Algorithm 3 significantly outperforms the naive algorithm. Therefore, we always use Algorithm 3 in the remaining experiments. Note that we did not evaluate the techniques in [26] since they are not generally applicable to our problem.

### B. Overall Performance

Figure 9 reports the results of the evaluation on processing time of Q-KNN, Naive Q-KNN, G-KNN, Naive G-KNN over real and synthetic datasets. As shown, Q-KNN and G-KNN are much more efficient than their naive versions (i.e. without using pruning techniques in the refinement phase) - upto 2 orders of magnitude. The improvement is less significant over NBA data. This is because in NBA dataset, objects' MBB sizes are very large relative to the whole data space; this gives very high overlapping degree among objects' MBBs. Thus less objects can be pruned during query processing.




(a) (b)

Fig. 9. Overall Performance

We further evaluate the pruning powers in the refinement phase by conducting the following experiment. Regarding the $\phi$-quantile KNN, we examine the running time of Naive Q-KNN, Naive Q-KNN with the Pruning Rule 1 (P1), Naive Q-KNN with the Pruning Rules 1 and 2 (P1-2), and the Naive Q-KNN with the Pruning Rules 1, 2, and 3 (P1-3, that is, Q-KNN). Similarly, for $\phi$-quantile group-base KNN, Naive G-KNN, Naive G-KNN with the Pruning Rule 4 (P4), and Naive G-KNN with the Pruning Rules 4 and 5 (P4-5, that is, G-KNN) are examined. The evaluation results are depicted in Figure 10. It shows that all these pruning rules are very effective and efficient. These 2 experiments indicate that Q-KNN and G-KNN are much more efficient than Naive Q-KNN and Naive G-KNN, respectively. Thus, in the rest of experiments we will no longer evaluate Naive Q-KNN and Naive G-KNN. Experiments are also conducted against different settings (e.g., different data distributions, dimensionality, data size, etc) and all experiments demonstrate the effectiveness of each proposed pruning rule with similar trends. Results are not included in the paper due to space limits.

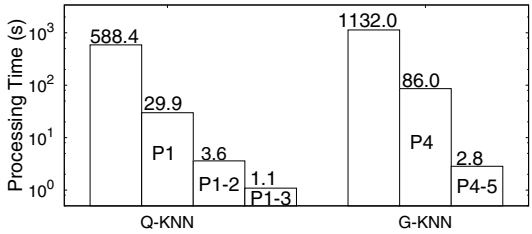



Fig. 10. Pruning Powers

### C. Accuracy

To evaluate the accuracy of G-KNN, we use two error measures. The first is the average distance error ratio. For $1 \leq i \leq K$, $approx(i)$ denotes the group-based distance of the top-$i$th object output by G-KNN, and $exact(i)$ denotes the group-based distance of the top-$i$th object in the exact solution.

$$err\_ratio = \frac{\sum_{i=1}^{K} \frac{|approx(i) - exact(i)|}{exact(i)}}{K}$$

The second measure records the "misplaced" ratio. For $1 \leq i \leq K$, if the $i$th object in the exact solution is not the same as the $i$th object in the solution output by G-KNN, then $mp(i) = 1$.

$$mp\_ratio = \frac{\sum_{i=1}^{K} mp(i)}{K}$$

As the $\phi$-quantile group-base KNN is NP-hard and no efficient algorithm exists, we generate the exact solutions by a trivial exhaustive search - it is exponential and very slow. We conduct a very small scale experiment as follows. Each object, including query object, has $4$ instances; there are total $100$ objects. Others all use the default settings in Table II. Table III reports the evaluation results when object distribution varies, while Table IV reports the results when the distribution of weights varies. Both demonstrate G-KNN is highly accurate and more accurate than the theoretical guarantee in Theorem 6; that is, err_ratio is much smaller than 2.

| | err_ratio | mp_ratio |
|---|---|---|
| anti | 0.015 | 0.02 |
| unif | 0.013 | 0.02 |
| norm | 0.015 | 0.04 |

TABLE III
VARY OBJECTS DISTRIBUTION

| | err_ratio | mp_ratio |
|---|---|---|
| unif | 0 | 0 |
| norm | 0.015 | 0.02 |

TABLE IV
VARY WEIGHT DISTRIBUTION

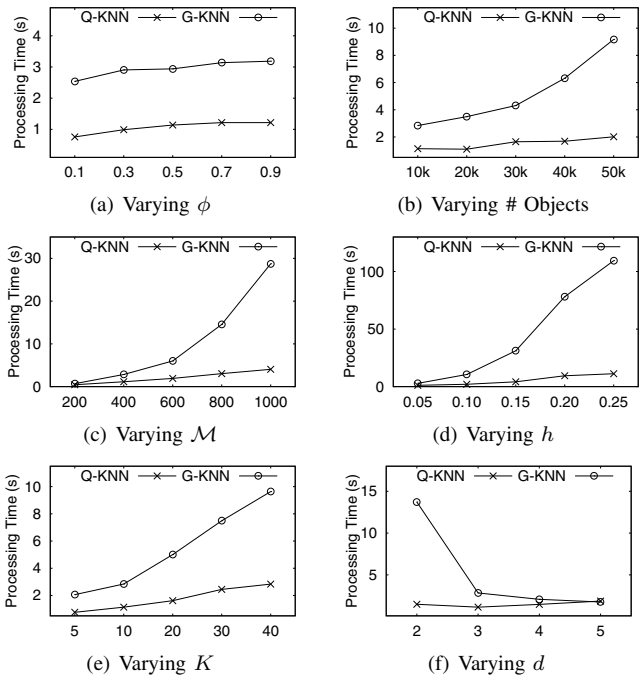
### D. Evaluating Impacts by Different Settings

**Distributions.** We evaluate possible impacts on algorithm efficiency by distributions of centers of objects, locations of instance, edge lengths of object MBBs, and weights. The results (time in seconds) for Q-KNN and G-KNN are reported in Table V, respectively. They demonstrate that Q-KNN is not quite sensitive to various distributions but G-KNN is quite sensitive towards different distributions. This is because of the nature of $\phi$-quantile group-base distance - group-base. Note that it is only meaningful for object locations to have anti-distributions; consequently, we do not evaluate other distributions using anti. Moreover, the experiment shows anti always leads to more computation time; this is the reason why we use anti as a default setting for locations.

| | Q-KNN | | | G-KNN | | |
|---|---|---|---|---|---|---|
| | unif | norm | anti | unif | norm | anti |
| object_loc | 0.9(s) | 0.8(s) | 1.1(s) | 2.3(s) | 2.0(s) | 2.8(s) |
| MBB_length | 1.1(s) | 1.2(s) | * | 2.8(s) | 2.9(s) | * |
| instance_loc | 1.1(s) | 1.0(s) | * | 2.8(s) | 2.3(s) | * |
| weights | 1.1(s) | 1.1(s) | * | 2.0(s) | 2.8(s) | * |

TABLE V
VARIOUS DISTRIBUTIONS

**Impacts by Other Settings.** In the next set of experiments, we study the scalability of our algorithms regarding different $\phi$-values, number of objects, number of instances ($\mathcal{M}$), lengths of MBB edges ($h$), $K$, and the dimensionality $d$. In our experiments, we record the average running time per query for each algorithm. While Q-KNN and G-KNN are not quite sensitive to different $\phi$-values due to the nature of the techniques developed, they are quite sensitive to the other settings especially G-KNN. The techniques in G-KNN do not have pruning rules for trimming object entries and the distance computation techniques of G-KNN do not have any pruning rules either. Thus, G-KNN is very sensitive to the increment of number of objects, number of instances, MBB lengths, and $K$. It is interesting to note that G-KNN runs faster when the dimensionality $d$ increases. This suggests that G-KNN prunes more objects in the refinement phase when $d$ increases. A possible reason is that when we fix the MBB edge length, the average area of MBBs gets smaller related to the whole data space; consequently, Pruning Rules 4 and 5 are more effective as they are group-based (thus, area based).




(a) Varying $\phi$ (b) Varying # Objects
(c) Varying $\mathcal{M}$ (d) Varying $h$
(e) Varying $K$ (f) Varying $d$

Fig. 11. Other Settings

### E. Summary

Our performance evaluation indicates that Q-KNN is very efficient and scalable. Although $\phi$-quantile group-base KNN is inherently more complex, our G-KNN techniques still perform quite efficiently. Furthermore, G-KNN is highly accurate and performs much more accurate than the theoretical bound.

## VII. RELATED WORK

Conventional NN or KNN search in a multidimensional space is fundamental in data analysis and information retrieval. Most existing techniques for NN and KNN search have been developed based on popular spatial access methods such as R-trees. The depth first search algorithm on R-trees is first proposed by Roussopoulos *et al* [23]. The best first search algorithm is proposed by Henrich [14] and is subsequently optimized by Hjaltason and Samet [15] with the I/O optimal performance guarantee. Efficient sub-linear approximation techniques (PTAS) have been proposed by Arya *et al* [2]. Many variations of KNN search have been studied in different contexts, including road networks [22], moving objects [19], continuous queries [25], etc. Conventional KNN treats every object as a point in multidimensional space.

KNN over uncertain objects are inherently different than conventional KNN where each uncertain object takes a set of mutually exclusive points in a multidimensional space. It combines a distances measure with an uncertain top-$K$ ranking model. There are two models of ranking top-$k$ uncertain tuples: 1) retrieving $k$ tuples that can co-exist in a possible world (e.g. U-top$k$) [24], and 2) retrieving tuples according to the probability that a tuple is top-$k$ or at a specific rank in all possible worlds (e.g. U-$k$Ranks and PT-$k$) [24], [17]. In KNN over uncertain objects, the probability for an uncertain object to be the KNN to a query object is cumulated over all possible worlds and is used to rank an uncertain object. A number of probabilistic models have been proposed [3], [5], [7], [20]. Recently, a new model based on expected ranks has been developed in [10] to rank top-K uncertain objects. As shown in Section I, they may not always be sensitive to the relative distributions among object instances. Ge *et al* [12] study the score distribution of top-$K$ vectors and choose most typical results from the score-probability dimensions; nevertheless, the work seems hard to be extended to objects with multiple instances.

As a tool to summarize data distribution, quantile computation has been extensively studied (e.g., [9], [26]). The most related work is the quantile computation in a multidimensional space [26]. In [26], two problems has been investigated: 1) single source query, and 2) multiple sources query. The single source problem is a special case of the problem of computing a $\phi$-quantile distance in this paper where a query object has only one instance and all instances have the same weight. The problem of multiple sources are inherently different than our problem. Thus, the techniques in [26] are not generally applicable to our problem. Nevertheless, our techniques runs in $O(n)$ time when query object has only one instance, while the single-source techniques in [26] runs in $O(n \log n)$.

## VIII. Conclusion

In this paper, we investigate the problem of KNN search over multi-valued objects. In particular, we use the quantile paradigm to retrieve KNN sensitive to the relative distribution among multi-valued objects. Two quantile KNN models have been proposed. One is based on a $\phi$-quantile ranking score (e.g. median score) and another is based on the overall ranking score of the $\phi$-quantile best population. We show that the second KNN problem is NP-hard. A set of efficient, novel techniques have been developed to process the first quantile KNN problem. Due to the NP-hardness of the second KNN problem, efficient approximate techniques with approximate factor 2 are presented. We conduct extensive experiments to illustrate the efficiency and effectiveness of our proposed techniques.

The current algorithms developed are based on main-memory computation. Although they can be immediately extended to support I/O involved computation, a possible future work may investigate I/O efficient techniques for these 2 KNN problems.

**Acknowledgement** Work of the first author is partially supported by NICTA. Work of the second author is supported by the ARC Discovery Grants (DP0987557, DP0881035, DP0987273 and DP0666428), Google Research Award and NICTA. Work of the fifth author is supported by the ARC Discovery Grants DP0987273 and DP0881779.

## References

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Conf. of Foundations of Data Organization and Algorithms 1993*.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. In *JACM 1998*.

[3] G. Bekales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. In *VLDB 2008*.

[4] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE 2001*.

[5] R. Cheng, J. Chen, M. Mokbel, and C. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE 2008*.

[6] R. Cheng, L. Chen, J. Chen, and X. Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *EDBT 2009*.

[7] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environment. In *TKDE 2004*.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms 2nd edition. chpater 9: Medians and order statistics. In *The MIT Press*.

[9] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Effective computation of biased quantiles over data streams. In *ICDE 2005*.

[10] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE 2009*.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA, 1990.

[12] T. Ge, S. Zdonik, and S. Madden. Top-k queries on uncertain data: On score distribution and typical answers. In *SIGMOD 2009*.

[13] M. M. Guntzer and D. Jungnickel. Approximate minimization algorithms for the 0/1 knapsack and subset-sum problem. In *Operations Research Letters 2000*.

[14] A. Henrich. A distance scan algorithm for spatial access structures. In *ACM GIS 1994*.

[15] G. Hjaltason and H. Samet. Distance browsing in spatial databases. In *TODS 1999*.

[16] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *SSD*, pages 83–95, 1995.

[17] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *SIGMOD 2008*.

[18] R. Kohavi and D. Sommerfield. Feature subset selection using the wrapper model: Overfitting and dynamic search space topology. In *Conf. on Kowledge Discovery and Data Mining 1995*.

[19] G. Kollos, D. Gunopulos, and V. Tsotras. Nearest neighbor queries in mobile environment. In *STDBM 1999*.

[20] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA 2007*.

[21] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP operations in spatial data warehouses. In *SSTD 2001*.

[22] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB 2003*.

[23] N. Roussopoulos, S. Kelley, and Frederick. Nearest neighbor queries. In *SIGMOD 1995*.

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

[25] X. Xiong, M. Mokbel, and W. Aref. SEA-CNN: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE 2005*.

[26] M. L. Yiu, N. Mamoulis, and Y. Tao. Efficient quantile retrieval on multi-dimensional data. In *EDBT 2006*.