Approximate Evaluation of Range Nearest Neighbor Queries with Quality Guarantee*

Chi-Yin Chow[1], Mohamed F. Mokbel[1], Joe Naps[1], and Suman Nath[2]

[1] Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA
[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
{cchow,mokbel,naps}@cs.umn.edu sumann@microsoft.com

Abstract. The range nearest-neighbor (NN) query is an important query type in location-based services, as it can be applied to the case that an NN query has a spatial region, instead of a location point, as the query location. Examples of the applications of range NN queries include uncertain locations and privacy-preserving queries. Given a set of objects, the range NN answer is a set of objects that includes the nearest object(s) to every point in a given spatial region. The answer set size would significantly increase as the spatial region gets larger. Unfortunately, mobile users in wireless environments suffer from scarce bandwidth and low-quality communication, transmitting a large answer set from a database server to the user would pose very high response time. To this end, we propose an approximate range NN query processing algorithm to balance a performance tradeoff between query response time and the quality of answers. The distinct features of our algorithm are that (1) it allows the user to specify an approximation tolerance level k, so that we guarantee to provide an answer set $\mathcal{A}$ such that each object in $\mathcal{A}$ is one of the k nearest objects to every point in a given query region; and (2) it minimizes the number of objects returned in an answer set, in order to minimize the transmission time of sending the answer set to the user. Extensive experimental results show that our proposed algorithm is scalable and effectively reduces query response time while providing approximate query answers that satisfy the user specified approximation tolerance level.

1 Introduction

Nearest-neighbor (NN) queries have been widely used in location-based services (e.g., see [1, 2, 3, 4, 5]). The problem of traditional NN queries can be defined as follows: "given a set of objects and a query location point p, find the nearest object(s) to p"; and thus, they are referred to as *point* NN queries. *Point* NN queries have been extended to find all NNs for line segments [6] and spatial regions [7, 8, 9] that are referred to as *linear* and *range* NN queries, respectively.

* This work is supported in part by the National Science Foundation under Grants IIS-0811998, IIS-0811935, and CNS-0708604.

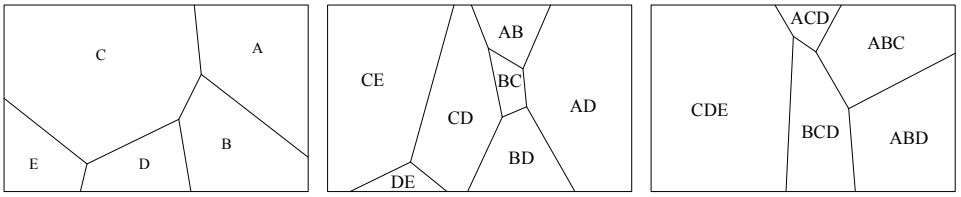
A *linear* NN query returns an answer set that includes the nearest object(s) to every point in a given line segment. On the other hand, a *range* NN query returns an answer set that includes the nearest object(s) to every point in a given spatial region, where the spatial region can be either a rectangular region [7, 9] or a circular region [8].

Recent research efforts have shown the importance of *range* NN queries in location-based services, as it can be applied to the following realistic scenarios:

- **Uncertain locations.** We have two kinds of location uncertainty, *measurement imprecision* and *sampling imprecision*. The measurement imprecision is due to the limitation of the underlying positioning techniques of network environments, e.g., 2G/3G and Wi-Fi. On the other hand, the sampling imprecision is due to continuous motion, network delays, and location update frequency even with highly accurate positioning devices, e.g., GPS. Thus, we have to use a spatial region where the user is guaranteed to be therein to represent the user location information in order to capture location uncertainty (e.g., see [10, 11, 12, 13, 14]).
- **Privacy-preserving queries.** Mobile users are not willing to reveal their exact location information to location-based service providers, as they want to preserve their location privacy. The most commonly used privacy-enhancing technique is to blur the user's exact location into a spatial region, i.e., spatial cloaking, that satisfies the user's specified privacy requirements (e.g., see [8, 9, 15, 16, 17, 18, 19]).

In these two scenarios, the mobile user sends her NN query along with a spatial region as the query location, i.e., a *range* NN query. Then, a database server returns an answer set that includes the nearest object(s) to every point within the spatial region. The answer set size would substantially increase as the query region gets larger. Unfortunately, the communication bandwidth between the user and the database server is very limited in a mobile environment, i.e., the downlink bandwidth ranges from 128 kbps at vehicular speeds to 2 Mbps at stationary or very slow speeds for 3G mobile subscribers. Transmitting large answer sets to the user would pose very high query response time. Furthermore, as mobile users receive their answer handheld devices with a small screen, it is convenient to return to the users very few answers with high quality.

In this paper, we propose a new *approximate range* NN query processing algorithm that enables the user to tune a trade off between query response time and the quality of query answers. Our proposed algorithm allows the user to specify an *approximation tolerance level* k, where we return an answer set $\mathcal{A}$ such that each object in $\mathcal{A}$ is one of the k nearest objects of every point in the query region. The larger the value of k, the smaller the answer set returned by a database server. Thus, the approximation tolerance level is a tuning parameter that trades off between query response time and the quality of answers. In the case that $k = 1$, we return the exact range NN answer of maximal size to the user. On the other hand, if $k > 1$, we return an approximate answer set, which is smaller than the exact answer, to the user; and thus, the transmission time



(a) 1-NN of each region (b) 2-NN of each region (c) 3-NN of each region

Fig. 1. A motivating example

of sending the answer set to the user is reduced. Since query response time is dominated by the communication overhead between the user and the database server, a larger value of k incurs lower query response time.

Figure 1 depicts a motivating example for our problem where we decompose the query region $Q.R$ of a range NN query Q into disjoint regions and label each region with its k-NN(s). Figure 1a shows the NN of each region. If a user wants to find the nearest object(s) to $Q.R$, i.e., the exact range NN query answer, the required answer set contains the nearest object(s) to every point in $Q.R$, i.e., $\mathcal{A}_1 = \{A, B, C, D, E\}$. Figure 1b shows the 2-NN of each region. If the user is satisfied with the 2-nd nearest object(s) to $Q.R$, the answer set $\mathcal{A}_2$ should contain at least one object among the 2 nearest objects to every point in $Q.R$. For example, if $\mathcal{A}_2 = \{A, B, C, D\}$, regardless of the actual user location within $Q.R$, the user is guaranteed to receive an object among her 2 nearest objects. However, we can still do better. For example, if $\mathcal{A}'_2 = \{A, C, D\}$, the user is still guaranteed to receive an object among her 2 nearest objects, regardless of her actual location within $Q.R$. Thus, the answer set of minimal size is a minimal set of objects such that there is at least one object among the 2 nearest objects of each region. Furthermore, if the user is satisfied with the 3-rd nearest object(s) to $Q.R$, i.e., the required answer set $\mathcal{A}_3$ should contain at least one object among the 3 nearest objects of each region, as depicted in Figure 1c, where the minimal answer set is $\mathcal{A}_3 = \{A, C\}$. Therefore, if a user accepts a higher approximation tolerance in a range NN query answer, the user will receive a smaller answer set and more user convenience.

The main idea of our proposed range NN query processing algorithm is to have an *off-line* process to pre-compute a set of k-order Voronoi diagrams, from order one to a predefined maximum order k_{max}, for a set of stationary data objects, e.g., restaurants, gas stations and hotels. For a k-order Voronoi diagram, each Voronoi cell is associated with a distinct set of k objects that are the k nearest objects to every point in the cell. To efficiently search in a Voronoi diagram, we propose an *incomplete pyramid structure* as an access method to index the Voronoi cells. Given a *range* NN query Q and an approximation tolerance level k, our proposed *on-line* range NN query processing algorithm first determines a set of Voronoi cells $\mathcal{V}$ that intersects the query region by accessing the *incomplete pyramid structure* of the relevant k-order Voronoi diagram. Then, the remaining query processing is reduced to a well-known set-covering problem where we use a greedy approach to select the minimal set of objects, i.e., the answer set $\mathcal{A}$,

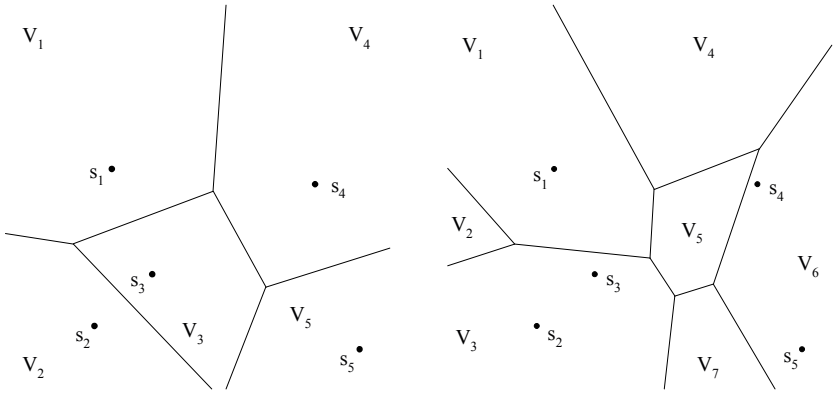
from the objects associated with the Voronoi cells in $\mathcal{V}$ such that at least one object from each Voronoi cell in $\mathcal{V}$ is selected. As a result, each object in $\mathcal{A}$ is one of the k nearest objects to every Voronoi cell in $\mathcal{V}$, i.e., each object in $\mathcal{A}$ is one of the k nearest objects to every point in the query region. With a larger value of k, there are more common objects associated among the Voronoi cells in $\mathcal{V}$; and hence, we would get smaller answer sets that incur lower query response time. In general, the contributions of this paper can be summarized as follows:

- We introduce a new location-based query type, *approximate range nearest neighbor* (NN) query, that returns an answer set $\mathcal{A}$ such that each object in $\mathcal{A}$ is one of the k nearest object(s) to every point in a given query region. k is a user specified approximation tolerance level that can be used to tune a performance tradeoff between query response time and answer quality.
- We design an *incomplete pyramid structure* as an access method for efficiently retrieving a set of Voronoi cells that intersects a given query region from a Voronoi diagram for our proposed query processing algorithm.
- We propose an approximate range NN query processing algorithm that aims to minimize the number of objects returned in an answer set to improve query response time and user convenience while guaranteeing that the answer set satisfies the user specified approximation tolerance level.
- We provide experimental evidence through a comparison between the state-of-the-art techniques that our proposed query processing algorithm is scalable in terms of query processing time, and it significantly reduces query response time while the returned approximate answer is guaranteed to be satisfied with the user desired approximation tolerance level.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 describes our system model. Our proposed approximate range NN query processing algorithm is presented in Section 4. The extensive experimental results are analyzed in Section 5. Finally, Section 6 concludes the paper.

2 Related Works

In location-based services, *point* nearest-neighbor (NN) queries have been extensively studied, e.g., [1, 2, 3, 4, 5]. Existing *point* NN query processing algorithms mainly focus on the scalability and efficiency of finding the nearest object(s) to a given query point. By considering user mobility, the concept of NN searches is extended to line segments [6] (referred to as *linear* NN query). The basic idea of *linear* NN query processing algorithm is to split a line segment into subsegments such that each subsegment has the same nearest object(s). All such nearest objects constitute the answer set of a *linear* NN query. Recently, the concept of NN searches is further extended to rectangular regions [7, 9] (referred to as *range* NN query). A minimal answer set for a *range* NN query includes all objects located in query region and the nearest objects to each edge of the query region [7]. By relaxing the minimality requirement, another existing *range* NN query processing algorithm, *Casper*, computes a candidate answer set that includes the exact



(a) 1-order Voronoi diagram (b) 2-order Voronoi diagram

Fig. 2. The 1-order and 2-order Voronoi diagrams for five sites s_1 to s_5

answer [9]. The *Casper* algorithm first finds the nearest object to each vertex of the query region as *filters*, and then extends each edge of the query region to a minimal distance that is computed based on the *filters* to form an extended query region. The candidate answer set is a set of objects that is included in the extended query region. Also, a *range* NN query processing algorithm is proposed for finding the minimal answer set for a circular query region [8].

Our proposed approximate range NN query processing algorithm distinguishes itself from all previous techniques, as (1) it allows the user to specify an approximation tolerance level k for a range NN query, so that each object in the answer set is one of the k nearest objects to every point in a given query region; and (2) it aims to minimize the number of objects returned in the answer set to improve query response time, as the response time is dominated by the transmission time of sending the answer set to the user. In Section 5, we will compare the performance of our proposed algorithm with the state-of-the-art *range* NN query processing algorithms (i.e., [7, 9]).

3 System Model

In this section, we first formally define our problem, and then present the basic concept of Voronoi diagrams and the underlying system architecture.

Problem definition. Our problem is defined as follows: *given a set of objects, a range nearest-neighbor query Q with a query region $Q.R$, and an approximation tolerance level k, find the minimal set of objects $\mathcal{A}$ such that each object in $\mathcal{A}$ is one of the k nearest objects to every point in $Q.R$.*

Voronoi diagrams. Given a set of points $\mathcal{S}$ on the plane, which are the Voronoi *sites*, the Voronoi diagram of $\mathcal{S}$, denoted as $V(\mathcal{S})$, is a decomposition of the space into disjoint regions, *cells*, such that each site s_i is associated with a cell V_j, denoted as $V_j = \{s_i\}$, containing all the points in the plane that are closer to s_i than any other site in $\mathcal{S}$. In other words, s_i is the nearest site to every point in V_j. Figure 2a depicts a Voronoi diagram of a set of five sites

$\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5\}$, $V(\mathcal{S})$. $V(\mathcal{S})$ decomposes the space into five cells V_1, V_2, V_3, V_4, and V_5 that are associated with the sites s_1, s_2, s_3, s_4, and s_5, respectively. For example, given a point p in cell V_1, s_1 is the nearest site to p.

Higher-order Voronoi diagrams. The k-order Voronoi diagram extends the concept of the Voronoi diagram by defining cells based on the k nearest neighbors. The k-order Voronoi diagram of $\mathcal{S}$, where $1 < k \leq |\mathcal{S}| - 1$, denoted as $V_k(\mathcal{S})$, is a decomposition of the space into disjoint cells, such that a distinct set of k sites $S_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ is associated with a cell V_j, $V_j = \{S_i\}$, containing all the points in the plane that have the sites in S_i as their k nearest sites. In other words, S_i contains k nearest sites to every point in V_j. Figure 2b depicts the 2-order Voronoi diagram of $\mathcal{S}$, $V_2(\mathcal{S})$, that decomposes the space into seven cells, i.e., $V_1 = \{s_1, s_3\}$, $V_2 = \{s_1, s_2\}$, $V_3 = \{s_2, s_3\}$, $V_4 = \{s_1, s_4\}$, $V_5 = \{s_3, s_4\}$, $V_6 = \{s_4, s_5\}$, and $V_7 = \{s_3, s_5\}$. For example, given a point p in cell V_3, the sites s_2 and s_3 are the two nearest sites to p.

System architecture. We consider a mobile environment where mobile users communicate with a location-based database server through a (2G/3G) cellular network. The data/control flow of our system is as follows: The mobile user sends range NN queries to the database server. Our proposed approximate range NN query processing algorithm that is implemented in the database server computes an answer set, and then the server sends the answer set to the user. We use the Euclidean distance as our distance metric.

4 Approximate Range NN Query Processing

In this section, we first describe an *off-line* process to compute Voronoi diagrams, from order one to order k_{max}, where k_{max} is the maximum allowable user specified approximation tolerance level, and present our proposed *incomplete pyramid structure* that is used as an access method for each Voronoi diagram. Then, we present an *on-line* query processing algorithm for approximate range NN queries.

4.1 Building Voronoi Diagrams

We use an off-line process to build k Voronoi diagrams for a set of objects, e.g., restaurants, hotels and gas stations, from order one to order k_{max}, where k_{max} is the maximum user specified approximation tolerance level. Thus, the user can specify her desired approximation tolerance level k from one to k_{max}. Notice that if $k = 1$, our algorithm provides an exact answer set of the minimal size for range NN queries. Given a set of objects $\mathcal{S}$, building a set of Voronoi diagrams, from order one to order k_{max}, i.e., $V_1(\mathcal{S}), V_2(\mathcal{S}), \ldots,$ and $V_{k_{max}}(\mathcal{S})$, takes $O(k_{max}^2 N \log N)$ time and $O(\sum_{k=1}^{k_{max}} k^2(N-k))$ space, where N is the number of objects in $\mathcal{S}$ [20]. After we build the k Voronoi diagrams, they are stored for later use in our proposed range NN query processing algorithm. For each Voronoi diagram, we maintain a table to store each Voronoi cell with its associated objects.

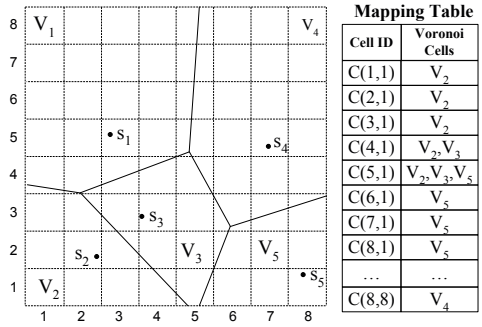


Fig. 3. The base level of an incomplete pyramid structure

4.2 Access Method for Voronoi Diagrams

We will construct an *incomplete pyramid structure* for each Voronoi diagram to support efficient range search among Voronoi cells. The idea of our proposed *incomplete pyramid structure* is that we need to find a set of Voronoi cells $\mathcal{V}$ that intersects a given range query region in order to retrieve their associated objects. Without any index structure, we have to scan every cell in a Voronoi diagram to find $\mathcal{V}$. When the number of objects and/or k is large, scanning all cells in a Voronoi diagram would pose a scalability issue. To this end, we propose an *incomplete pyramid structure* to overcome this issue. The construction of an *incomplete pyramid structure* for each Voronoi diagram includes two main steps.

STEP 1: Base level step. This step decomposes the space into grid cells where each grid cell $C(c, r)$ is uniquely identified by its column number c and row number r. Also, we use a hash table, *mapping table*, that associates each grid cell identity with a list of Voronoi cells that intersects the grid cell. Figure 3 depicts an 8×8 grid structure for the Voronoi diagram given in Figure 2a and the corresponding *mapping table*. For example, given a grid cell identity $C(5, 1)$, we can retrieve a set of Voronoi cells that intersects $C(5, 1)$, i.e., V_2, V_3, and V_5.

STEP 2: Merge step. This step merges quadtree-like neighbor cells to their parent if they intersect the same set of Voronoi cells. The idea of this step is to adaptively determine the height of an *incomplete pyramid structure* for a Voronoi diagram to minimize search time. This is due to the fact that the k Voronoi diagrams we maintain have different structures, e.g., the number of Voronoi cells and Voronoi cell size distribution, with respect to the number of objects, the object distribution and the degree of order (k). Thus, the shape of an *incomplete pyramid structure* would be different for each computed Voronoi diagram. Other than the base level, each cell $C(l, c, r)$ at upper levels is identified by the level of the *incomplete pyramid structure* l, column number c and row number r. This step uses a bottom-up approach to construct the upper levels of an *incomplete pyramid structure*. Starting from the base level, if all quadtree-like sibling cells (i.e., the cells have the same parent) intersect the same set of Voronoi cells, they are merged to their parent. The merge process includes three tasks, (i) adding an entry that associates the parent cell identity with the set of intersected Voronoi cells of its children to the *mapping table*, (2) removing the

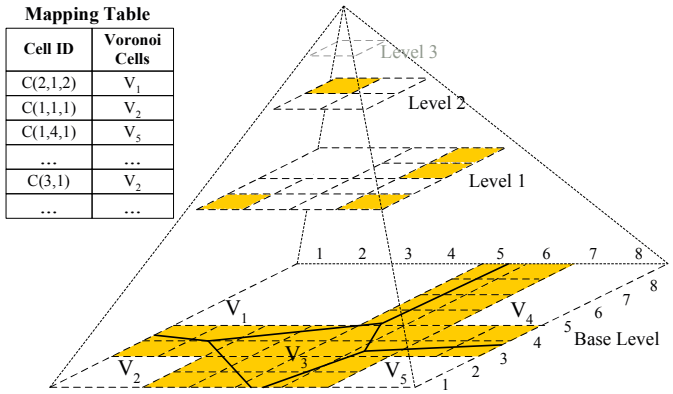


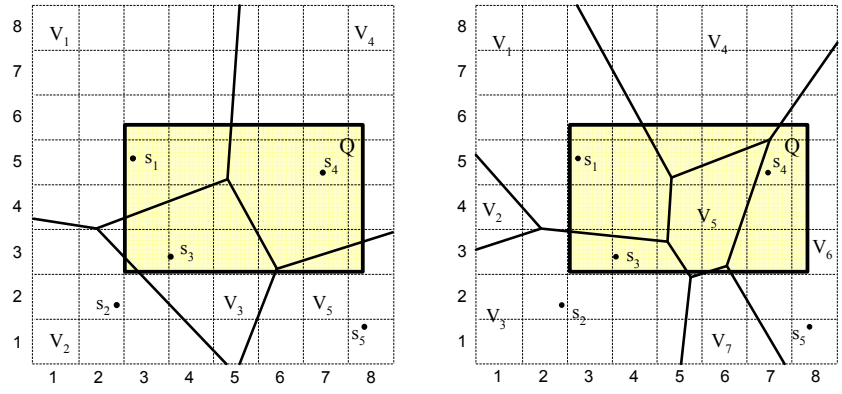
Fig. 4. Incomplete pyramid structure

entries of the merged child cells from the *mapping table*, and (3) annihilating the merged child cells by removing their pointers at their parent.

Figure 4 depicts an *incomplete pyramid structure* with a *mapping table* for the base level given in Figure 3, where the underlying Voronoi diagram is shown at the base level for the sake of illustration. Starting from the base level, we merge the quadtree-like sibling cells to their parent if they intersect the same set of Voronoi cells. For example, the four cells at the left bottom corner ($C(1,1)$, $C(2,1)$, $C(1,2)$, and $C(2,2)$) intersect the same Voronoi cell V_2, these cells are merged to their parent. To complete the merge process, we add an entry with the parent identity $C(1,1,1)$ with the intersected Voronoi cell V_2 to the *mapping table*, remove the entries of the merged child cells from the *mapping table*, and annihilate the merged child cells. We illustrate merged child cells by removing the grid cells. Similarly, we merge the cells at the other corners. At level one, the sibling cells at the left top corner (i.e., $C(1,1,3)$, $C(1,2,3)$, $C(1,1,4)$, and $C(1,2,4)$) intersect the same Voronoi cell V_1, so they are merged to their parent at level two, i.e., $C(2,1,2)$. At level two, since all cells intersect different sets of Voronoi cells, we cannot merge any cells and this step terminates. In this example, the shaded cells depict the lowest maintained cells of the *incomplete pyramid structure*, and there is an entry for each shaded cell in the *mapping table*. The height of the *incomplete pyramid structure* is two.

4.3 Online Query Processing Algorithm

The distinct feature of our proposed approximate range NN query processing algorithm is to enable the user to specify an approximation tolerance k for a range NN query, so that we provide a minimal answer set $\mathcal{A}$ where each object is guaranteed to be one of the k nearest objects to every point in the query region. If $k = 1$, we return an exact answer set with the maximal size to the user. In the case that $k > 1$, we return an approximate answer set with a smaller size than the exact one to the user; and thus, the transmission time of the answer set is reduced. The input of our proposed algorithm is a range NN query Q, a user specified approximation tolerance level k, and a k-order Voronoi diagram that



(a) Voronoi diagram (b) 2-order Voronoi diagram

Fig. 5. Range NN query processing using Voronoi diagrams

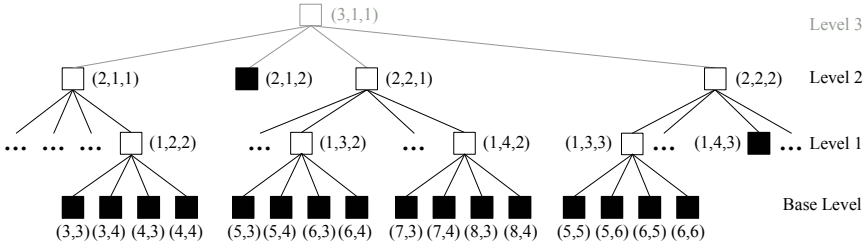


Fig. 6. Example of a range search in the incomplete pyramid structure (Figure 4)

is pre-computed by an off-line process (as described in Section 4.1). Figure 5 depicts a running example for our proposed algorithm where the query region of the input range NN query Q is represented as a bold rectangle and the maximum approximation tolerance level k_{max} is two. The algorithm consists of two key steps, *range search* step (Section 4.3.1) and *query-covering* step (Section 4.3.2).

4.3.1 Range Search Step

In this step, we retrieve a set of Voronoi cells $\mathcal{V}_k = \{V_1, V_2, \ldots, V_n\}$ that intersects the query region $Q.R$ and each Voronoi cell in $\mathcal{V}_k$ associates with k objects, i.e., $V_i = \{s_{i_1}, \ldots, s_{i_k}\}$ $(1 \leq i \leq n)$. We use a top-down approach to traverse an *incomplete pyramid structure*. Initially, at the highest maintained level of the *incomplete pyramid structure* of the k-order Voronoi diagram, we find a set of grid cells that intersects $Q.R$, and then recursively search each of these grid cells. During a recursive search, if an encountered grid cell C that intersects $Q.R$ is at the lowest maintained level or base level of the *incomplete pyramid structure*, we retrieve the set of Voronoi cells that intersects C from the *mapping table*, and then return it. Otherwise, we recursively search the four child cells of C.

Figure 6 illustrates the *range search* step for the running example depicted in Figure 5a where $k = 1$. For the sake of illustration, we only show the grid cells intersecting the query region $Q.R$, and the grid cells at the lowest maintained

Algorithm 1 Query-Covering Step

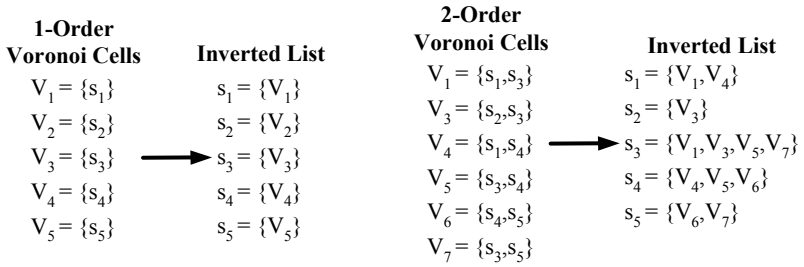
```
1: function QueryCovering (RNNQuery Q, ToleranceLevel k, VoronoiCellSet V_k)
2: AnswerSet A ← {∅}
3: Construct an inverted list of V_k, i.e., L(V_k) = {L(s_1), L(s_2), ..., L(s_m)}, where L(s_i) =
   {V_{i_1}, V_{i_2}, ..., V_{i_|L(s_i)|}} and 1 ≤ i ≤ m
4: while L(V_k) ≠ {∅} do
5:    Select the object s_i ∈ L(V_k) with the largest |L(s_i)|
6:    for each object s_j ∈ L(V_k) (i ≠ j) do
7:       L(s_j) ← L(s_j) − L(s_i)
8:       if L(s_j) = {∅} then
9:          L(V_k) ← L(V_k) − {L(s_j)}
10:      end if
11:   end for
12:   A ← A ∪ {s_i}
13:   L(V_k) ← L(V_k) − {L(s_i)}
14: end while
15: return A
```

level or base level of the *incomplete pyramid structure* (as depicted in Figure 4) are represented shaded nodes. As shown in Figure 4, the highest maintained level of the *incomplete pyramid structure* is level two where we start the *range search* step. Since all grid cells at level two, i.e., $C(2,1,1)$, $C(2,1,2)$, $C(2,2,1)$ and $C(2,2,2)$, intersect $Q.R$, we will recursively search these grid cells. For $C(2,1,1)$, only one child cell $C(1,2,2)$ intersects $Q.R$, so we search their child cells $C(3,3)$, $C(3,4)$, $C(4,3)$, and $C(4,4)$ at the base level. Since all these child cells intersect $Q.R$, we retrieve the Voronoi cells that intersect them from the *mapping table* and return the set of retrieved Voronoi cells. As $C(2,1,2)$ is at the lowest maintained level, we retrieve the Voronoi cells that intersects $C(2,1,2)$ from the *mapping table* without further search. Then, we search the grid cell $C(2,2,1)$ where it has two child cells $C(1,3,2)$ and $C(1,4,2)$ intersecting $Q.R$. Since all child cells of $C(1,3,2)$ and $C(1,4,2)$ at the base level intersect $Q.R$, we retrieve the Voronoi cells that intersect them from the *mapping table* and return the retrieved Voronoi cells. Similarly, we search the grid cell $C(2,2,2)$. As a result, the set of Voronoi cells that intersects $Q.R$ is $\mathcal{V}_1 = \{V_1, V_2, V_3, V_4, V_5\}$. For the other running example where $k = 2$, the *range search* step searches the *incomplete pyramid structure* of the 2-order Voronoi diagram and the set of Voronoi cells that intersects $Q.R$ is $\mathcal{V}_2 = \{V_1, V_3, V_4, V_5, V_6, V_7\}$.

4.3.2 Query-Covering Step

Algorithm 1 gives the pseudo code of this step where we aim to compute the minimal set of objects in which each object is one of the k nearest objects to every point in the query region $Q.R$. First, we construct an inverted list of the set of Voronoi cells $\mathcal{V}_k$ retrieved from the previous step, $\mathcal{L}(\mathcal{V}_k)$ (Line 3). In the inverted list $\mathcal{L}(\mathcal{V}_k)$, each object s_i has a list of Voronoi cells $L(s_i) = \{V_{i_1}, \ldots, V_{i_m}\}$ $(m \leq n)$, where s_i is associated with V_{i_j} $(1 \leq j \leq m)$.

Figure 7a depicts the inverted list of our running example for $k = 1$, as given in Figure 5a, where the Voronoi cells in $\mathcal{V}_1$ with their associated objects retrieved from the *range search* step are $V_1 = \{s_1\}$, $V_2 = \{s_2\}$, $V_3 = \{s_3\}$, $V_4 = \{s_4\}$,



(a) Voronoi cell set $\mathcal{V}_1$ (b) Voronoi cell set $\mathcal{V}_2$

Fig. 7. Inverted lists

and $V_5 = \{s_5\}$. The inverted list of $\mathcal{V}_1$ is $\mathcal{L}(\mathcal{V}_1) = \{L(s_1) = \{V_1\}, L(s_2) = \{V_2\}, L(s_3) = \{V_3\}, L(s_4) = \{V_4\}, L(s_5) = \{V_5\}\}$. On the other hand, Figure 7b depicts the inverted list of our running example for $k = 2$, as shown in Figure 5b where the Voronoi cells in $\mathcal{V}_2$ with their associated objects retrieved from the *range search* step are $V_1 = \{s_1, s_3\}$, $V_3 = \{s_2, s_3\}$, $V_4 = \{s_1, s_4\}$, $V_5 = \{s_3, s_4\}$, $V_6 = \{s_4, s_5\}$, and $V_7 = \{s_3, s_5\}$. The inverted list of $\mathcal{V}_2$ is $\mathcal{L}(\mathcal{V}_2) = \{L(s_1) = \{V_1, V_4\}$, $L(s_2) = \{V_3\}$, $L(s_3) = \{V_1, V_3, V_5, V_7\}$, $L(s_4) = \{V_4, V_5, V_6\}$, $L(s_5) = \{V_6, V_7\}\}$.

After constructing the inverted list of $\mathcal{V}_k$, $\mathcal{L}(\mathcal{V}_k)$, our objective is to select the minimal set of objects from $\mathcal{L}(\mathcal{V}_k)$ such that every Voronoi cell in $\mathcal{V}_k$ has at least one associated object selected in the answer set. In other words, we consider the items in the inverted list as sets and the Voronoi cells in $\mathcal{V}_k$ as elements, and then select a minimum number of sets so that the selected sets contain all the elements that are contained in any of the sets in the inverted list. Thus, our problem can be reduced to a well-known *set-covering problem*. Since computing the optimal solution for the set-covering problem is NP-hard [21], we use a greedy approach to compute an answer set. Basically, the greedy approach selects an object with the largest set of Voronoi cells, and then remove the Voronoi cells associated with the selected object from other objects' lists. Then, the selected object and the objects with an empty list are removed from the inverted list. We repeat this procedure until the inverted list is empty (Lines 4 to 14 in Algorithm 1). The set of selected objects is returned as the answer set to the user.

In our running example for $k = 1$, i.e., the user wants to have an exact query answer, the *query-covering* step simply add all objects in the inverted list $\mathcal{L}(\mathcal{V}_1)$ (Figure 7a) to the answer set, i.e., $\mathcal{A}_1 = \{s_1, s_2, s_3, s_4, s_5\}$. On the other hand, Figure 8 depicts the *query-covering* step for our running example for $k = 2$, based on the inverted list of $\mathcal{V}_2$, $\mathcal{L}(\mathcal{V}_2)$, as given in Figure 7b. Since $L(s_3)$ has the largest size, we select s_3 and remove the Voronoi cells in $L(s_3)$, i.e., V_1, V_3, V_5, and V_7, from other objects' lists, i.e., $L(s_1)$, $L(s_2)$, $L(s_4)$, and $L(s_5)$. Then, we add s_3 to an answer set $\mathcal{A}_2$ and remove $L(s_3)$ from $\mathcal{L}(\mathcal{V}_2)$. The updated inverted list is $\mathcal{L}(\mathcal{V}_2) = \{L(s_1) = \{V_4\}$, $L(s_2) = \{\emptyset\}$, $L(s_4) = \{V_4, V_6\}$, $L(s_5) = \{V_6\}\}$. After we remove the empty list $L(s_2)$ from $\mathcal{L}(\mathcal{V}_2)$, $L(s_4)$ has the largest size. Thus, we select s_4 and remove the Voronoi cells in $L(s_4)$, i.e., V_4 and V_6, from other

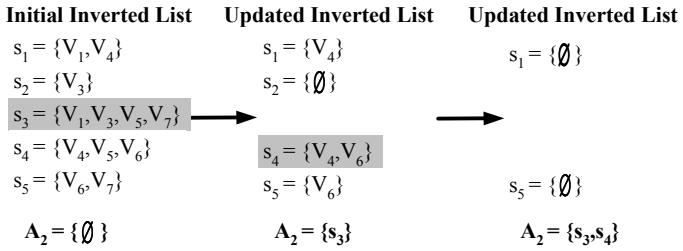


Fig. 8. Example of the query-covering step, based on Figure 7b

objects' lists, i.e., $L(s_1)$ and $L(s_5)$. Then, we add s_4 to $\mathcal{A}_2$ and remove $L(s_4)$ from $\mathcal{L}(\mathcal{V}_2)$. The updated inverted list is $\mathcal{L}(\mathcal{V}_2) = \{L(s_1) = \{\emptyset\}$ and $L(s_5) = \{\emptyset\}\}$. Since all lists are empty, they are removed from the inverted list, and the *query-covering* step terminates and returns the answer set $\mathcal{A}_2 = \{s_3, s_4\}$ to the user. From this example, we can see that our proposed approximate range NN query processing algorithm reduces the answer set size by 60%, i.e., from five objects in the exact answer set to two objects in the approximate answer set.

5 Experimental Results

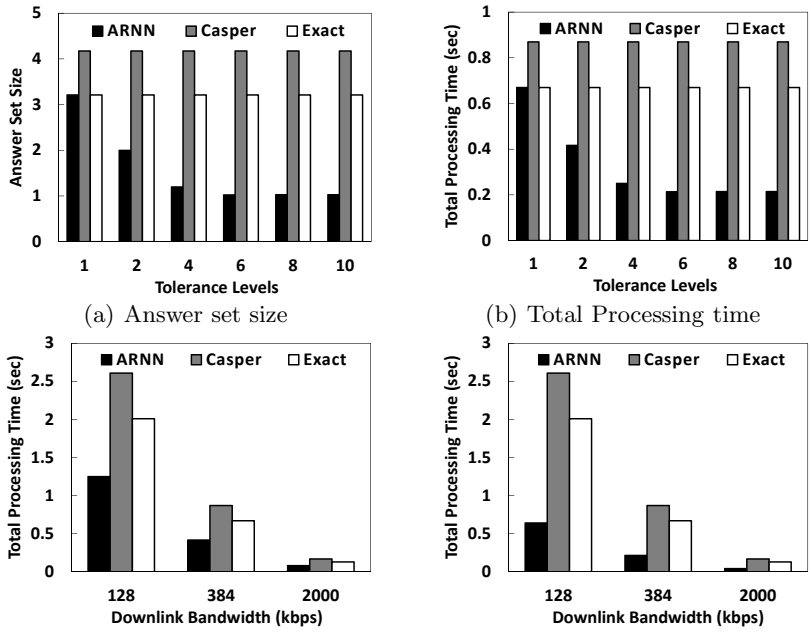
In this section, we evaluate our Approximate Range nearest-neighbor (NN) query processing algorithm (denoted as ARNN) with respect to user specified approximation tolerance levels (k), query region size, the number of objects, downlink bandwidth, and object size. We compare our ARNN algorithm with two state-of-the-art range NN query processing algorithms as baseline algorithms. The first baseline algorithm computes an exact answer set of the minimal size for range NN queries (denoted as Exact) [7], while the other baseline algorithm computes a candidate answer set that contains the exact answer for range NN queries (denoted as Casper) [9].

We have two performance measures: (1) *total processing time* that includes the query processing time at the database server and the transmission time of sending the answer set to the user, and (2) *answer set size* that is the average number of objects returned in the answer sets. The *answer set size* is important as it indicates communication overhead and the power consumed by the user device to receive the answer set and user convenience.

In all experiments, we assume that the user communicates with a database server through a 3G cellular network. The downlink (i.e., from the database server to the user) bandwidth varies with respect to the user mobility speed, i.e., 128 kbps (i.e., kbits per second) at vehicular speeds, 384 kbps at pedestrian speeds, 2 Mbps at stationary or very slow movement speeds. Unless mentioned otherwise, the experiments consider 200 objects in a square space of a length $l = 1000$. The mobile user moving at pedestrian speeds (i.e., the downlink bandwidth is 384 kbps) issues 1,000 range NN queries, and the object size is 10 Kbytes. The default user specified approximation tolerance level (k) is four and the query region size is $0.05l \times 0.05l$. Table 1 summarizes the parameter settings.

Table 1. Parameter settings

Parameter	Default Value	Range
Approximation tolerance (k)	4	1 to 10
Number of objects	200	100 to 300
Query region size	$(0.05l)^2$	$(0.008l)^2$ to $(0.256l)^2$ (where $l = 1000$)
Downlink bandwidth	384 kbps	128 kbps to 2 Mbps
Object size	10 Kbytes	0.5 Kbytes to 20 Kbytes



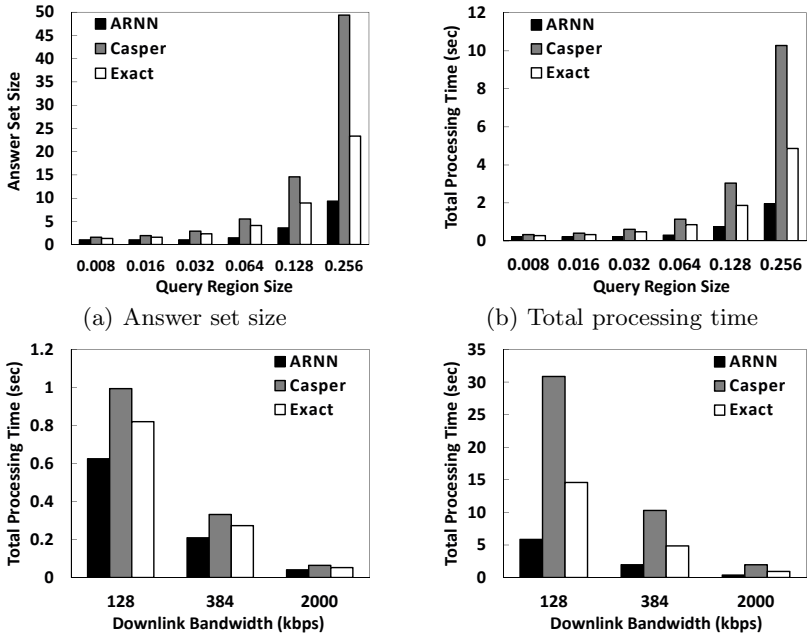
(a) Answer set size (b) Total Processing time

(c) Tolerance level: $k = 2$ (d) Tolerance level: $k = 10$

Fig. 9. Approximation tolerance levels (k)

5.1 Effect of Approximation Tolerance Levels

Figure 9 depicts the performance of our proposed algorithm (ARNN) with respect to varying the approximation tolerance level (k) from 1 to 10. The performance of the baseline algorithms (Casper and Exact) is not affected by varying the value of k. Figure 9a gives the number of objects returned in the answer set, while Figure 9b indicates the total processing time that includes the query processing time at the database server and the transmission time of sending the answer set to the user. Figure 9a shows that ARNN effectively reduces the answer set size as k gets larger. When $k = 2$ ($k = 10$), ARNN reduces the size of the answer sets given by Casper and Exact by 34.3% and 66% (79.3% and 89.3%), respectively. Since the transmission time is much higher than the total processing time, the total processing time of ARNN decreases as k gets larger (Figure 9b). Figures 9c and 9d show that ARNN performs better than the baseline algorithms for all mobility speeds. Since ARNN effectively reduces the answer set size, when the



(a) Answer set size (b) Total processing time

(c) Query region size: $(0.008l)^2$ (d) Query region size: $(0.256l)^2$

Fig. 10. Query region size

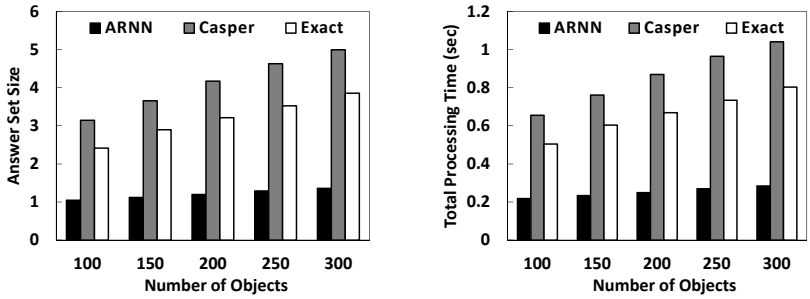
downlink bandwidth is more limited, ARNN performs much better than Casper and Exact.

5.2 Effect of Query Region Size

Figure 10 depicts the performance of our proposed algorithm (ARNN) with respect to increasing the query region size from $(0.008l)^2$ to $(0.256l)^2$, where $l = 1000$. Figure 10a shows that the answer set of all algorithms gets larger as the query region size increases. With small query regions, i.e., $(0.008l)^2$, the answer set size of ARNN is 94.7% and 59.8% smaller than Casper and Exact, respectively. For large query regions, i.e., $(0.256l)^2$, the answer set size of ARNN is 442.1% and 145.3% smaller than Casper and Exact, respectively. Thus, ARNN performs much better than the baseline algorithms for larger query regions. Since the transmission time is much higher than the total processing time, ARNN outperforms the baseline algorithms in terms of query response time (Figure 10b). Figures 10c and 10d depict that ARNN effectively reduces the total processing time of the baseline algorithms regardless of user mobility speeds.

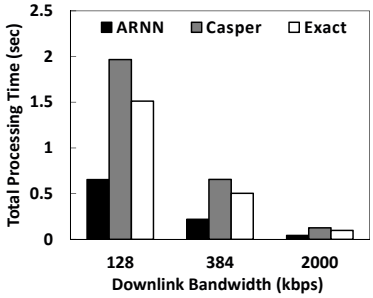
5.3 Effect of Number of Objects

Figure 11 gives the performance of our proposed algorithm (ARNN) with respect to varying the number of objects from 100 to 300. When the number of objects increases, there are more nearest objects to the query region; and thus, the

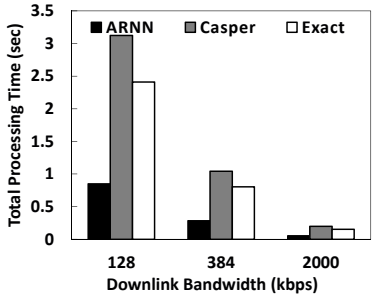


(a) Answer set size

(b) Total processing time

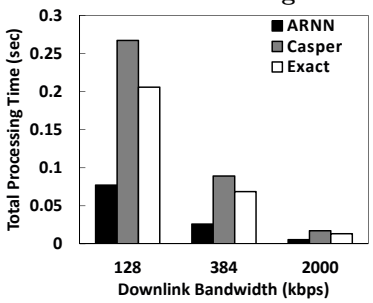


(c) 100 objects

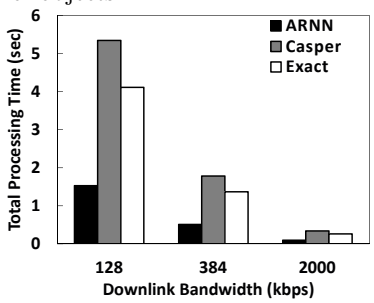


(d) 300 objects

Fig. 11. Number of objects



(a) 0.5 KB



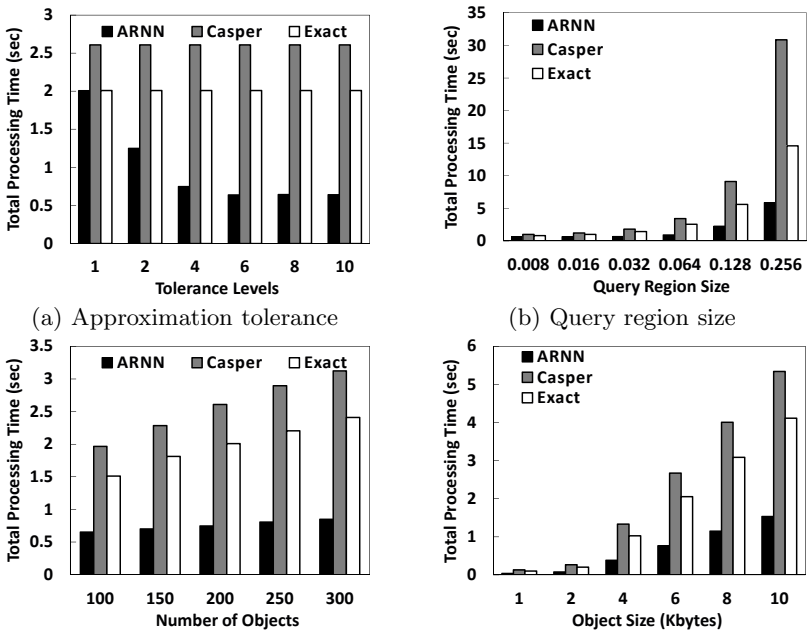
(b) 20 KB

Fig. 12. Object size

answer set size of all algorithms gets larger (Figures 11a). Similar to the previous experiments, the transmission time is much higher than the total processing time. Since the answer set size of ARNN is smaller than the baseline algorithms Casper and Exact, ARNN incurs the lowest total processing time for any number of objects (Figures 11b). Likewise, ARNN effectively reduces the answer set size, the total processing time of ARNN is better than Casper and Exact for all user mobility speeds, as depicted in Figures 11c and 11d.

5.4 Effect of Object Size

Figure 12 depicts the performance of our proposed algorithm (ARNN) with respect to the object size of 0.5 and 20 Kbytes. Since varying the object size does



(a) Approximation tolerance (b) Query region size

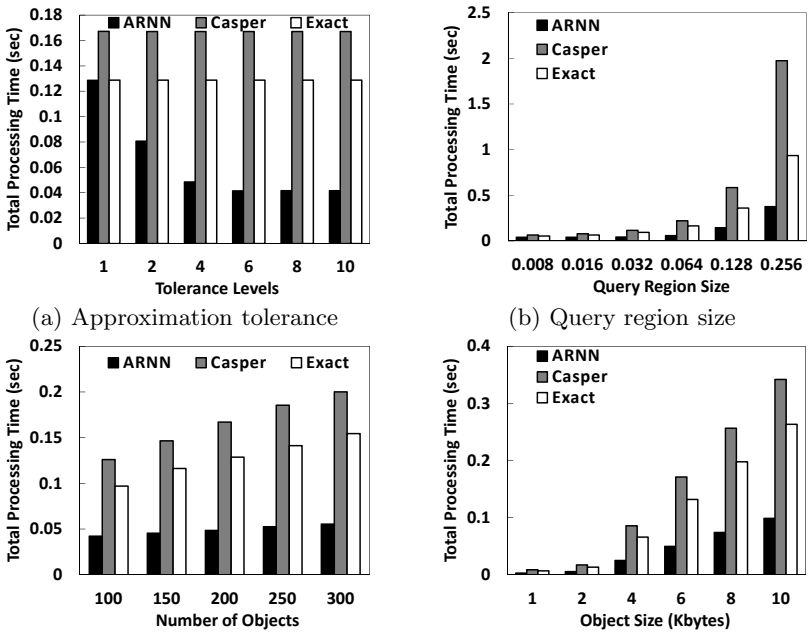
(c) Number of objects (d) Object size

Fig. 13. Downlink bandwidth at vehicular speeds (128 kbps)

not affect the answer set size, the answer set size of all algorithms is the same as the case that $k = 4$ in Figure 9a. It is interesting to see that the transmission time is much higher than the total processing time even if the object size is small and the answer set is sent to the user through the downlink with the largest possible bandwidth, i.e., 2 Mbps. Therefore, the results indicate that reducing the answer set size is an effective way to improve query response time. This is the motivation of our proposed algorithm ARNN that aims to minimize the answer set size while guaranteeing that the answer set is satisfied with the user specified approximation tolerance level k.

5.5 Effect of Communication Bandwidth

Figures 13 and 14 give the comprehensive evaluation of our proposed algorithm (ARNN) with respect to all parameters for users moving at vehicular speeds and very slow speeds, respectively. Although Casper gives the best query processing time, it suffers from very high transmission time. This is because the candidate answer set provided by Casper is much larger than the answer set of ARNN and Exact. Thus, the total processing time of Casper is always worse than ARNN and Exact. Since ARNN provides approximate answers that satisfy the user specified approximation tolerance level, the answer set size of ARNN is smaller than the exact answer set provided by Exact. As a result, ARNN performs better than Casper and Exact in terms of the total processing time for all parameter settings.



(a) Approximation tolerance (b) Query region size

(c) Number of objects (d) Object size

Fig. 14. Downlink bandwidth at stationary or very slow speeds (2 Mbps)

6 Conclusion

In this paper, we propose a new query type, *approximate range nearest-neighbor (NN) query*, for location-based services. The distinct features of this new query type are that (1) It aims to minimize the number of objects returned to the user so as to reduce the transmission time of sending the answer to the user; and (2) It provides quality guarantee for the query answer, i.e., each object in an answer set is one of the k nearest objects to every point in a given query region, where k is a user specified tuning parameter for a tradeoff between query response time (that is dominated by transmission time as shown in all experimental results) and the quality of answers. To achieve these two features, we propose an *approximate range NN query processing* algorithm. The main idea is to have an *off-line* process to compute Voronoi diagrams, from order one to order k_{max}, where k_{max} is the maximum allowable user specified approximation tolerance level, and then build our proposed *incomplete pyramid structure* as an access method for each Voronoi diagram. Given a range NN query and an approximation tolerance level k, our *on-line* query processing algorithm accesses the *incomplete pyramid structure* of the k-order Voronoi diagram to retrieve a set of Voronoi cells that intersects the query region and the k nearest objects to each Voronoi cell. Then, the remaining query processing is reduced to a set-covering problem where we use a greedy approach to find a minimal answer set. Extensive experimental results show that our proposed algorithm is scalable in terms of query processing time, and effective to reduce query response time com-

pared with the state-of-the-art techniques while guaranteeing that the answer set satisfies the user desired approximation tolerance level.

References

[1] Benetis, R., Jensen, C.S., Karciauskas, G., Saltenis, S.: Nearest and reverse nearest neighbor queries for moving objects. VLDB Journal **15**(3) (2006) 229–249

[2] Hu, H., Xu, J., Lee, D.L.: A generic framework for monitoring continuous spatial queries over moving objects. In: SIGMOD. (2005)

[3] Mouratidis, K., Papadias, D., Hadjieleftheriou, M.: Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In: SIGMOD. (2005)

[4] Mokbel, M.F., Xiong, X., Aref, W.G.: Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In: SIGMOD. (2004)

[5] Zheng, B., Xu, J., Lee, W.C., Lee, D.L.: Grid-partition index: A hybrid method for nearest-neighbor queries in wireless location-based services. VLDB Journal **15**(1) (2006) 21–39

[6] Tao, Y., Papadias, D., Shen, Q.: Continuous nearest neighbor search. In: VLDB. (2002)

[7] Hu, H., Lee, D.L.: Range nearest-neighbor query. IEEE TKDE **18**(1) (2006) 78–91

[8] Kalnis, P., Ghinita, G., Mouratidis, K., Papadias, D.: Preventing location-based identity inference in anonymous spatial queries. IEEE TKDE **19**(12) (2007) 1719–1733

[9] Mokbel, M.F., Chow, C.Y., Aref, W.G.: The new casper: Query processing for location services without compromising privacy. In: VLDB. (2006)

[10] de Almeida, V.T., Güting, R.H.: Supporting uncertainty in moving objects in network databases. In: ACM GIS. (2005)

[11] Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Querying imprecise data in moving object environments. IEEE TKDE **16**(9) (2004) 1112–1127

[12] Pfoser, D., Jensen, C.S.: Capturing the uncertainty of moving-object representations. In: SSD. (1999)

[13] Trajcevski, G., Wolfson, O., Hinrichs, K., Chamberlain, S.: Managing uncertainty in moving objects databases. ACM TODS **29**(3) (2004) 463–507

[14] Yiu, M.L., Mamoulis, N., Dai, X., Tao, Y., Vaitis, M.: Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. IEEE TKDE **21**(1) (2009) 108–122

[15] Bamba, B., Liu, L., Pesti, P., Wang, T.: Supporting anonymous location queries in mobile environments with privacygrid. In: WWW. (2008)

[16] Cheng, R., Zhang, Y., Bertino, E., Prabhakar, S.: Preserving user location privacy in mobile data management infrastructures. In: PET. (2006)

[17] Ghinita1, G., Kalnis, P., Skiadopoulos, S.: Mobihide : A mobile peer-to-peer system for anonymous location-based queries. In: SSTD. (2007)

[18] Gedik, B., Liu, L.: Protecting location privacy with personalized k-anonymity: Architecture and algorithms. IEEE Trans. on Mobile Computing **7**(1) (2008) 1–18

[19] Hu, H., Xu, J.: Non-exposure location anonymity. In: ICDE. (2009)

[20] Lee, D.T.: On k-nearest neighbor voronoi diagrams in the plane. IEEE Trans. on Computers **31**(6) (1982) 478–487

[21] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd Edition. MIT Press, Cambridge, MA (2001)