A kNN Search Protocol Using a Voronoi Diagram in Wireless Broadcast Environments

Wei-Chi Yeh, Chuan-Ming Liu, and Kai-Yun Ho
Department of Computer Science and Information Engineering
National Taipei University of Technology
Taipei 106, TAIWAN
{t7598035,cmliu,s4618026}@ntut.edu.tw

Abstract

Data broadcasting is an effective way to disseminate information to a large amount of mobile clients in wireless mobile environments. The k nearest neighbor (kNN) search is one of the important location-based services. In this paper, we propose kNN search protocols using data broadcasting. We consider how the server generates the broadcast schedule with a Voronoi diagram and how the client can efficiently execute the query process in terms of latency (time interval from issuing and terminating a query) and tuning time (time spent on listening to the broadcast). Our proposed kNN search protocols will not use an index. Instead, the proposed protocols use some additional information for each broadcast data and consider the locality of data points. The effectiveness of the proposed protocols will be verified by proofs and a simulation.

I. Introduction

Emerging technologies on wireless communication, positioning systems, and networking make it possible for people to access information ubiquitously. In such a wireless mobile environment, the bandwidth of the downlink is much larger than the bandwidth of the uplink [7]. Under such an asymmetric environment, the traditional one-to-one client/server communication is not an appropriate method to provide the information or services because of the bottleneck of the uplink. Instead, data broadcasting provides an effective way to disseminate information to a large pool of mobile clients.

In different information services, the location-based service (LBS) is one of the important services and provides the information related to the positions of the mobile clients. The k nearest neighbors (kNN) search is one of the query types in the location-based services. A mobile client can have the query like "where is the nearest gas station?" or " please give me five nearest book stores". In this paper, we consider how to provide the kNN search using data broadcasting in wireless mobile environments [4], [10], [16].

When applying data broadcasting to provide information services, two cost measures are usually considered. The *latency* is the time elapsed between issuing and termination of the query and can be used as one of the parameters for the Quality of Service (QoS) provided by the system. The *tuning time* is the amount of time spent on listening to the channel and represents the power consumption of mobile clients. In general, when the broadcast consists of only data without an index, these two cost measures are equivalent. The papers about data broadcasting without an index focus on how to schedule the broadcast in order to achieve a shorter latency [1], [7], [11], [13]. By adding an index in the broadcast can reduce the tuning time; therefore, leading a less power consumption [4], [5], [6], [7], [10], [16]. A *broadcast cycle* is one instance of all the broadcast data. When designing the data broadcasting protocols, two aspects are generally considered. From the perspective of a server, how to organize the data in the broadcast in order that the clients can access the information efficiently in terms of the latency and tuning time is challenging. From the viewpoint of a mobile client, how to receive the information in an energy-efficient manner (i.e., with smaller tuning time) raises an important issue since the energy is a scarce resource for a mobile client.

Our proposed protocol for the kNN search will consider how the server schedules the broadcast and how the client efficiently executes the query processing on the corresponding broadcast. In our design, we will not include an index on the data set. This differentiate our work from [4], [10] Instead, we add some additional information with each data point to allow selectively tuning in order to reduce the tuning time (energy consumption). And, we consider the locality of the data points in the broadcast schedule to further reduce the latency and tuning time. For simplicity, we assume that each data point corresponds to one packet

978-1-4244-5113-5/09/$25.00

in the broadcast. The additional information is derived from the Voronoi diagram composed by the data set. More details about the proposed protocols is given and analyzed in Section III. Besides, we provide the simulation results in Section IV. Section V concludes this paper. In the next section, we give the related works and preliminaries.

II. Preliminaries

The nearest neighbor (NN) search problem has been studied for many decades and many solutions have been proposed. One approach to solve the NN search problem for a given data set is to use an index structure, like R-trees, k-d-trees, or Quad-trees [14]. The first NN search algorithm on an R-tree [12] followed the branch-and-bound algorithmic design pattern and could be generalized to the kNN search ($k > 1$). This algorithm searches the R-tree in a depth-first fashion and prunes the irrelevant nodes during the traversal. Based on this algorithm, the authors in [4] and [10] proposed different approaches for the kNN search using broadcast R-trees in wireless broadcast environments.

The other approach to find the kNN for a given query point in a data set is to use the Voronoi diagram [2], [16]. A Voronoi diagram for a given data set can be constructed by computing the perpendicular bisectors of the line segments between the pairs of data points. Each data point is then associated with a *cell* in the Voronoi diagram [3]. The nearest data point of a query point q is the data point whose associated cell contains q. Hence, one can find the nearest neighbor by examining the cells in the Voronoi diagram. Figure II illustrates a Voronoi diagram for a data set with a given query point.

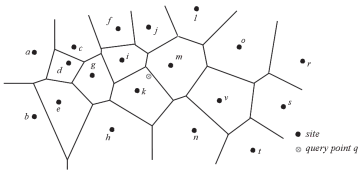


Fig. 1. A Voronoi diagram for a set of data points on the plane and a query point q.

Using the Voronoi diagram, one can find the NN for a query point but can not find the kNN for an arbitrary $k > 1$. To find the kNN, one can consider the order-k Voronoi diagram [9]. However, the value of k is fixed. In [8], the authors proposed an approach for kNN queries in spatial network databases using order-1 Voronoi diagram. The search first precomputes the possible data points around each cell and then filter the considered cells and the neighboring cells until the result is done. In wireless broadcast environments, the authors in [16] discussed some efficient NN search protocols, including the solution-based and object-based approaches, and proposed a hybrid one. Their solutions are mainly for the NN search. Our proposed approach will use the order-1 Voronoi diagram and consider the neighbors of each site, when generating the broadcast schedule.

In the following of this section, we give the terminology and notations used in this paper. Given two data points p and q in a plane, we denote the distance between p and q as $dist(p, q)$ and

$$dist(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

Let $P = \{p_1, p_2, \cdots, p_n\}$ be the set of n distinct data points in the plane. These points are the *sites*. In the remainder of this paper, the *data points* or *sites* are used interchangeably. Each site $p_i \in P$, $i = 1, 2, \cdots, n$, has an area, *Voronoi cell* $v(p_i)$,associated with it. The Voronoi diagram of P is denoted by $VD(P)$. Figure II shows a Voronoi diagram for a data points on the plane and the 3NN of the query point q is $\{i, k, m\}$.

If two sites share the same boundary of the corresponding Voronoi cells, these two sites are *1-order neighbors* to each other. For a site p_i, if the associated Voronoi cell has m boundaries, then p_i has m 1-order neighbors, say sites $u_1, u_2, \cdots, u_m$. We define the set of the 1-order neighbors of p_i as

$$Adj(p_i) = \{u_1, u_2, \cdots, u_m\}.$$

For instance, in Figure II, the 1-order neighbors of d are $Adj(d) = \{a, c, e, g\}$.

III. Proposed kNN Search Protocols

A straightforward approach for the kNN search in wireless broadcast environment is to have the server broadcast all the data points sequentially and the clients filter all the data points by always tuning into the broadcast. However, this approach is too naive and less efficient in terms of latency and tuning time since all the data points should be examined. So, some papers use an index in the broadcast to improve the performance on the latency and tuning time. In this section, we present our kNN search protocol which does not use an index but add some additional information in each site in the broadcast. Using the proposed approach, the client can perform the selective tuning to save the energy and will terminate the

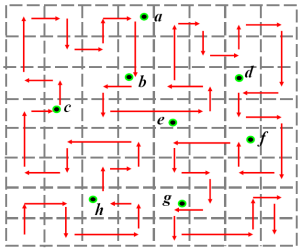
query processing when necessary, thus leading to a shorter latency. We now describe our approach in two parts: on the server side, we explore how to generate the broadcast; on the client side, we design an efficient query processing for the kNN search.

A. Broadcast Schedules on the Server

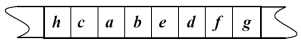
Our kNN search protocol for wireless broadcast environments is based on the NN search using a Voronoi diagram. For a given data set $P = \{p_1, p_2, \cdots, p_n\}$, the corresponding Voronoi diagram $VD(P)$ is first constructed. Note that a broadcast consists of a sequence of packets. For simplicity, the packet ID is the position (or address) of that packet in the broadcast. Each site in the constructed Voronoi diagram is assigned to one packet in the broadcast. Hence, each site is associated with a position which is the packet ID in the broadcast. Recall that a broadcast cycle is one instance of all the broadcast packets. The cycle length is n if we use one packet as the metric unit and we can number the packets from 0 to n-1.

Scheduling the broadcast is to assign all the sites to the packets in the broadcast in some order. Then each site has a packet ID associated. We consider four ways to schedule the broadcast. The first one is to simply assign the sites to the packet randomly and we call such a schedule as *R-schedule*. The R-schedule does not hold the locality of data points in the broadcast sequence and may lead to a longer average latency. In order to have a shorter latency, we can keep the locality of data points in the broadcast. We hence refer to some space filling curves, like Z-curves, G-curves, and Hilbert curves to generate the broadcast [15]. Due to the space limitation and Hilbert curve is one of the popular space-filling curves, we only shortly introduce the Hilbert curve. The readers can refer to [15] for the others. Hilbert curve which maps high-dimensional data into 1-dimensional data and passes all the data point once can keep the locality of data points. According to the Hilbert curve, the Hilbert order of each site can be calculated by the location of that site. We then use the Hilbert order as the packet ID in the broadcast. Hence, all the sites are broadcast according to the Hilbert order. We refer to such a schedule as *H-schedule*. Figure 2 demonstrates an H-schedule for a data points using Hilbert curves.

In order to improve the efficiency for searching, our broadcast schedule will add additional information in each packet. The additional information is derived from the Voronoi diagram $VD(P)$. For each site p, we include the information of p's 1-order neighbors, $Adj(p)$, into the corresponding packet. A



(a) Data set and a Hilbert curve



(b) The broadcast schedule

Fig. 2. An 8 point data set and a Hilbert curve on a $2^3 \times 2^3$ grid and the corresponding broadcast schedule by H-schedule

generic approach for generating the broadcast for a given data set P is shown in Figure 3. Different broadcast schedules are different in Step 2. The time

Broadcast_Schedule(P)
▷ $P = \{p_1, p_2, \cdots, p_n\}$ is the data set.
1 Create the Voronoi diagram of P, $VD(P)$;
2 Find the associated packet ID for each site p_i;
3 Place the sites to the broadcast packet according to the packet ID;
4 For each site p_i, add the additional information of p_i to the corresponding packet;

Fig. 3. A high-level description of the generic schedule for the broadcast on the server

complexity for generating the broadcast mainly depends on the time for generating the Voronoi diagram and the policy for the assignment for sites. So far, the Voronoi diagram can be constructed in $O(n \log n)$ for data set of size n. The time complexity of the assignment for sites in all the proposed broadcast schedules is $O(n)$ time. Step 3 and Step 4 are $O(n)$ respectively by a linear scan on each packet. So, all the proposed broadcast schedules can be generated in $O(n \log n)$ time.

B. Client Query Processing

With the broadcast schedule in the previous subsection, our query processing can start at an arbitrary time instance. There are two data structures used during the query execution. *D-list* is a sorted list, with the position of a site as the key, to store the positions, which the query processing can skip, in an increasing order. *C-list* is a sorted list with the distance to query point q as the key and stores the candidate sites in an non-decreasing order. The high-level description

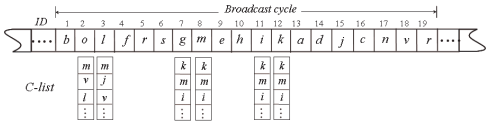


Fig. 4. The example of 3NN query process.

of the algorithm for the query processing is shown in Figure 5.

Initially, all the lists are empty and a packet is received from the broadcast. The query process will examine all the information contained in the packet. Suppose the site in the received packet is p. Then site p is marked to indicate that p has been examined and the distance between p and q is also computed. For each site u in $Adj(p)$, the set of 1-order neighbors of p, we compute the distance between u and q. All the sites, including p and the 1-order neighbors of p, are inserted into *C-list* and all the sites in *C-list* are sorted according to their distances to q in a non-decreasing order. During the insertion, if a site is already in *C-list*, this site is ignored. In *C-list*, we decide the first k sites as the candidates for the kNN of q. If there are more than k sites in *C-list*, we insert the positions of the rest sites into *D-list* and then delete the them from *C-list*. If all the k sites in *C-list* are marked, the query process stops and these k sites is the kNN of q. Otherwise, the process continues and the client needs to access the next packet. The next packet to be received depends on *D-list*. If the position of the next packet is in *D-list*, the client can be in sleep mode and skip the next packet.

```
kNN-Query-VD(q, k)
 1  count ← 0;
    ▷ count: count the number of marked sites.
 2  while count ≠ k
 3        do receive a packet which contains site p;
 4           mark site p;
 5           count++;
 6           for each u in Adj(p) ∪ {p}
 7                do compute dist(p, u)
 8           insert p and each u into C-list;
 9           decide the candidates in C-list;
10           if k < |C-list|
11              then for each of the rest sites, u
12                        do insert the position of u
                             into D-list;
13                           if u is marked
14                              then count - -;
15                           delete u from C-list
16           Use D-list to compute the next
             packet to receive
```

Fig. 5. A high-level description of the kNN query processing on the client.

Consider the broadcast schedule in Figure 4 and suppose that we have a 3NN query at the query point q. Assume that the query starts at the position of site o. After receiving o, we mark o and increase the counter for the number of marked nodes by one. The 1-order neighbors of o are m, l, r, s, and v. We then insert all these sites, including o, into *C-list*. According to their distances to q, the 3 candidate sites are m, v, and l. The other sites can be deleted but their positions should be recorded for selective tuning. In this case, the positions 5 and 6 can be skipped since sites r and s are impossible to be included in the final result. Then we check whether all the candidate sites are marked. Since no candidate site is marked at this time instance, the process continues to receive the next packet. The next packet can be determined by comparing the current position with the positions stored in *D-list*. Because position 3 isn't recorded in *D-list*, this position must be examined. When l (packet 3) is received, we mark l and insert sites j and f, into *C-list* only since r, o, and m have been inserted. Note that, the 1-neighbors of l are f, j, m, o and r. According to the distance to q, we now keep m, j, and v in *C-list* and store positions 4, 5, and 6 in *D-list*. Then the next packet to receive is the packet at position 7. The same process continues until the packet at 12 which contains site k. After receiving this site, we mark k and all the sites in *C-list* are marked and the process stops. The resulting 3NN is k, m, i. In this example, the process experiences 6 and 11 packets in tuning time and latency respectively.

C. Correctness

In following, we show the correctness of the proposed search protocol. Recall that the set of data points (sites) is $P = \{p_1, p_2, \cdots, p_n\}$ and the corresponding Voronoi diagram is $VD(P)$. We start with the following two Lemmas. Due to the space limitation, we skip the proofs of them here.

Lemma 1: Let $p_i \in P$ be the nearest neighbor of query point q. Then q is in Voronoi cell $v(p_i)$ and the next nearest neighbor of q is in $Adj(p_i)$.

Lemma 2: Suppose the kNN of a query point q is $\{p_1, p_2, \cdots, p_k\}$. The $(k+1)$th nearest neighbor of query point q must be in $\bigcup_{i=1}^{k} Adj(p_i)$.

With these two lemmas, we can derive the following theorem.

Theorem 3: Given a kNN query at point q. If all the k sites are marked in *C-list*, those k sites in *C-list* are the kNN of q.

Proof: Let the resulting kNN be $\{p_1, p_2, \cdots, p_k\}$ and p_k be the kth nearest neighbor of q. We suppose that the query process starts and stops at position x

and y, respectively. Let F be the set of the sites from position x to position y on the broadcast and B is the set of the rest sites from position $y+1$ to position z, where $z-x+1$ is the total number of packets in the broadcast cycle. Our proof consists of two parts:
(1) When the query processing stops, the kNN of q in set F must be the k sites in *C-list*; and
(2) For each site p_i in B, $dist(q,p_k) \leq dist(q,p_i)$.

For part (1), in our algorithm, a site is marked when it is received and the *C-list* stores the candidate sites according to their distances to q and arrange them in a non-decreasing order. During the execution of the algorithm, we delete the sites which are impossible in the resulting kNN since their distances to q are longer than the distance between the current kth nearest site and q. The algorithm thus will examine all the possible sites and ignore the sites which are impossible in the result. Hence, when the algorithm stops, the k sites kept in *C-list* is the kNN of q in F.

In order to make sure that no more sites in B should be examined, we claim part (2). We assume that $dist(q,p_k) \geq dist(q,p_i)$, $p_i \in B$, and argue this by contradiction. Then, p_k should be replaced by p_i in the resulting kNN=$\{p_1, p_2, \cdots, p_k\}$ of q. Let the 1-order neighbors of the $(k-1)$NN is $\{u_1, u_2, \cdots, u_m\}$ for some integer m. Since $p_i \in B$, p_i does not belong to the kNN of q in set F and $p_i \notin \{u_1, u_2, \cdots, u_m\}$ according to Lemma 2. We also know that $p_k \in \{u_1, u_2, \cdots, u_m\}$. Since $dist(q,p_i) \leq dist(q,p_k)$, there exists at least one site u_j in $\{u_1, u_2, \cdots, u_m\}$ with $dist(q,u_j) \leq dist(q,p_k)$, where $j \leq m$. Then, sites u_j and p_k are both in $\{u_1, u_2, \cdots, u_m\}$ and $dist(q,u_j) \leq dist(q,p_k)$. According to our algorithm, u_j should be examined. Then, consider two cases. If u_j comes earlier than p_k in the broadcast, p_k will never be in the result. This contradicts to the conclusion in part (1). On the other hand, if u_j comes later than p_k in the broadcast, the process will not stop at p_k. This also contradicts to our process which stops at p_k. Hence, no more sites in B should be examined further and the algorithm stops correctly. □

IV. Simulation Results

In this section, we present our simulation. To evaluate the proposed protocols, we measure the tuning time and latency. In the simulation, 100,000 data points are generated uniformly in the area of [0,1000] × [0,1000]. The query point is generated randomly in the given area and the value of k varies from 1 to 100. The considered broadcast schedules includes H-schedule(HS), Z-schedule(ZS), G-schedule(GS), and Y-schedule(YS), R-schedule(RS), which use-Hilbert curve, Z-curve, gray-code order, Y-axis order, and random assignment for assigning packet ID to sites. The data reported is the average of 100 different queries for each value of k on a given broadcast. We use one site (or packet) as a measure unit for simplicity. The packet is of size 1024 byte which can contain all the information of an individual site. So, the broadcast cycle length is 100,000 in number of packets.

A. Tuning Time

The tuning time indicates the cost spent on listening to the broadcast by clients and reflects the power consumption of the mobile clients for the kNN search in wireless broadcast environments. Our kNN query process starts at the arbitrary position of a broadcast. Recall that the proposed protocol uses *D-list* to record the positions, so the query processing can skip listening some packets. Hence, our protocols can avoid spending energy on receiving the irrelevant sites. The simulation result shown in Figure 6 indicates that the proposed query process can explore fewer data points than the naive approach; thus leading to a smaller tuning time. Note that the tuning time is on the y-axis and on the x-axis are the values of k from 1 to 100. Furthermore, all the four schedules using space filling curves result in a smaller tuning time than R-schedule because the locality of the data points can be kept and the query processing thus can converge to the result by exploring fewer data points.

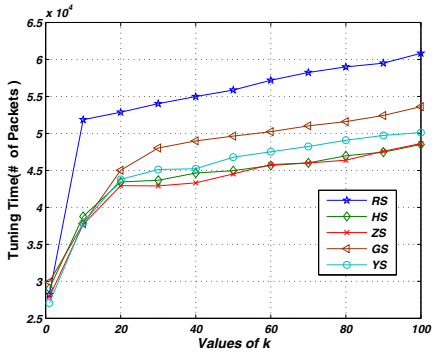


Fig. 6. The tuning time of the proposed kNN search protocols on the data set of size 100,000 with different values of k

B. Access Latency

In our protocol, when the client execute the query process, we use *C-list* to store the candidate sites. Using the candidate sites, we provide a condition to derive the final result and allow the process can stop

earlier than the naive kNN search process. Hence, our proposed protocol can achieve a shorter latency. Figure 7 demonstrates one of the simulation results about the latency. All the other simulation results have the same trend. Because the four schedules using space-filling curve retains the locality of the data points, they can lead to a shorter latency in average than RS. During the query processing, when a nearby site is approached, all the relevant sites will be checked soon and then the process will stop. Besides, the latency increases as the value of k increases. When k is small, the protocol will experience a shorter latency.

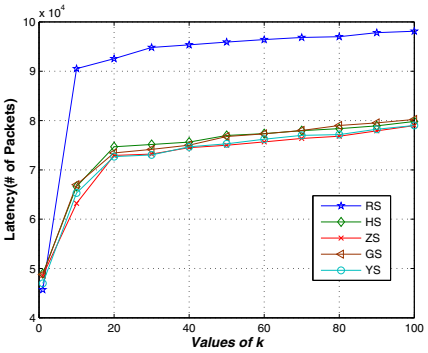


Fig. 7. The latency of the proposed kNN search protocols on the data set of size 100,000 with different values of k

V. Conclusions

In this paper, we present a kNN search protocol in wireless broadcast environments. The idea of the proposed protocols is simple but can reduce the latency and tuning time effectively. Unlike some other protocols, the protocol does not use an index in the broadcast and can support the kNN search for an arbitrary k. Our simulation results show the performance of our approaches in terms of tuning time and latency. In the future, we plan to have more simulation experiments on our protocols using different space-filling curves and compare it with the other existing kNN search protocols in wireless broadcast environments. We will also discuss further about the constraints on the distribution of the input data points on the plane. It is possible that a data point u in some given data point set of size n may have $n-1$ 1-order neighbors. This will hurt the proposed search protocols due to the limited packet size. However, this case is rare in general. Besides, we believe that the proposed approach can be easily adapted to the traditional environments to save the number of I/O operations when executing the query process. Recall that our approaches do not need an index structure. How to place the data points into the pages in order to make the search more efficient in terms of number of I/O operations becomes interesting.

References

[1] Swarup Acharya, Michael Franklin, and Stanley Zdonik. Balancing push and pull for data broadcast. In *Proceedings of the 1997 ACM SIGMOD international conference on management of data*, pages 183–194, 1997.

[2] S. Berchtold, D. A. Keim, H.-P. Kriegel, and T. Seidl. Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):45–57, 2000.

[3] M. de Berg, M. van Krefeld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2000.

[4] Bugra Gedik, Aameek Singh, and Ling Liu. Energy efficient exact knn search in wireless broadcast environments. In *Proceedings of the 12th annual ACM international workshop on geographic information systems*, pages 137–146, 2004.

[5] Susanne Hambrusch, Chuan-Ming Liu, Walid G. Aref, and Sunil Prabhakar. Efficient query execution on broadcasted index tree structures. *Data and Knowledge Engineering*, 60(3):511–529, 2007.

[6] Susanne Hambrusch, Chuan-Ming Liu, and Sunil Prabhakar. Broadcasting and querying multi-dimensional index trees in a multi-channel environment. *Information Systems*, 31(8):870–886, 2006.

[7] T. Imieliński, S. Viswanathan, and B. R. Badrinath. Data on air: Organization and access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):353–372, 1997.

[8] Mohammad Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *vldb'2004: Proceedings of the Thirtieth international conference on Very large data bases*, pages 840–851. VLDB Endowment, 2004.

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

[10] Chuan-Ming Liu and Shu-Yu Fu. Effective protocols for knn search on broadcast multi-dimensional index trees. *Information Systems*, 33(1):18–35, 2008.

[11] Chuan-Ming Liu and Kun-Feng Lin. Disseminating dependent data in wireless broadcast environments. *Distributed and Parallel Databases*, (1):1–25, 2007.

[12] N. Roussopoulos, S. Kelley, and F.Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 71–79, 1995.

[13] N. Vaidya S. Hameed. Efficient algorithms for scheduling data broadcast. *ACM/Baltzer Journal of Wireless Networks*, 5(3):183–193, 1999.

[14] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Massachusetts, 1990.

[15] Shashi Shekhar and Sanjay Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2003.

[16] Baihua Zheng, Jianliang Xu, Wang-Chien Lee, and Dik Lun Lee. Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services. *The VLDB Journal*, 15(1):21–39, 2006.