An Air Index for Spatial Query Processing in Road Networks

Weiwei Sun, Chunan Chen, Baihua Zheng, *Member, IEEE*, Chong Chen, and Peng Liu

Abstract—Spatial queries such as range query and kNN query in road networks have received a growing number of attention in real life. Considering the large population of the users and the high overhead of network distance computation, it is extremely important to guarantee the efficiency and scalability of query processing. Motivated by the scalable and secure properties of wireless broadcast model, this paper presents an air index called *Network Partition Index* (*NPI*) to support efficient spatial query processing in road networks via wireless broadcast. The main idea is to partition the road network into a number of regions and then build the index to carry some pre-computation information of each region. We also propose multiple client-side algorithms to facilitate the processing of different spatial queries such as kNN query, range query and CNN query. A comprehensive experimental study has been conducted to demonstrate the efficiency of our scheme.

Index Terms—Wireless data broadcast, kNN query, air indexing, road network

1 Introduction

Mobile devices with computational and wireless communication capabilities are becoming more and more popular in our daily life. Most of these mobile devices such as smart phones and pads are equipped with positioning systems. The integration of positioning techniques and mobile computing techniques has led to the rapid rise of *Location-Based Services* (LBSs). For example, mobile users ask for nearby information via issuing spatial queries in road networks (e.g., range queries and kNN queries) [1].

In most, if not all, of the works, *point-to-point access* model is employed to process queries in road networks [2], [3], [4]. It assumes the mobile client posts a query to the server, and then the server returns the query result to the client through a point-to-point channel. Although this model is ideal for many applications, it has some disadvantages in supporting spatial query processing in the road network. For example, when multiple clients located in the same area request for the same information, this model wastes network resources to deliver the same information multiple times. In other words, this model might cause network overloading when the number of mobile clients increases. In addition, it is very hard for the clients to protect their location privacy under this model as the mobile clients have to send the details of their locations to the server.

Wireless data broadcast is an alternative to disseminate data to mobile clients. Under this model, the server periodically broadcasts the data via a wireless channel, while the clients tune in the channel to retrieve the information interested in. A broadcast of the common requested data can satisfy an arbitrary number of mobile clients simultaneously, which is of great significance in wireless networks with limited bandwidth. In other words, the unique feature that the network overload is independent of the number of clients offers the broadcast systems with super scalability. Another advantage of this model is that there is no need to pass the clients' locations to the server, and hence the location privacy of clients is well protected.

Traditional technologies utilize disk-based spatial index [1], [2], [3], [4] to speed up spatial query processing in road networks. These indexes are not suitable in broadcast model for two reasons: i) Existing indexing techniques consider only random access in disk-based environments, whereas broadcast model only supports sequential access; ii) Disk-based indexes mainly aim at reducing access latency (AT); while in wireless broadcast model, the tuning time (TT) is the other important metric, which determines the energy consumption and Internet traffic charge. The development of new air indexes for spatial query processing in wireless broadcast environments is in high demand.

In the literature, there are a lot of works on wireless data broadcast to address various system issues, among which several proposals are on delivering spatial data via wireless data broadcast. These techniques mainly focus on euclidean space [5], [6], [7]. However, in most of real life applications, the objects' movement is constrained by road networks. Recently, [8], [9] propose air indexes to support shortest path queries in wireless broadcast environments. However, the issue of supporting spatial queries (e.g., range queries and kNN queries) on road networks via broadcasting systems has not been addressed yet.

In this paper, we propose a novel spatial air index namely *Network Partition Index* (NPI) to support a variety of spatial queries in road networks. Compared with existing schemes, NPI is a more general wireless broadcast scheme to disseminate the road network data to mobile clients. The main idea

- *W. Sun, C. Chen, C. Chen, and P. Liu are with the School of Computer Science, Fudan University, Shanghai 201203, P.R., China. E-mail: {wwsun, chenchunan, chenchong, liupeng}@fudan.edu.cn.*
- *B. Zheng is with the School of Information Systems, Singapore Management University, Singapore. E-mail: bhzheng@smu.edu.sg.*

Manuscript received 30 May 2013; revised 14 Apr. 2014; accepted 19 Apr. 2014. Date of publication 12 June 2014; date of current version 23 Dec. 2014. Recommended for acceptance by W.-S. Han.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2330836

is to partition the original network into smaller cells, and to pre-compute some information (e.g., the minimum network distance between every two cells and the diameter of every cell) that will be carried by NPI. On the server side, it periodically broadcasts NPI, together with the network connectivity information of each cell. On the client side, upon issuing queries, they tune in the channel to fetch the index information first, based on which some cells that definitely do not contain the query results can be pruned away. Then, the mobile clients only need to retrieve the data corresponding to those cells that might contain the results. After downloading the subset of the road network information, the query processing is executed at the client side. The contributions of this paper can be summarized as follows:

- We propose a novel air index namely NPI to broadcast road network data to mobile clients to support spatial query processing at client side.
- We propose multiple client-side search algorithms to support different spatial queries efficiently, including range query, kNN query, and CNN query.
- An extensive simulation is constructed using real road network data to demonstrate the efficiency of NPI.

The remainder of this paper is organized as follows. Section 2 briefly introduces the wireless broadcast model and overviews the existing work on air index for LBSs, as well as the query processing techniques in road networks. Section 3 presents our NPI broadcast scheme. Section 4 discusses how to process different spatial queries at the client side. Section 5 discusses the optimal grid granularity selection. Section 6 reports the experimental evaluation results, and finally Section 7 concludes this paper.

2 Preliminaries

2.1 Wireless Data Broadcast

In the wireless data broadcast model, the server repeatedly broadcasts the data to the clients via a wireless channel, while the mobile clients tune into the broadcast channel to retrieve the data on air and process the query locally. Usually, *access latency* and *tuning time* are the main performance metrics for a wireless broadcast system [10]. The former refers to the time elapsed from the moment a query is issued to the moment it is answered; and the latter is the time a mobile client stays in active mode to receive the requested spatial data and index information. On the one hand, access latency well demonstrates the responsiveness of the system and is very important to user experience. On the other hand, tuning time is the determinant factor of the power consumption at client side [11] and hence a smaller tuning time is preferred. In addition, smaller tuning time means less Internet usage by the clients. This is desirable as the mobile users might be charged by the amount of Internet traffic.

Air indexing techniques are often used for conserving the energy of mobile clients. With indexing information (including searchable attributes and delivery time of data objects) carried by air indexes, mobile clients can find out the arrival time of desired data objects and schedule the sleep time for sub-sequential data access. Consequently, the search of data objects is facilitated.

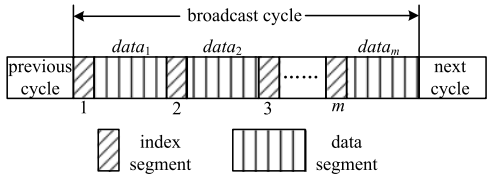


Fig. 1. (1, m) interleaving technique on the broadcast channel.

The (1, m) indexing scheme [10], as depicted in Fig. 1, is the most common organization of the broadcast cycle. It divides the broadcast data into m equal parts (data segments) and broadcasts each data segment preceded by the index on the broadcast channel. In other words, the index information is broadcast m times within one broadcast cycle. Although the repeated index information extends the broadcast cycle and hence the average access latency, it effectively cuts down the tuning time.

2.2 Spatial Query Processing in Wireless Broadcast Environments

Hambrusch et al. [12] use a traditional spatial index, i.e., R*-tree,to support range queries in wireless broadcast environments. This method can efficiently support range queries but not those with answer objects not fixed, e.g., kNN query [6]. The *D-tree* [5] and *Grid-index* [13] have been proposed to support NN queries in wireless broadcast environments. They use Voronoi Diagram to partition the service area into disjoint Voronoi cells (VCs) with each corresponding to one object. Given an object a, it is guaranteed to be the nearest neighbor to any client located inside the corresponding VC. However, both D-tree and grid-index are not general index structures as they only support 1NN query. The Hilbert space-filling curve, a spatially optimal method to transform the multi-dimensional data into a one-dimensional space, has been applied to organize spatial data in the sequentially accessed broadcast channel. Representatives include *Hilbert Curve Index* (HCI) [14] and *Distributed Spatial Index* (DSI) [6]. Mouratidis et al. [7] propose the *Broadcast Grid Index* (BGI), which outperforms the previous techniques in both static and dynamic environments. BGI uses grid cell as the index because grid cell is not only of small size but also very efficient for objects updates.

However, all above-mentioned approaches only consider the spatial queries in a euclidean space. In many real life applications, the objects' movements are constrained in a road network. The index techniques such as R*-tree based method or grid cell can't be directly applied in road networks because the network distance (i.e., the shortest path distance) can't be computed using only the boundary of the *minimum boundary rectangle* (MBR) or grid cell. The same problem occurs in the Voronoi-based technique and the Hilbert-curve based methods.

2.3 Spatial Query Processing in Road Networks

In general, a road network is modeled as an undirected graph $G(V, E)$, with V being the set of vertices and E being the set of edges. As road networks usually are sparse graphs, it is common to store G using adjacency lists. An edge $(v_i, v_j) \in E$ represents that vertices v_i and v_j are connected in the network. The weights of edges are captured by W. A

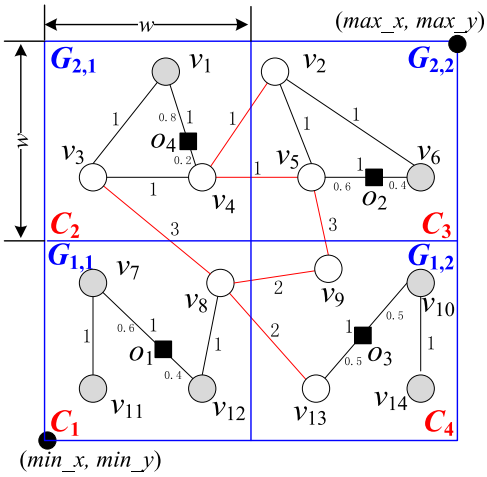


Fig. 2. Partitioning a road network into 2×2 grids.

non-negative weight $w(v_i, v_j) \in W$ of edge $(v_i, v_j) \in E$ can represent physical distance, travel time or other costs according to different application context. Given two vertices v_i and v_j of a graph $G(V, E)$, a path and the shortest path connecting them are defined in Definition 1. In this paper, the distance (or the network distance) between two vertices in a road network refers to their shortest distance.

Definition 1 (Path and Shortest Path). *Given a road network $G(V, E)$ and $v_i, v_j \in V$, a path $P(v_i, v_j)$ connecting v_i and v_j sequentially passes vertices $v_{p1}, v_{p2} \ldots, v_{pm}$, denoted as $P(v_i, v_j) = \{v_i, v_{p1}, v_{p2}, \ldots, v_{pm}, v_j\}$. The length of $P(v_i, v_j)$, denoted as $|P(v_i, v_j)|$, is $\sum_{n=0}^{m} w(v_{pn}, v_{p(n+1)})$ with $v_{p0} = v_i$ and $v_{p(m+1)} = v_j$. The shortest path $SP(v_i, v_j)$ is the one with the shortest length among all the paths from v_i and v_j, and its length, denoted as $||v_i, v_j||(= |SP(v_i, v_j)|)$, is the network distance between v_i and v_j.*

Fig. 2 depicts an example road network, which serves as a running example in the remainder of this paper. It consists of 14 vertices and 17 edges. The integer next to each edge indicates its weight. From Fig. 2 we can easily see that $SP(v_1, v_5) = \{v_1, v_4, v_5\}$ and $||v_1, v_5|| = 2$.

In traditional spatial databases, spatial query processing in road networks has been well studied. R-trees have been used to store the objects, the crosses in the road network, and the shapes of the road segments [3]. Two different algorithms, namely IER and INE, based on this storage schema have been proposed. NVD applies Voronoi Diagrams on road networks to support general kNN queries [2]. The network embedding framework has also been utilized to process kNN queries in road networks [1]. A distance signature based solution (DS) has been proposed to process range query and kNN query in road networks [15]. Shortest Path Quad-tree (SPQ) has also been proposed to support best-first kNN search [4].

However, the query processing techniques mentioned above can't be applied directly in wireless broadcast environments. This is because in the traditional spatial database, data are stored in disk or memory and can be accessed randomly. Consequently, the algorithms developed assume the data are always available. However, in wireless broadcast systems, the data are broadcast based on certain fixed order. Once the information is broadcast, the client has to wait for the next time the same information is broadcast as the information is only available when it is broadcast.

Due to space limitation, we only review three representative approaches, i.e., IER, INE, and NVD. Their inefficiency is actually common to other approaches as well. Given a query point q and an integer k, IER needs search the R-tree index to retrieve the kNN of q according to euclidean distance. As discussed in Section 2.2, kNN algorithms using R-tree index in wireless broadcast environments may cause long access latency due to backtracking. INE shares some similarity with Dijkstra's algorithm [16] when exploring the road network. From the query issuing point, it expands the road network based on network distance, while the adjacency lists of all the nodes are broadcast by the server based on certain fixed order. In other words, it is very likely that when the client needs to retrieve the adjacency list of node n, it has been already broadcast in the current cycle. The client has to wait until the next broadcast cycle to retrieve n's information, which results in long access latency. Alternatively, the client may read the data in the entire broadcast cycle to cut down the access latency, in cost of large tuning time. NVD suffers from the same problem. Suppose a 2NN query is issued. The client can find its nearest neighbor by locating the network Voronoi cell, says V_q, it is in. Then, it has to explore V_q's adjacent diagrams in order to find the second nearest neighbor. However, some of the V_q's adjacent diagrams might be broadcast earlier than V_q in the current broadcast cycle. The client has to either wait until the next cycle or blindly retrieve all the data within one cycle.

The wireless broadcast model is first adopted by Kellaris and Mouratidis [8] to provide LBSs in road networks. Two different methods, namely *EB* and *NR*, are proposed to support shortest path computation on air. An energy-efficient air index, namely *BagIndex*, has also been proposed to support shortest path queries. Both [8] and [9] only support shortest path queries but not common spatial queries such as range query and kNN query. A more general index that can support multiple spatial queries is desired.

3 NPI: Air Index for Spatial Query Processing in Road Networks

The result of a spatial query often lies somewhere near the client's location. For example, the 10 nearest restaurants to a query point q are centered at q and they are not far away from q. This is because: i) points of interest such as restaurants are usually very dense in a real road network, and ii) the value of k posted by the mobile user won't be too large because it is meaningless and not practical for a mobile device to deal with hundreds or thousands of objects. In other words, the client needs not to read the network information too far away.

Motivated by this observation, we partition the original road network into small regions and pre-compute certain information for each region, which forms the index (i.e., NPI). The index information, as well as the adjacency lists of each region, is interleaved in the broadcast channel. In this section, we first introduce the network partition and

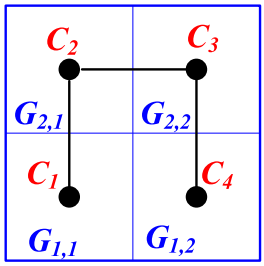


Fig. 3. Grid cells ordered by Hilbert Curve.

some important concepts. Then, we present the information carried by NPI. Next, we discuss the content of the data segment, i.e., the road network structure represented by the adjacency lists as well as the objects location and description. Finally, we explain how to interleave the index and data segment in a wireless channel.

3.1 Grid Partition of the Road Networks

There are several strategies to partition the road network [17]. In this paper, we adopt the grid partitioning algorithm for the size of a grid index is very small. Other partition methods such as quadtree partition or kd-tree partition can be applied into our NPI too. The road network is partitioned into $n \times n$ $(= N)$ equal-size grids. Here, n refers to the number of grids in each dimension. All the grid cells $G_{i,j}$ $(1 \leq i,j \leq n)$ are in square shape with w being the side length. The header of NPI contains the grid partition information, i.e., the minimum/maximum x/y coordinate of the service area (denoted as min_x/min_y, max_x/max_y), and the grid width w. Given a client located at a position $p(x, y)$, the grid cell $G_{i,j}$ that the client falls in can be computed with Equation (1). As shown in Fig. 2, the road network is partitioned into 2×2 (i.e., $n = 2$ and $N = 4$) grids. The nodes v_1, v_3, and v_4 are in the grid cell $G_{2,1}$, the nodes v_2, v_5, and v_6 are in the grid cell $G_{2,2}$, and so on.

$$\begin{cases} i = \lceil (y - min\_y)/w \rceil \\ j = \lceil (x - min\_x)/w \rceil. \end{cases} \quad (1)$$

For the purpose of convenience in expression, we assign each cell with one unique ID (from 1 to N) based on some specific mapping function. In the rest of this paper, we use the notation C_a to refer to the cell with ID of a and avoid using the two-dimensional notation $G_{i,j}$. Fig. 3 shows the mapping result of the cells in Fig. 2. For example, C_1 refers to the grid cell $G_{1,1}$ and C_3 refers to the grid cell $G_{2,2}$. The details of the mapping function will be given later.

3.2 Pre-Computation Information

As we mentioned before, NPI contains the grid partition information in its header. Now we explain the second type of information carried by NPI, i.e., some pre-computation information of the road network. Before we present the details of the pre-computation information, we first introduce the concept of *border points*, as defined in Definition 2.

Definition 2 (Border points). *Given a vertex u located in the grid cell C_a, u is called a border point of C_a if u is connected to another vertex v that is located in another grid cell C_b $(a \neq b)$ via an edge (u, v).*

$$\begin{pmatrix} d_{1,1} & \cdots & d_{1,N} \\ \vdots & \ddots & \vdots \\ d_{N,1} & \cdots & d_{N,N} \end{pmatrix} = \begin{pmatrix} \text{Null} & (3,4) & (5,5) & (2,2) \\ (3,4) & \text{Null} & (1,2) & (4,6) \\ (5,5) & (1,2) & \text{Null} & (3,7) \\ (2,2) & (4,6) & (3,7) & \text{Null} \end{pmatrix}$$

Fig. 4. Example distance bound matrix.

Border points refer to those vertices lying on the edges bypassing more than one grid cell. For example, as shown in Fig. 2, vertices v_2 and v_4 are border points as the corresponding edge (v_2, v_4) bypasses two grid cells, i.e., C_2 and C_3. All the border points are indicated by hollow-circles in Fig. 2. We use BP_a to refer to all the border points located inside cell C_a, e.g., $BP_1 = \{v_8\}$ and $BP_2 = \{v_3, v_4\}$.

Obviously, the border points of a given cell C_a serve as the only entrances to and exits from C_a, e.g., all the paths from a point located outside cell C_1 to a point inside C_1 must pass v_8, the only border point of C_1. Consequently, for each pair of cells C_a and C_b, we pre-compute the minimum network distance $\alpha_{a,b}$ (or maximum network distance $\beta_{a,b}$) between any border point p of C_a and any border point p' of C_b, i.e., $\alpha_{a,b} = MIN_{p \in BP_a, p' \in BP_b}(||p, p'||)$, and $\beta_{a,b} = MAX_{p \in BP_a, p' \in BP_b}(||p, p'||)$. $\alpha_{a,b}$ and $\beta_{a,b}$ can facilitate the approximation of the network distance from a point in cell C_a to a point located in another grid cell C_b. This will be detailed in the next section.

Take the cells C_2 and C_4 in our example road network as an example. Since $BP_2 = \{v_3, v_4\}$ and $BP_4 = v_9, v_{13}$, $\alpha_{2,4} = MIN(||v_3, v_9||, ||v_3, v_{13}||, ||v_4, v_9||, ||v_4, v_{13}||) = ||v_4, v_9|| = 4$, and $\beta_{2,4} = MAX(||v_3, v_9||, ||v_3, v_{13}||, ||v_4, v_9||, ||v_4, v_{13}||) = ||v_4, v_{13}|| = 6$. These distance data are maintained by a $N \times N$ matrix M, namely *distance bound matrix*. Each element of M, denoted as $d_{a,b}$, contains the lower bound $\alpha_{a,b}$ and the upper bound $\beta_{a,b}$ of the network distance between the border points of cell C_a and border points of another cell C_b. That's, $d_{a,b} = (\alpha_{a,b}, \beta_{a,b})$, for $(1 \leq a, b \leq N)$. The distance bound matrix for our example road network is depicted in Fig. 4. The other pre-computation information is the *diameter* of each cell, as defined in Definition 3. Back to our example network. The diameters of C_1, C_2, C_3, and C_4 are 3, 1, 1 and 6 respectively (notice that $dia(C_4) = ||v_9, v_{14}|| = 6$).

Definition 3 (Cell Diameter). *Given a grid cell C_a, the diameter of C_a is defined as the maximum distance between all pairs of vertices located in C_a, i.e., $dia(C_a) = MAX_{v,v' \in C_a}(||v, v'||)$.*

In addition, the number of objects located inside each cell is also pre-computed. The objects in C_a is denoted as $C_a.obj$, and the number thereof is denoted as $|C_a|$. The four black squares in Fig. 2 indicate the objects in the road network. We can easily see that $|C_1| = |C_2| = |C_3| = |C_4| = 1$.

The structure of NPI is depicted in Fig. 5. To be more specific, NPI contains three components: i) the header part which contains the grid information, i.e., the size of the road network represented by (min_x, min_y) and (max_x, max_y), the grid width w, and an array of N elements, with each element e_i carrying the number of objects in C_i and the diameter of C_i respectively, i.e., $e_i = (|C_i|, dia(C_i))$; ii) a pointer array with each p_i pointing to the next broadcast time of the ith row in the distance bound matrix M; and iii) the distance bound matrix M.

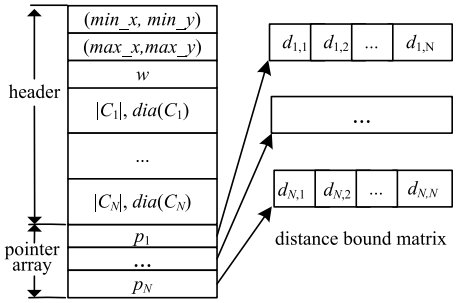


Fig. 5. Structure of NPI.

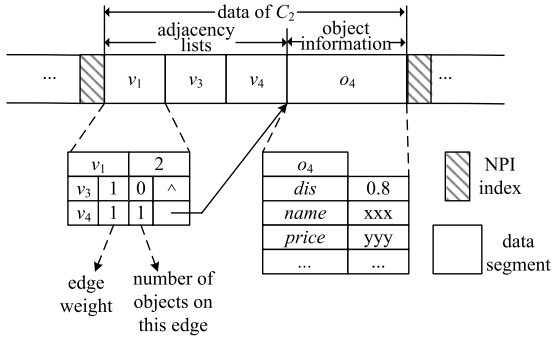


Fig. 6. Data structure of C_2.

3.3 Data Segment

As depicted in Fig. 1, a broadcast program contains index and data. In the context of this work, the index refers to NPI, while the data refers to the network structure of the road network and the object information[1], in the unit of grid cell. Here, we assume the road network structure is captured by the adjacency lists.

For example, the data corresponding to the grid cell C_2 of our example network is depicted in Fig. 6. The first part is the adjacency lists corresponding to all the vertices inside C_2, which are broadcast according to order of the node ID. Take the structure of v_1's adjacency list as an example. The integer "2" in the first row indicates the number of neighbors of v_1; in the following two rows, those two neighbors are listed. Each neighbor is represented by a four-tuple vector. Take the vector (v_3, 1, 0, $null$) corresponding to v_3 as the example. The first field "v_3" is the adjacent node which means v_3 is connected to v_1; the second field is an integer "1" indicating the weight of the edge (v_1, v_3); the third field is an integer "0" which means the number of object lying on this edge; and the last field is a pointer (offset in the channel) which points to the first object on this edge. If there is no object on this edge, the pointer is set to $null$.

Notice that objects corresponding to the same edge are always broadcast together. In other words, with the third and fourth fields, the client is able to retrieve all the objects associated with a given edge. All the objects are broadcast in the second part right after the adjacency lists, as shown in Fig. 6. First, dis indicates the distance from the object to the vertex of the edge with smaller node ID, i.e., it represents the location of object along the edge. In our example, $dis = 0.8$ means: along the edge (v_1, v_4), the distance between object o_4 and the vertex with smaller ID (i.e., v_1) is 0.8. Then, different information of the objects can be carried by the textual description, which is based on the application settings. As shown in Fig. 6, we assume the objects are restaurants, and the textual description it carries includes the name of the restaurant, the average price, and so on.

3.4 Data Organization on Air

After presenting the index structure and data segment content, we are ready to discuss how to organize them in a wireless channel. First, as service area is partitioned into N grid cells, we need to decide the broadcast order of grid cells. The order might not be important for some queries, but it has a direct impact on the performance of spatial query processing. In this paper, we use the *Hilbert space filling curve* as the mapping function to sort grid cells into a one-dimensional space, which is widely used for spatial query processing in wireless broadcast environment [6], [7], [14]. Fig. 3 depicts the Hilbert curve for a (2×2) grid of our example road network. The Hilbert values for cell $G_{1,1}$ and $G_{2,2}$ are 1 and 3, respectively. According to [14], given a cell $G_{i,j}$, the client can compute the Hilbert value of $G_{i,j}$ in a constant time. Moreover, as Hilbert curve can preserve the spatial locality, the cells with close Hilbert values are normally close to each other as well. In other words, the grid cells that contain the objects requested by a user will appear closely in the wireless channel, which will facilitate the client's data retrieval process.

We assume NPI is much smaller, compared with the data segments, and hence we adopt $(1, m)$ scheme as the data organization strategy for better tuning time performance. Each index segment contains the entire NPI. The grid cells are ordered based on their Hilbert values, and their data are partitioned into m parts with each part carried by one data segment. In the broadcast channel, we interleave index segments with data segments, as shown in Fig. 1.

4 Client-Site Query Processing

As explained before, the NPI enables the client to easily get the following information:i) the distribution of the objects; ii) the approximate network distance from the query point to other grid cells and hence to the objects; and iii) the arrival time of the data in each grid cell. Based on the above information, clients can answer various kinds of spatial queries in road networks. In this section, we discuss the search algorithms based on NPI for range queries, snapshot kNN queries, and continuous kNN queries.

The basic idea of the search algorithms is to first locate the grid cell containing the client and then to read the lower and upper bound of its distance to other grid cells. According to the distance bounds and the distribution of the objects, the client can get the locations of the candidate objects. Then, following the NPI, the clients can retrieve the cells containing the candidate objects but ignore the rest. When the candidate objects are locally available, the clients can locate the result objects based on real network distance via local processing. Because the mobile clients usually ask for nearby information (e.g., the range of range queries is normally small), the number of candidate cells is much smaller compared with the total number of cells. In other words, the retrieved grid

1. In this work, the object information contains the object's location and its description.

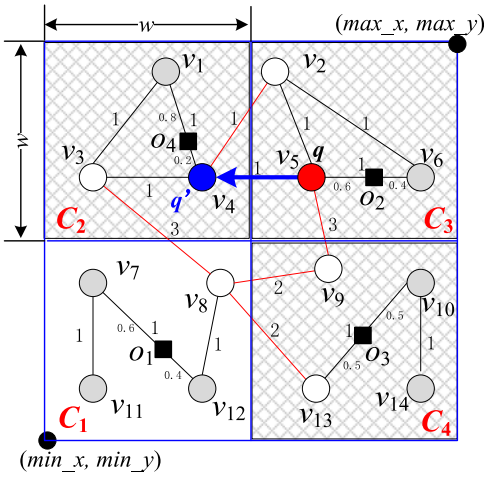


Fig. 7. Range query processing.

cells will form a sub-graph that is significantly smaller than the original road network. In other words, it is practical for mobile clients to adopt some basic road network exploring techniques for the local processing. In our simulation, we adapt Dijkstra's algorithm because it is simple and efficient in the small sub-graph downloaded by clients.

Before we present different search algorithms, we highlight an important assumption we make. Like many other existing works [4], [8], [9], we assume the clients' locations are located at network nodes to simplify our discussion. However, our algorithms can be easily extended to support cases where clients' locations are located along the network edges.

4.1 Range Queries

Given a query point q, a value d and a dataset S in a road network, a range query retrieves all the objects in S that are within the network distance d from q, denoted as $Range(q, d, S) = \{o \mid o \in S \wedge ||o, q|| \leq d\}$. As the network distance between objects a and b will not be shorter than the lower bound $\alpha_{i,j}$ between cells C_i and C_j with $a \in C_i.obj$ and $b \in C_j.obj$ (i.e., $||a, b| \geq \alpha_{i,j}$), only those cells C_n with their lower bound distances $\alpha_{n,q}$ to cell C_q bounded by d may contain the result objects, i.e., $Range(q, d, S) \subseteq \{o \mid o \in \bigcup_{\alpha_{n,q} \leq d} C_n.obj\}$. The correctness of above statement is guaranteed by Lemma 1.

Lemma 1. *Given a point q located at grid cell C_q and a grid cell C_n, if $\alpha_{n,q} > d$, the network distance from any object inside C_n to q is larger than d, i.e., $q \in C_q \wedge \alpha_{n,q} > d \Rightarrow \forall\, o \in C_n.obj$, $||o, q|| \geq \alpha_{n,q} > d$.*

Proof. The proof is straightforward and hence it is ignored for space saving. □

Based on above finding, the algorithm for supporting range query is straightforward. We adopt the filtering-and-refinement strategy, and Algorithm 1 lists its pseudo-code. In the filtering phase, the client listens to NPI. It locates the cell C_q it lies in based on its location and the grid partition information, and then fetches the qth row of the distance bound matrix (lines 1-4). Based on $\alpha_{i,q}$, it can decide all the cells that might contain the result objects and filter out the rest (lines 5-7). It then retrieves the candidate cells one by one (lines 8-12), constructs a sub-graph locally based on all the candidate cells, and finally retrieves the result objects using Dijkstra's algorithm to finish the processing (lines 13). It is noticed that the shortest path located based on the sub-graph locally at client side is the real shortest path guaranteed by Lemma 2.

Algorithm 1: NPI-Based Range Query Processing on Air

Input: a source point q, a value d and a dataset S
Output: $Range(q, d, S)$
Procedure:
1: TuneIntoChannel();
2: $NPIHeader$ = retrieveIndexHeader();
3: locate the grid cell C_q containing q;
4: read the q^{th} row R_q corresponding to C_q in the matrix;
5: **for** each cell C_i **do**
6: **if** $\alpha_{q,i} \leq d$ **then** add C_i to the candidate cells;
7: sort the candidate cells by their arrival time;
8: **for** each candidate cell **do**
9: sleepUntilCellBroadcast();
10: TuneIntoChannel();
11: $adjacencyLists$ = retrieveCell();
12: $subGraph$.add($adjacencyLists$);
13: **return** DijkstraExpansion($subGraph$, q, d);

Lemma 2. *Given a point q located at grid cell C_q and a distance bound d, let set C_{can} be the set of grid cells C_n with $\alpha_{n,q} \leq d$, i.e., $C_{can} = \{C_n \mid \alpha_{n,q} \leq d\}$. For any given object located in the road network, if its shortest distance to q is bounded by d, its shortest path only passes the cells inside C_{can}. In other words, the local shortest paths from q located in the partial network formed by grid cells in C_{can} with distance bounded by d are the real shortest path.*

Proof. Let G' denote the partial road network formed by the grid cells of C_{can}. Assume the lemma is not valid, and there is at least one object o whose shortest path $P_1(q, o)$ located based on G' is not the real one. In other words, the real shortest path $SP(q, o)$ must pass at least one grid cell C_j outside C_{can}, i.e., $\exists v \in C_j \wedge C_j \notin C_{can}$ such that $v \in SP(q, o)$. As $SP(q, o)$ is the real shortest path, and $P_1(q, o)$ is the local shortest path, $|SP(q, o)| < |P_1(q, o)| \leq d$. As v is on the shortest path $SP(q, o)$, $|SP(q, v)| < |SP(q, o)| < d$. Given the fact that $q \in C_q$ and $v \in C_j$, we can derive $\alpha_{j,q} \leq d$ based on the finding that $SP(q, d) < d$ according to Lemma 1. As $C_{can} = \{C_n \mid \alpha_{n,q} \leq d\}, C_j \in C_{can}$ that contradicts with our assumption $C_j \notin C_{can}$. Consequently, our assumption is invalid, and the proof completes. □

We use an example to explain the query processing for a range query. As shown in Fig. 7, there are four objects in the road network (i.e., o_1, o_2, o_3, o_4) and a range query is issued at q (i.e., located at node v_5) with $d = 3$. The client first tunes into the channel and retrieves the header of NPI. Based on the size of the service area and the width of the grid cell, the client understands that it is located inside cell C_3 according to Equation (1). Then, it fetches the third row of the distance bound matrix and knows the minimum distances from a

border node of C_3 to a border node of C_1, C_2, C_3 and C_4 are 5, 1, 0 and 3, respectively. As $d = 3$, objects in cell C_1 are definitely not qualified and hence C_1 is filtered out. In other words, cells C_2, C_3 and C_4 are the candidate cells whose adjacency lists shall be retrieved by the client in order to find out the result objects. Accordingly, the client downloads the data segments of C_2, C_3 and C_4, and re-constructs the partial road network locally, i.e., the shaded area in Fig. 7. Finally, the client explores the partial road network from q based on Dijkstra's algorithm to find out the result objects o_2 and o_4.

4.2 kNN Queries

Given a query point q and a dataset S in a road network, a k nearest neighbor (kNN) query retrieves the k objects in S whose network distances to q are the k smallest, denoted as $kNN(q, k, S) = \{O = \bigcup_{i\in[1,k]} o_i \,|\, O \subseteq S \wedge \forall o' \in S - O,\ \forall o_i \in O, ||q, o'|| \geq ||q, o_i||\}$. Different from range queries, the search area of an kNN query is not fixed as it depends on the location of q and the value of k. In order to find out the candidate cells, we need to decide the upper bound distance d_{max} between q and any of its result objects, i.e., $\forall o_i \in O$, $||q, o_i|| \leq d_{max} \wedge \exists o' \in O$, $||q, o'|| = d_{max}$. Once d_{max} is decided, an kNN query can be converted into a range query, i.e., retrieving the objects that are within network distance d_{max} to q. In the following, we explain how to estimate d_{max} based on NPI.

Our estimation of d_{max} is based on the upper bound of the network distance between any object in C_u and any object in C_v, denoted as $UB(C_u, C_v)$, i.e., $\forall u \in C_u.obj, \forall v \in C_v.obj$, $||u, v|| \leq UB(C_u, C_v)$. Recall that NPI keeps the diameter $dia(C_u)$ of each cell C_u and the maximum distance $\beta_{u,v}$ between border nodes of two cells C_u and C_v. Then, we can derive $UB(C_u, C_v)$, as stated in Lemma 3.

Lemma 3. *Given two cells C_u and C_v, the network distance between any object u in C_u and any object v in C_v is bounded by $(dia(C_u) + dia(C_v) + \beta_{u,v})$, denoted as $UB(C_u, C_v) \leq dia(C_u) + dia(C_v) + \beta_{u,v}$.*

Proof. Assume the above statement is not valid, and there is at least one pair of objects (u, v) such that u is in C_u and v is in C_v with $||u, v|| > dis(C_u) + dia(C_v) + \beta_{u,v}$. As u and v are located in different cells, the shortest path between u and v will pass a border node b_1 of C_u and a border node b_2 of C_v. In other words, $SP(u, v)$ starts from u, then leaves C_u via b_1, reaches C_v via b_2, and finally reaches v, i.e., $||u, v|| = ||u, b_1|| + ||b_1, b_2|| + ||b_2, v||$. As $||b_1, b_2|| \leq \beta_{u,v}$, $||u, b_1|| \leq dia(C_u)$ and $||b_2, v|| \leq dia(C_v)$, $||u, v|| \leq dis(C_u) + dia(C_v) + \beta_{u,v}$. Consequently, our assumption is not valid, and our proof completes. □

With the help of Lemma 3, d_{max} can be estimated as follows. Given a query point q located in a cell C_q, we access the cells C_i based on non-descending order of $\alpha_{q,i}$, i.e., C_i will be visited earlier than C_j if $\alpha_{q,i} < \alpha_{q,j}$. If there is a tie (i.e., there are two cells C_i and C_j with $\alpha_{q,i} = \alpha_{q,j}$), $UB(C_q, C_i)$ and $UB(C_q, C_j)$ are used as the tie-breaker and the one with smaller UB value will be visited first. Two parameters are maintained during this process, one to count the total number of objects in all the cells visited so far, denoted as $count$, and the other being $UB(d_{max})$, the upper-bound of d_{max}. For each visited cell C_i, we calculate $UB(C_q, C_i)$ and set $UB(d_{max})$ to the maximum $UB(C_q, C_i)$ found so far, and meanwhile we check $|C_i|$ and update $count$. This process continues until $count$ reaches k. Lemma 4 proves the correctness of above process to approximate d_{max}.

Lemma 4. *Suppose a road network is partitioned into N grid cells C_i with $i \in [1, N]$ and an kNN query is issued at point q located at grid cell C_q. Assume a set C' consists of a few cells which satisfy following conditions: i) $C' \subseteq \bigcup_{i\in[1,N]} C_i$; ii) $\forall C_a \in C', \forall C_b \in (\bigcup_{i\in[1,N]} C_i - C')$, $\alpha_{q,a} \leq \alpha_{q,b}$; and iii) $\sum_{C_a\in C'} |C_a| \geq k$. Then, $d_{max} \leq MAX_{\forall C_a \in C'} UB(C_q, C_a)$.*

Proof. Assume the above statement is not valid, and there is at least one result object $o \in kNN(q, k, S)$ with $d_{max} \geq ||q, o|| > MAX_{\forall C_a\in C'} UB(C_q, C_a)$. Based on Lemma 3, we know object o must be located in some cell C_o such that $C_o \notin C'$. On the other hand, as $\sum_{C_a\in C'} |C_a| \geq k$ and $o \in kNN(q, k, S)$, there must be at least one object o' located in a cell $C_{o'}$ ($\in C'$) such that $o' \notin kNN(q, k, S)$. Then $||q, o'|| \leq UB(C_q, C'_o) \leq MAX_{\forall C_a\in C'} UB(C_q, C_a) < ||q, o||$ that contradicts with our assumption. Consequently, the assumption is invalid and the proof completes. □

After finding $UB(d_{max})$, the client can retrieve the kNN candidates Can_q via a range query with $d = UB(d_{max})$. With the help of the single source Dijkstra's algorithm, the client can explore the partial road network, and the first k visited objects in Can_q will be returned as the result set. The pseudo-code of the kNN query algorithm is shown in Algorithm 2.

Algorithm 2: NPI-Based kNN Query Processing on Air

Input: a source point q, an integer k and a dataset S
Output: k objects in S closest to q
Procedure:
1: $Q \leftarrow \emptyset, count \leftarrow 0, UB(d_{max}) \leftarrow 0$;
2: TuneIntoChannel();
3: $NPIHeader$ = retrieveIndexHeader();
4: locate the grid cell C_q containing q;
5: read the q^{th} row R_q corresponding to C_q in the matrix;
6: sort the cells C_i based on non-descending order of $\alpha_{q,i}$ and maintained in Q;
7: **while** Q is not empty **do**
8: $C_i \leftarrow Q$.pop();
9: $count = count + |C_i|$;
10: $UB(d_{max}) = MAX(UB(d_{max}), UB(C_q, C_i))$;
11: **if** count $\geq k$ **then** break;
12: ($subGraph$, Can_q) =$rangeQuery(q, UB(d_{max}), S)$;
13: **return** DijkstraExpansion($subGraph$, Can_q, q, k);

Recall the example in Fig. 7. Suppose a 2NN query is issued at q. The client first needs to find out $UB(d_{max})$. It first locates the cell C_3 it lies in, and visits the cells C_i in the order of (C_3, C_2, C_4, C_1), following the non-descending order of $\alpha_{3,i}$. As $|C_3| = 1$ and $|C_2| = 1$ (i.e., $|C_3| + |C_2| = k$), $UB(d_{max})$ is set to $MAX(UB(C_3, C_3), UB(C_3, C_2)) = MAX(dia(C_3), dia(C_3) + dia(C_2) + \beta_{3,2}) = dia(C_3) + dia(C_2) + \beta_{3,2} = 4$. Next, the client needs to retrieve all the cells C_i with

$\alpha_{3,i} \leq UB(d_{max})$, i.e., cells C_2, C_3 and C_4, to construct the partial road network locally. Finally, with Dijkstra's algorithm, the first two visited objects o_2 and o_4 are returned as the result objects.

4.3 Continuous NN Queries

In real applications, mobile users might issue queries when they are moving. For example, a taxi driver may keep asking for the closest client while he/she is driving. This is known as the *continuous nearest neighbor* (CNN) queries [18], [19]. Given a query path $P(v_1, v_2, \ldots, v_L)$ in a road network, the CNN query finds a set of kNN results corresponding to each segment (namely *valid interval*) in P. The kNN results of all query points lying on one valid interval are identical, and the start/end points of all the valid intervals are called *split points*. A naive solution to support CNN queries is to continuously issue kNN queries along each moving point. However, it is not efficient as the answers to kNN queries issued at nearby locations might be same. Consequently, a more efficient search algorithm is needed to support CNN queries.

Recall that kNN search algorithm presented above utilizes the parameter $UB(d_{max})$ to find the answer objects. Given two kNN queries issued at two different locations within the same grid cell, they share the same $UB(d_{max})$. In other words, the NPI-based kNN search algorithm only considers the grid cell where the query is issued, but not the exact location of the query point. That is to say the kNN candidates and the partial road network remain the same if the client does not move out of the current grid cell. Consequently, if the client issues kNN queries when moving, she/he needs not tune into the channel to download new candidates or new adjacency lists if she/he is still within the current grid cell. We also find that even if the client leaves the current grid cell but has not moved a long distance, the kNN candidates and partial road network will still remain the same, as stated in Lemmas 5 and 6.

Lemma 5. *Given a query point q, let $UB(d_{max})$ be the upper bound calculated using Lemma 3 and Lemma 4, Can_q be the set of kNN candidates returned by algorithm 2, and d_k be the distance from q to its k^{th} NN. For a new query point q', if $||q, q'|| \leq (UB(d_{max}) - d_k)/2$, then Can_q contains q''s kNN, i.e., $||q, q'|| \leq \frac{UB(d_{max})-d_k}{2} \Rightarrow kNN(q', k, S) \subseteq Can_q$.*

Proof. Assume the above statement is invalid, and one of q''s k nearest neighbors o' is not in Can_q. Let $\{o_1, o_2 \ldots, o_k\}$ be the kNN result of q. Based on the triangle inequality, we have $(1 \leq i \leq k)$

$$||q', o_i|| \leq ||q', q|| + ||q, o_i|| \leq ||q', q|| + d_k.$$

For $||q', q|| \leq \frac{UB(d_{max})-d_k}{2}$, we have $(1 \leq i \leq k)$

$$||q', o_i|| \leq \frac{UB(d_{max})-d_k}{2} + d_k = \frac{UB(d_{max})+d_k}{2}.$$

That is to say, there are k objects (i.e., $o_1, o_2, \ldots, o_k$) whose distances to q' are smaller than $(UB(d_{max}) + d_k)/2$. For o' is not in Can_q, so o' is in the grid cell whose minimum distance to q is larger than $UB(d_{max})$, that is, $||q, o'|| > UB(d_{max})$. As $||q, o'|| \leq ||q, q'|| + ||q', o'||$, we have $||q', o'|| \geq ||q, o'|| - ||q, q'|| > UB(d_{max}) - ||q', q||$. For $||q', q|| \leq \frac{UB(d_{max})-d_k}{2}$, we have $||q', o'|| > UB(d_{max}) - \frac{UB(d_{max})-d_k}{2} = \frac{UB(d_{max})+d_k}{2}$. Consequently, there are k objects (i.e., $o_1, o_2, \ldots, o_k$) whose distances to q' are smaller than $||q', o'||$. This contradicts our assumption and the proof completes. □

Lemma 6. *Assume q and q' satisfy all the conditions specified in Lemma 5, and let G' be the partial road network downloaded for kNN search at the query point q. The shortest path from q' to its kNN won't pass a node outside G'.*

Proof. The proof is straightforward based on the proof of Lemma 2 and Lemma 5, and hence it is ignored for space saving. □

Continue the example 2NN query depicted in Fig. 7 with $UB(d_{max}) = 4$, the partial road network G' formed by cells C_2, C_3 and C_4, $Can_q = \{o_2, o_3, o_4\}$, and $d_k = ||q, o_4|| = 1.2$. Suppose the client moves to a new location v_4. Although v_4 is located in a different grid cell, the moving distance $||v_4, v_5|| = 1 < (UB(d_{max}) - d_k)/2$. Consequently, the 2NN of q' is still within Can_q and G' remains valid. If the moving distance of the client exceeds $(UB(d_{max}) - d_k)/2$, the client needs to tune into the channel to get the new kNN candidates and retrieve new partial road network. At the downloading step, if the new partial road network overlaps with the former one, we can ignore the retrieval of the overlapped portion as it is locally available.

Accordingly, we propose to decompose a given CNN query path $P(v_1, v_2, \ldots, v_l)$ into disjoint *space valid segments*, denoted as $SVS_1, SVS_2, \ldots, SVS_l$, and all the query points lying on a space valid segment share the same kNN candidates and partial road network. In other words, the client only needs to issue an kNN query at the starting point of a space valid segment to retrieve the kNN candidates and partial road network. For the rest points along the space valid segment, no downloading from the wireless channel is necessary. The pseudo-code of the query path decomposition is depicted in Algorithm 3. We want to highlight that this step is the key of our CNN algorithm, which helps to reduce the tuning time and shorten access latency significantly.

Algorithm 3: NPI-Based Space Validation for CNN

Input: a query path $P(v_1, v_2, \ldots, v_L)$
Output: space valid segment $(SVS_1, SVS_2, \ldots, SVS_l)$
Procedure:

```
1: start ← 1; G'' ← ∅; n = 1
2: while start ≤ L do
3:    TuneIntoChannel();
4:    C_q ← the grid cell that contains v_start; G' ← ∅;
5:    Compute UB(d_max) of v_start using Algorithm 2;
6:    for each grid cell C_i with α_{i,q} ≤ UB(d_max) do
7:       if C_i ∈ G'' then copy C_i to G';
8:       else retrieve C_i and add it to G' and G'';
9:    Search the kNN of v_start based on G';
10:      Find the first node v_i in P that ||v_start, v_i|| >
         (UB(d_max) − d_k)/2;
11:   Output SVS_n (v_start, ..., v_{i−1}); n++
12:   start = i;
```

Suppose a given query path is decomposed into several space valid segments, we now explain how to process CNN locally for a given space valid segment. We adopt the approach proposed in [18] to further decompose a space valid segment into several valid intervals via split points. Given the fact that the result set remains the same for CNN queries issued at any point along a valid interval, we only invoke an kNN query at the starting point of a valid interval.

5 Grid Granularity Selection

Notice that all the algorithms we develop are based on the pre-computed distance bound information. The more precise the distance bound is, the more powerful the pruning is, the less the information retrieval is and hence the more efficient the query processing is. Obviously, the grid size directly affects the precision of the distance bounds. In an extreme case where each grid contains at most one point, the distance bound reflects the *real* network distance between any two points. In order to facilitate the selection of a proper grid size, we develop an analytical model to analyze the impact of grid size on the system performance.

Intuitively, a fine granularity achieves tighter bound of the search space but leads to larger index size. Both the size of the index and the size of grid cells affect the tuning time and access latency, so there is a tradeoff between different grid partitions. Recall that we always partition the entire road network into $2^i \times 2^i$ uniform grids, in purpose of utilizing the Hilbert curve. Let PS_i be a uniform partitioning strategy that partitions the service area into $2^i \times 2^i$ uniform grids. For simplicity, we assume that both the nodes and the objects of the road network are uniformly distributed in each grid cell. Let DS be the data size (in Bytes) of the entire road network, including the adjacency lists and the object information. Then the data size of each grid GS_i of PS_i is $\frac{DS}{2^i \times 2^i} = \frac{DS}{4^i}$. For the index size, as depicted in Fig. 5, assume that both the number of objects (i.e., $|C_i|$) and the distance between two points (i.e., $dia(C_i)$, $\alpha_{a,b}$, and $\beta_{a,b}$) occupy 4 Bytes, then the size of the NPI index IS_i is $12NG_i + 8NG_i^2 = 4^{i+1}(3 + 2 \times 4^i)$. As we employ the (1, m) index scheme, the cycle length of PS_i is

$$Cycle_i = DS + m \times IS_i.$$

Let $\overline{r}$ be the average query scope of all range queries (notice that an kNN query is also answered via a range query), and $Read_i$ be the number of grids of PS_i that downloaded by a range query with radius of $\overline{r}$. As the approximation of $Read_i$ for $\overline{r}$ in road networks is affected by many factors, which make the analysis very complicated. In this paper, we simplify the analysis by using the euclidean space and assume that $\overline{r} \leq \frac{w_i}{2}$. Assume that the road network in embedded in a $L \times L$ euclidean space, then for a partition strategy PS_i, the width of a grid $w_i = \frac{L}{2^i}$. For a range query with radius $\overline{r}$, Equation (2) states the expected number of grids retrieved and its proof is presented in Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2014.2330836.

$$Read_i = 1 + \frac{4\overline{r}}{w_i} + \frac{\pi r^2}{{w_i}^2}. \tag{2}$$

Let TT_i and AT_i be the tuning time and access latency of a range query with radius of $\overline{r}$ respectively, we have $TT_i = IS_i + Read_i \times GS_i$, and $AT_i = \frac{Cycle_i + Read_i \times GS_i}{2}$. Assume we use the Equation (3) to evaluate the performance of a a wireless data broadcast system, where $\alpha \in (0, 1)$ is a parameter to balance tuning time and access latency.

$$System_i = \alpha \times TT_i + (1 - \alpha)AT_i, \tag{3}$$

Let $t = 2^i$, based on the above analysis we have

$$\begin{aligned} System_i = {} & 4(2\alpha + m - m\alpha)t^4 + 6(2\alpha + m - m\alpha)t^2 \\ & + \frac{2\overline{r} \cdot DS \cdot (3 - \alpha)}{L \cdot t} + \frac{DS \cdot (3 - \alpha)}{2t^2} \\ & + \frac{L^2 \cdot DS \cdot (1 - \alpha) + \pi\overline{r}^2 \cdot DS \cdot (3 - \alpha)}{2L^2}. \end{aligned} \tag{4}$$

From Equation (4), we estimate the formulation of $System_i$, a function of i. By computing the minimum objective function value of Equation (4) and gaining the value of i accordingly, we can select the optimal grid granularity. The derivative of $System_i$ is a polynomial with power higher than 5. According to the Abel-Ruffini theorem [20], there is no general algebraic solution to polynomial equations of degree 5 or higher. So there is no a formula of the optimal i, and we present the optimal selecting result in Section 6.

6 Evaluation

In this section, we conduct extensive experiments to evaluate the performance of NPI for supporting range queries, kNN queries, and CNN queries in road networks. Some state-of-the-art road-network query processing techniques are implemented as the competitors, including RNE and INE algorithms [3], NVD algorithm [2], and NGE algorithm [1]. All the algorithms were implemented in C++, and the performance evaluation is simulated on a Genuine Intel(R) 1.80 GHz PC with 3.00G RAM, running Microsoft Windows 7 Ultimate. We first briefly describe the experimental settings, and then present the experimental results.

6.1 Experimental Setup

Two real road network datasets are used in our simulation. They are the City of Oldenburg (OL) Road Network and the California (CAL) Road Network from [21]. The OL road network contains 6,105 nodes and 7,035 edges, while CAL consists of 21,048 nodes and 21,693 edges. For each road network, a set of objects are randomly generated and uniformly distributed over the network. Even though our framework can handle objects of different sizes, for the sake of simplicity, we fix the size of an object at 128 bytes.

The evaluation is run on a simulator which consists of a server, a client and a broadcast channel. The server precomputes the distance bound matrix and the diameter of each cell. The pre-computation information along with the road network data is broadcast on the channel repeatedly. We run only one client in our simulation for simplicity, because the number of the clients will not affect the

TABLE 1
Parameter Settings

Parameter	Setting
k	1, 5, 10, 15
Query scope (d/D_N)	0.01, 0.05, 0.1, 0.2
Object density ($\|S\|/\|V\|$)	0.01, 0.05, 0.1, 0.2
Number of cells(N)	16, 64, 256

performance of a broadcast system. For each set of experiments, 400 randomly issued queries are evaluated and the average performance is reported. As mentioned before, we assume the query issuing points are always at the network nodes although our algorithms can be easily extended to support cases where queries are issued along the edges. We adopt both the access latency and tuning time as the main performance metrics. For simplicity, we assume the bandwidth is fixed and measure AT and TT in terms of number of bytes of the data transferred in the wireless channel instead of actual clock time in the client side.

In our experimental studies, we mainly consider the impacts caused by four parameters. They are the number of results k asked by kNN queries, the distance d of range queries, the object density, and the number of grid cells N. Table 1 lists their values with the underlined values standing for the default settings. Notice that D_N refers to the diameter of the road network, $|S|$ refers to the number of object set S, and $|V|$ refers to the number of nodes in the road network. Without loss of generality, we vary the value of one parameter in each set of experiments while the other three parameters are set at their defaults.

6.2 Cycle Length

First, we report the broadcast cycle length of different indexes. We use OL dataset with the object density set to 0.1. Here, a broadcast cycle consists of the index and the data with data referring to the structure of the road network and the object information. Both RNE and INE are based on Dijkstra's algorithm, so they do not maintain any special index and employ the adjacency lists of all nodes to represent the road network. NGE carries the distance vector of each node in its index, and the number of reference nodes is set to 10 that is the default setting used by [1]. NVD maintains the Voronoi cells of all the objects in the road network as its index. NPI broadcast grid partition information and some pre-computed distance information in its index. As its size is determined by the number of grid cells, we test the performance of NPI with 4^2, 4^3, 4^4 grid cells, denoted as NPI-16, NPI-64, and NPI-256 respectively. Notice that we set the number of grid cells as 4^i is to simplify the Hilbert-Curve based ordering. Except RNE and INE, we adopt $(1, m)$ scheme to generate the broadcast program with the value of m set to its optimal value [10].

TABLE 2
Broadcast Cycle Length

Method	Index size (byte)	Data size (byte)	Cycle length (byte)
RNE/INE	0	367764	367764
NGE-10	244200	367764	611964
NVD	1152045	76800	1076800
NPI-16	2196	367764	389724
NPI-64	33300	367764	700764
NPI-256	526356	367764	5631324

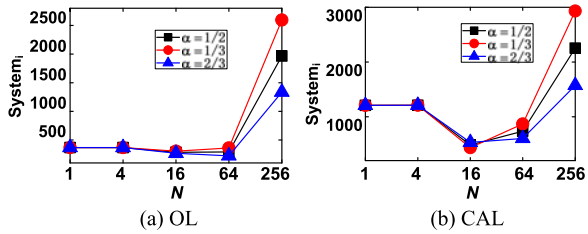


Fig. 8. Evaluation on grid granularity selection.

The experimental result is listed in Table 2. It is observed that RNE and INE have the smallest cycle, because they do not maintain any index. Among other schemes, NPI-16 has a pretty small index and its broadcast cycle length is very close to that of RNE and INE. Both NGE and NVD have relatively long broadcast cycle which is mainly caused by the large size of the index. The cycle length has a direct impact on the access latency. Our following experimental studies will further verify this.

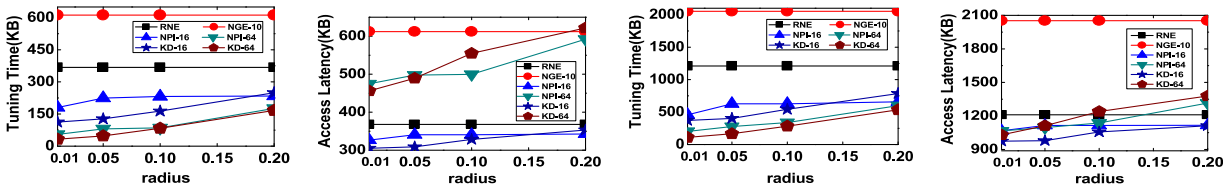
6.3 Evaluating Different Grid Granularities

In the following, we evaluate the performance of NPI with different grid granularities. The road network is partitioned into $2^i \times 2^i$ uniform grids, where i varies from 0 to 4, i.e., the number of grids ranges from 1, to 4, to 16, to 64, and to 256. The partition strategy with $i = 0$ is the same as the RNE algorithm, of which each broadcast cycle contains only the adjacency lists and the objects information of the road network. The average query scope of range queries is set as $0.1D_N$, where D_N is the diameter of the road network. Fig. 8 plots the system performance of NPI estimated by the model developed in Section 5. The x-axis corresponds to the number of grids, and the y-axis corresponds to the system performance $System_i = \alpha \times TT_i + (1 - \alpha)AT_i$. For different broadcast systems may have different preferences to tuning time and access latency, we vary the parameter α from $\frac{1}{3}$, to $\frac{1}{2}$, and to $\frac{2}{3}$. From Fig. 8 we can see that partition strategies with 16 or 64 grids gain the best performance for different α. The correctness of the model will be further demonstrated in the following experiments where we can get the real performance of NPI with different grid partition granularities.

6.4 Range Query

Next, we conduct experiments to evaluate the performance of NPI-16, NPI-64, RNE, and NGE in answering range queries. Since each range query has to explore the sub-road-network that is within d distance to the query issuing point, the number of objects would not significantly affect the performance and hence we ignore the impact of object density but fix that at 0.1. The radius of range query is varied from $0.01D_N$, to $0.05D_N$, to $0.1D_N$, and to $0.2D_N$.

The performance of different methods is plotted in Fig. 9. It can be observed from Figs. 9a and 9c that both NPI-16 and NPI-64 are superior to other methods in terms of tuning time, especially for range queries with small radius. For



(a) Tuning Time(OL) (b) Access Latency(OL) (c) Tuning Time(CAL) (d) Access Latency(CAL)

Fig. 9. Performance of range query vs. $radius$.

example, to answer range queries with $d = 0.01D_N$, NPI-64 requires only 15 percent of the tuning time of RNE for both OL and CAL road networks. Even for range queries with large radius, the advantage of our NPI is still significant.

As for the access latency as depicted in Figs. 9b and 9d, all the methods can finish the query processing within one broadcast cycle. Even though RNE has the shortest cycle, NPI-16 actually incurs a shorter access latency than RNE consistently, and NPI-64 also has a smaller access latency than RNE in some cases. This is because NPI only downloads a subset of the broadcast data, which is organized based on the Hilbert curve value and will not stretch the entire cycle.

Besides, this set of experiments gives a comparison between the performance of NPI-16 and NPI-64. NPI-16 has smaller index size, so the cycle is shorter, resulting in smaller access latency; while NPI-64 has finer grid cells, which provide a tighter upper bound and hence smaller search space for range query, resulting in shorter tuning time. In other words, when access latency is more important (e.g., $\alpha = \frac{1}{3}$), NPI-16 is expected to perform better than NPI-64; on the contrary, when tuning time is more important (e.g., $\alpha = \frac{2}{3}$), NPI-64 is expected to produce a better system performance than NPI-16. We want to highlight these findings are consistent with the observations obtained from our analytic model presented and evaluated in Sections 5 and 6.3, respectively.

6.5 kNN Query

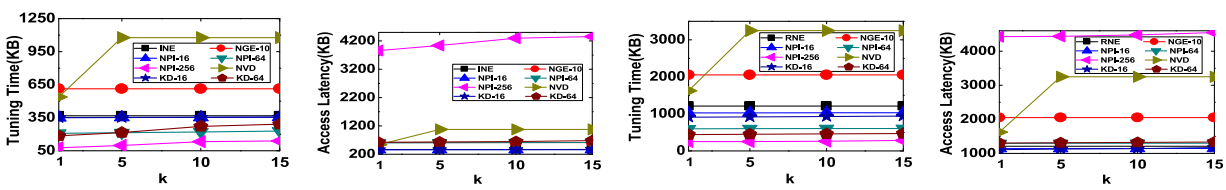
Then, we compare the performance of NPI for supporting kNN queries with existing INE (based on $Dijkstra$'s algorithm), NGE with 10 reference nodes (denoted as NGE-10), and NVD. We vary the number of NN (i.e., k), the number of grid cells, and the object density in the following experiments.

First, we evaluate the performance of different algorithms under various k values, as reported in Fig. 10. The objects density is fixed at 0.1, while k ranges from 1, to 5, to 10, and to 15. We consider only small k in our experiments because it's unpractical to display a large number of NNs on the small screen of mobile devices.

Take a look at the tuning time performance that is presented in Figs. 10a and 10c. We find that NPIs with different grid partitions outperform other method consistently, and NPI-256 performs best. This is because the pre-computation information carried by NPI enables the clients to prune the search space effectively. In contrast, clients under other schemes have to listen to the entire cycle. Notice that NVD for 1NN queries needs only listen to the first Voronoi diagram that contains the query point. However, for the kNN queries with $k > 1$, the clients have to listen to the adjacent diagrams so that the entire cycle needs to be downloaded. We also observe that the tuning time of NPIs increases as k becomes larger. This is because the upper bound of the distance between the query point and the kth NN is enlarged by k, so the clients would process range query with larger radius (see the details of kNN search algorithms in the client side in Section 4.2). However, even under a relatively large k, the tuning time of NPIs is still smaller than that of other methods. For example, for k = 15, the tuning time of NPI-64 is about 40 percent of that under INE.

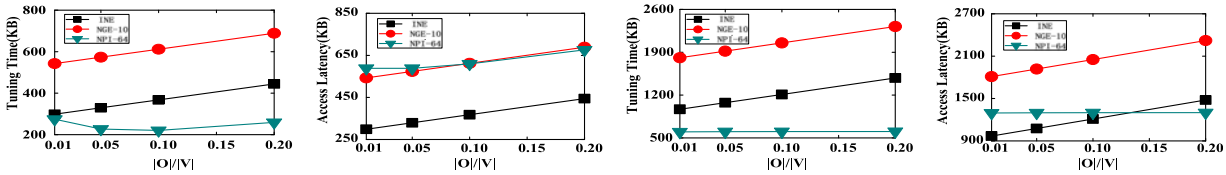
As for the access latency that is depicted in Figs. 10b and 10d, NPI-16 and INE perform the best because of the small index size. NPI-64 also demonstrates a good performance, compared with NGE-10 and NVD. NPI-256, though very energy efficient, has a large access latency because of the large index size.

Similar as previous experimental study, this set of experiments also demonstrates the the strength and weakness of NPI under different grid partitioning. NPI-256 has finer grid cells so that the diameter of each cell is smaller, which can provide a tighter upper bound of the kth NN's distance. It offers a good tuning time performance, but may force the clients to wait for a long time to retrieve all the necessary network data as the index duplicated in each cycle is very large. It is expected to provide a good system performance when tuning time is more important (e.g., large α), and the decrease of α causes its performance downgrades that is consistent with the analytic model (e.g. Fig. 8). On the other hand, both NPI-16 and NPI-64 have relatively small index size so that the access latency is smaller than that under NPI-256. However, with large grid cell, the diameter of each cell is large so that the pruning power is not that powerful which explains why their corresponding tuning time performance is relatively longer.



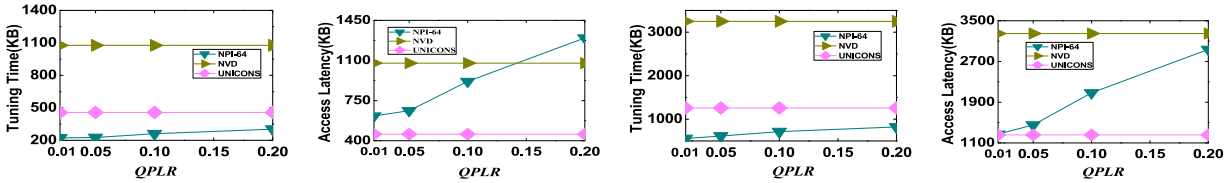
(a) Tuning Time(OL) (b) Access Latency(OL) (c) Tuning Time(CAL) (d) Access Latency(CAL)

Fig. 10. Performance of kNN queries vs. k.



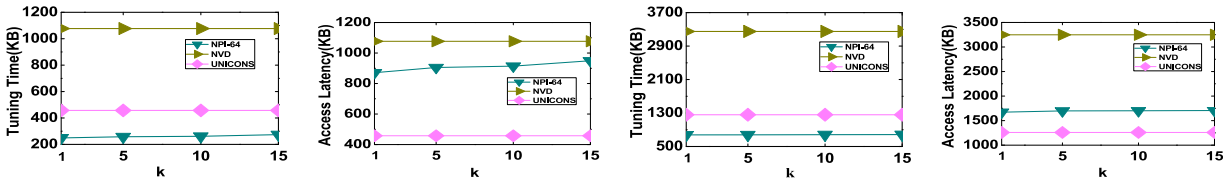
(a) Tuning Time(OL) (b) Access Latency(OL) (c) Tuning Time(CAL) (d) Access Latency(CAL)

Fig. 11. The performance of 10NN queries versus object density.



(a) Tuning Time(OL) (b) Access Latency(OL) (c) Tuning Time(CAL) (d) Access Latency(CAL)

Fig. 12. The performance of CNN queries versus $QPLR$.



(a) Tuning Time(OL) (b) Access Latency(OL) (c) Tuning Time(CAL) (d) Access Latency(CAL)

Fig. 13. The performance of CNN queries versus k.

Second, we evaluate the performance of different indexes under various object density, as reported in Fig. 11. To simplify the comparison, we assume the number of grid cells is fixed at 64 for NPI. In addition, NVD is not as competitive as others for both tuning time and access latency, and hence it is ignored in this set of experiments. It is observed that the tuning time and access latency of both INE and NGE increase as the objects become denser in the road network. This is because INE and NGE always listen to the entire cycle to process kNN queries, and the cycle becomes longer as the number of objects increases. In contrast, NPI-64 performs much more stable to the variation of the object density. This is because NPI enables the clients to check the number of objects in each grid cell to calculate the upper bound of the distance of the kth NN. When the objects density becomes denser, the upper bound decreases which prunes away more unnecessary grid cells.

6.6 CNN Query

We also conduct experiments to evaluate the CNN query performance of different approaches. The parameter $QueryPathLengthRation$ ($QPLR$) represents the ratio of the query path length to the diameter of the road network. The query path is generated by the shortest path between two randomly selected nodes in the road network.

Two approaches for CNN queries in road networks are selected as the competitors: CNVD (NVD-based CNN algorithm) [19], and the UNICONS [18], denoted as CNVD and UNICONS. CNVD is similar to NVD; while UNICONS pre-computes the 10NN of all *intersection points* (the nodes with degree larger than 2) and broadcasts the NN lists, along with the adjacency lists and the object information to the clients.

First, we evaluate the performance of different approaches under different settings of $QPLR$. The object density is fixed at 0.1 and $QPLR$ ranges from 0.02 to 0.2. Fig. 12 shows the performance for the OL and CAL datasets for answering CNN queries with $k = 10$. For CNVD and UNICONS, both the tuning time and access latency equal to the cycle length. On the other hand, NPI-64 enables space pruning, and its performance varies as $QPLR$ changes. On average, NPI-64 only requires 55 percent of tuning time of UNICONS and 24 percent of tuning time of CNVD under OL dataset, and it requires 50 percent of tuning time of UNICONS and 20 percent of tuning time of CNVD under CAL dataset. However, we also observe that NPI-64 suffers from a longer access latency than UNICONS. This is because the CNN search algorithm under NPI index allows the clients to tune into the channel to update the partial road network and kNN candidates only when they arrive at the first point of a new SVS. For each SVS, the access latency is guaranteed not exceeding one cycle. Before they move out of the current SVS, no additional access latency occurs. Considering the time cost by the clients for moving out of the current SVS, the total access latency is relatively small and will not affect the user experience.

Next, we evaluate the performance of the three approaches for supporting CNN queries for different k. We fix $QPLR$ at 0.1, and vary k from 1 to 15. The object density is also fixed at 0.1 for both OL and CAL datasets. The result is plotted in Fig. 13. NPI is more energy-efficient than CNVD and UNICONS for different k. This is because; 1) the pre-computation by NPI helps clients prune the needless regions; 2) the statements in Lemma 5 and Lemma 6 avoid the clients keeping tuning into the channel repeatedly as the clients move to a new position. On

TABLE 3
System Performances Comparison (64 Cells)

		$\alpha = 0.1$		$\alpha = 0.5$		$\alpha = 0.9$	
		NPI	kd	NPI	kd	NPI	kd
OL-R	0.05	**91.9**	122.4	**268.6**	289.2	**445.2**	455.9
	0.1	130.9	**126.9**	319.3	**292.6**	507.6	**458.1**
	0.2	**212.5**	218.0	394.3	**384.4**	576.2	**550.7**
OL-k	1	**229.2**	247.8	**404.6**	405.2	579.9	**562.6**
	5	257.3	**253.2**	425.1	**409.6**	592.8	**566.0**
	15	330.1	**267.5**	485.7	**421.4**	641.3	**575.3**
CAL-R	0.05	**260.7**	359.2	**639.4**	686.7	1018.1	**1014.2**
	0.1	**378.3**	418.3	761.1	**737.7**	1144.0	**1057.1**
	0.2	**619.8**	670.7	957.1	**957.0**	1294.4	**1243.4**
CAL-k	1	**526.9**	668.1	**872.2**	946.5	**1217.5**	1225.0
	5	**536.9**	671.7	**882.2**	950.0	**1227.5**	1228.2
	15	**556.9**	674.3	**902.2**	952.3	1247.5	**1230.3**

average, for the OL dataset, NPI consumes only 57 percent of UNICONS's tuning time, and 25 percent of CNVD's. However, NPI requires longer access latency than UNICONS in most cases.

6.7 Comparison of Different Partition Strategies

As mentioned in Section 3.1, there are different strategies to partition the road network into smaller cells. Although we adopt the grid partition as the default strategy, NPI is actually *independent* on the underlying network partitioning algorithms. In last set of experiments, we demonstrate that NPI is flexible to accommodate different partitioning strategies.

We implement kd-Tree as a representative network partitioning that generates non-uniform grid cells. Due to the space limitation, we combine the access latency and tuning time, and present in Table 3 the system performance derived based on Equation (3) with the number of cells being 64. Here, OL-R (OL-k) represents the performance of range query (kNN) under OL dataset with the radius of the range queries set to 0.05, 0.1, and 0.2, and CAL-R (CAL-k) represents the performance of range query (kNN) under CAL dataset with k set to 1, 5, and 15. For easy reference, bold numbers refer to the performances of the winner partition. There are two major observations. First, both partition strategies have their own advantages and neither of them outperforms the other absolutely. Second, the index size of grid partition is smaller than that of kd-tree partition and consequently grid partition performs better when the access latency is much more important (e.g., $\alpha = 0.1$). On the other hand, kd-tree partition favors the tuning time performance. When the tuning time plays a more important role in system performance (e.g., $\alpha = 0.9$), kd-tree performs better.

7 Conclusions

With the rapid development of wireless communication technologies and the popularity of mobile devices, the location-based services are emerging. This paper proposes an air index, namely *NPI*, to support various spatial queries in road networks in wireless broadcast environment. Algorithms for different spatial queries with *NPI* at client side are presented. The performance of *NPI* is evaluated using real road networks. The experimental result shows that our scheme is energy-efficient and access-time-friendly. This work represents our first step in developing a systematic framework to support various road network based spatial queries via wireless broadcast systems. There are a few important issues we are not able to address in this work and hopefully we can address them in our future work. First, we would like to enrich the nature of spatial queries via exploring the textual description of the objects. Ultimately, the clients can issue queries to locate objects that are physically close to the clients and semantically similar to the queries. Second, we assume the underlying wireless broadcast channel is reliable in this work and do not consider the cases where the information is lost because of network link errors. In the near future, we would like to address the issue of network link errors and to make sure our framework is efficient and meanwhile error-resilient.

References

[1] H. Kriegel, P. Kröger, P. Kunath, M. Renz, and T. Schmidt, "Proximity queries in large traffic networks," in *Proc. 15th Annu. ACM Int. Symp. Adv. Geographic Inf. Syst.*, 2007, pp. 21–28.

[2] M. Kolahdouzan and C. Shahabi, "Voronoi-based k nearest neighbor search for spatial network databases," in *Proc. 30th Int. Conf. Very Large Data Bases*, 2004, pp. 840–851.

[3] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *Proc. 29th Int. Conf. Very Large Data Bases*, 2003, pp. 802–813.

[4] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 43–54.

[5] J. Xu, B. Zheng, W. Lee, and D. Lee, "Energy efficient index for querying location-dependent data in mobile broadcast environments," in *Proc. 19th Int. Conf. Data Eng.*, 2003, pp. 239–250.

[6] B. Zheng, W. Lee, K. Lee, D. Lee, and M. Shao, "A distributed spatial index for error-prone wireless data broadcast," *Int. J. Very Large Data Bases*, vol. 18, no. 4, pp. 959–986, 2009.

[7] K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous monitoring of spatial queries in wireless broadcast environments," *IEEE Trans. Mobile Comput.*, vol. 8, no. 10, pp. 1297–1311, Oct. 2009.

[8] G. Kellaris, and K. Mouratidis, "Shortest path computation on air indexes," *Proc. VLDB* Endowment, vol. 3, nos. 1/2, pp. 747–757, 2010.

[9] Y. Jing, C. Chen, W. Sun, B. Zheng, L. Liu, and C. Tu, "Energy-efficient shortest path query processing on air," in *Proc. 19th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2011, pp. 393–396.

[10] T. Imielinski, S. Viswanathan, and B. Badrinath, "Data on air: Organization and access," *IEEE Trans. Knowl. Data Eng.*, vol. 9, no. 3, pp. 353–372, May/Jun. 1997.

[11] E. Jung and N. Vaidya, "An energy efficient MAC protocol for wireless lans," in *Proc. IEEE Conf. Comput. Commun.*, 2002, pp. 1756–1764.

[12] S. Hambrusch, C. Liu, W. Aref, and S. Prabhakar, "Query processing in broadcasted spatial index trees," in *Proc. Int. Symp. Adv. Spatial Temporal Databases*, pp. 502–521, 2001.

[13] B. Zheng, J. Xu, W. Lee, and L. Lee, "Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services," *Int. J. Very Large Data Bases*, vol. 15, no. 1, pp. 21–39, 2006.

[14] B. Zheng, W. Lee, and D. Lee, "Spatial queries in wireless broadcast systems," *Wireless Netw.*, vol. 10, no. 6, pp. 723–736, 2004.

[15] H. Hu, D. Lee, and V. Lee, "Distance indexing on road networks," in *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, pp. 894–905.

[16] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[17] R. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm, "Partitioning graphs to speed up dijkstras algorithm," *J. Exp. Algorithms*, vol. 11, pp. 273–283, 2005.

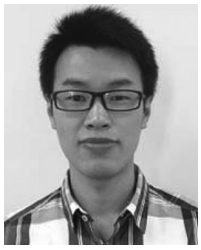
[18] H. Cho and C. Chung, "An efficient and scalable approach to CNN queries in a road network," in *Proc. 31st Int. Conf. Very Large Data Bases*, 2005, pp. 865–876.

[19] M. Kolahdouzan and C. Shahabi, "Continuous k-nearest neighbor queries in spatial network databases," in *Proc. Spatio-Temporal Databases Manage.*, 2004, pp. 33–40.
[20] N. Jacobson, *Basic Algebra I*. New York, NY, USA: Dover, 2012.
[21] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng, "On trip planning queries in spatial databases," in *Proc. 9th Int. Conf. Adv. Spatial Temporal Databases*, 2005, pp. 923–923.

Weiwei Sun received the bachelor's degree in computer science and technology in 1992, and the master's and PhD degrees in computer software and theory from Fudan University, China, in 1998 and 2002, respectively. He is currently an associate professor in the School of Computer Science and the director of Mobile Data Management Laboratory, Fudan University. His interests include spatial database, wireless data broadcast, and geo-social.

Chunan Chen received the bachelor's degree in information security in 2010 and the master's degree in computer software and theory in 2013 from Fudan University. His research interests include wireless data broadcast and spatial data management.

Baihua Zheng received the bachelor's degree in computer science from Zhejiang University, China, in 1999, and the PhD degree in computer science from the Hong Kong University of Science and Technology, Hong Kong, in 2003. She is currently an associate professor in the School of Information Systems, Singapore Management University, Singapore. Her research interests include mobile and pervasive computing and spatial databases. She is a member of the IEEE and the ACM.



Chong Chne received the bachelor's degree in computer science and technology from Fudan University in 2012, where he is currently working toward the master's degree in computer software and theory. His interest includes spatial database.

Peng Liu received the bachelor's degree in computer science and technology in 2008 and the master's degree in computer software and theory in 2013 from Fudan University. His research interests include wireless and xml data broadcast.