Information Systems 42 (2014) 177–194

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Searching continuous nearest neighbors in road networks on the air

Yanhong Li [a], Jianjun Li [b,*], LihChyun Shu [c,d], Qing Li [e], Guohui Li [b], Fumin Yang [b]

[a] College of Computer Science, South-Central University for Nationalities, Wuhan, China
[b] School of Computer Science and Technology, Huazhong University of Science Technology, Wuhan, China
[c] Department of Accountancy, National Cheng Kung University, Taiwan, ROC
[d] College of Information and Engineering, Chang Jung Christian University, Taiwan, ROC
[e] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 28 April 2013
Received in revised form
8 January 2014
Accepted 10 January 2014
Recommended by: F. Korn
Available online 22 January 2014

Keywords:
CNN queries
Wireless broadcast
CN^3B
Road networks

ABSTRACT

Recently, people have begun to deal with location-based queries (LBQs) under broadcast environments. To the best of our knowledge, most of the existing broadcast-based LBQ methods are aimed at Euclidean spaces and cannot be readily extended to road networks. This paper takes the first step toward processing Continuous Nearest Neighbor queries in road Networks under wireless Broadcast environments (CN^3B). Our method leverages the key properties of Network Voronoi Diagram (NVD). We first present an efficient method to partition the NVD structure of the underlying road networks into a set of grid cells and number the grid cells obtained, based on which we further propose an NVD-based Distributed air Index (NVD-DI) to support CN^3B query processing. Finally, we propose an efficient algorithm on the client side to process CN^3B queries. Extensive simulation experiments have been conducted to demonstrate the efficiency of our approach. The results show that our proposed method is about 4 and 7.6 times more efficient than a less-sophisticated D-tree air index based method, in access latency and tuning time, respectively.

1. Introduction

With the rapid development of wireless technology and ever growing popularity of smart mobile devices, location-based services (LBSs) have become popular over the past few years. Being one of the enabling technologies for LBS, location-based queries (LBQs) have been a hot research topic. An important class of LBQs is the continuous nearest neighbor (CNN) query, which continuously finds the objects nearest to a query client while the client keeps moving from one place to another. For example, "continuously finding the nearest gas station along the path on which a car moves". LBQs return results based on certain location information. Generally speaking, there are two approaches to accessing location information via wireless technology: on-demand access and periodic broadcast [1]. On-demand access employs a basic client–server model, where a mobile client (MC) submits a query to the server, which is responsible for processing a query and returning the result to the client via the dedicated point-to-point channel. This method is suitable for light-loaded systems where contention for transmission channels and server resources is not severe. However, this on-demand access method has three critical drawbacks: (1) the server processing ability and the uplink bandwidth will be the bottleneck of the system, which jeopardizes the scalability of the system; (2) the server has to process a large number of query requests simultaneously, which will affect the timeliness of the query results; and (3) it fails to exploit the similarity of content desired by all users.

* Corresponding author. Tel.: +86 27 87543104.
E-mail address: jianjunli@hust.edu.cn (Y. Li).

0306-4379/$ - see front matter © 2014 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2014.01.003

On the contrary, periodic broadcast requires the server to actively push data to the clients via a broadcast channel. Without sending any request to the server, each client simply listens to the broadcast channel to retrieve data based on its own query and processes the data retrieved to get the query result autonomously. Periodic broadcast communication provides more efficient data delivery as the broadcast data can be simultaneously accessed by an arbitrary number of MCs, and hence is especially suitable for heavy-loaded systems. For example, "a large number of people from the four corners of the world attend a large-scale event (e.g., Olympic games, art festivals, or World Cups). After the event, people would like to query the nearest hotel (or restaurant) when they are leaving the activity spot. Considering that there are so many people from all over the world and they are not acquainted with and have not stored the map of this city, it would be necessary for them to request the answer through the wireless data broadcast approach". In fact, wireless data broadcast is a matured technology and the relevant services have been available as commercial products for several years, such as StarBand and MSN Direct Service. MSN Direct Service is based on the smart personal objects technology (SPOT) and the DirectBand Network. By this service, mobile clients can continuously receive timely information such as stock quotes, airline schedules, news stories, weather, and traffic information.

Recently, researchers have begun to zero in on processing LBQs using periodic broadcast. However, to our knowledge, most of the existing broadcast-based LBQ methods are limited to Euclidean space, and few of them can be used in network space. But in real life situations, MCs always move within a certain network, such as a road network or a railway network. In a road network, the distance between objects is determined by the connectivity of the network, rather than the objects' coordinates in Euclidean space. Thus, existing LBQ methods used in Euclidean space cannot give accurate query results, and it is therefore essential to examine broadcast-based LBQ methods suitable for use in real road networks.

In this paper, we concentrate on processing Continuous Nearest Neighbor queries in road Networks under data Broadcast environments (abbreviated as CN^3B in the following). In the periodic broadcast method, the structure of a road network and the objects in it are broadcast periodically via a broadcast channel, and MCs are responsible for query processing. Since a road network is in a two-dimensional space whereas data sent in a broadcast channel is a linear sequence, a major challenge for CN^3B query processing is how to partition the underlying network and organize the partitioned network together with its objects into a set of sequential packets to be broadcast sequentially. Moreover, since MCs are usually energy-constrained, while an air index makes it possible for MCs to listen selectively to the desired data instances in the wireless broadcast channel and hence reduce energy consumption [11], how to design efficient air index is another important problem that must be addressed for CN^3B query processing. Moreover, the added air index will enlarge the total amount of bits in a broadcast cycle, which in turn will increase the access latency. Thus, a good air index should consider the trade-off between access latency and selective tuning ability, and greatly cuts down the energy consumption with limited access latency prolongation. In this paper, by fully utilizing the properties of Network Voronoi Diagram (NVD), we propose a novel NVD constructing and partitioning scheme, based on which we further propose an efficient distributed index to support CN^3B query processing. To fulfill the entire CN^3B query processing, we also present an efficient algorithm on the client side. The main contributions of this paper can be summarized as follows:

- We propose an efficient method to construct and partition the NVD structure of the underlying road networks to derive an NVD quad tree, and then transfer the tree into a linear sequence of data packets, with each data packet corresponding to one grid cell, which has a unique ID associated with it, in the road network. In particular, the grid cells obtained by our method are generally balanced in size and preserve good locality behavior.
- Based on the partition result derived above, we propose an NVD-based distributed index (namely NVD-DI) to support CN^3B query processing. NVD-DI exhibits the following properties: (i) it allows a CNN search to start its execution at arbitrary time instant; (ii) each search can be finished within one broadcast cycle; and (iii) it can significantly reduce the energy consumption at the expense of limited access time prolongation.
- Combining with the broadcasting scheme on the server side, we design an efficient algorithm on the client side to process CN^3B queries. We evaluate the performance of the proposed CN^3B query processing method via extensive experiments on both a real road network and a synthetic 2-dimensional grid network, and the experimental results show that our method outperforms the D-tree [26] air index based method significantly.

The remainder of this paper is structured as follows. Section 2 reviews related work. In Section 3, we first present a straightforward D-Tree based scheme for CN^3B query processing, and then propose our novel NVD-DI based method. Section 4 reports the performance evaluation of the proposed methods. Finally, we conclude the paper with a brief discussion of future work in Section 5.

2. Related work

Nearest neighbor search, one of the important research issues in location-based queries, has been studied over the last two decades, and several NN search methods [4–6,15,22,25] have been proposed. In this section, we briefly review some existing work on processing NN queries in road networks and data broadcast algorithms for NN queries in Euclidean space.

2.1. Nearest neighbor queries in road networks

Jensen et al. [12] first tackled the problem of NN search in road networks in the year 2003. Then, Papadias et al. [19] presented an architecture that integrates network and

Euclidean information to process static network-based queries. They proposed two methods, IER and INE. The efficiency of INE depends on the density of objects. If the objects are sparsely distributed in the road network, the method can be inefficient. To avoid on-line distance computation in processing kNN queries, Kolahdouzan et al. [13] proposed the VN^3 method, which is based on the network Voronoi diagrams. The VN^3 can be efficiently used for networks that have a low density. However, in high density networks, its efficiency can be low since it includes operations to construct, store and inquire a large number of Voronoi polygons to answer kNN queries. Huang et al. [9] proposed the Islands approach which consists of a pre-computation component and an online network expansion component. Since the Islands approach can control the sizes of the islands, it offers flexibility in balancing the amount of pre-computation data, the cost of updating the pre-computed data, and the efficiency of the kNN queries.

Kolahdouzan et al. [14] proposed two methods, IE and UBA, for processing continuous k-nearest neighbor queries in spatial network databases. IE is based on examining the kNNs of all the nodes on a given path and the split points between adjacent nodes whose nearest neighbors are different. UBA eliminates the kNN computation for the nodes that cannot have any split points in between. Cho et al. [3] solved the same problem by introducing UNICONS, which combines the pre-computed kNN lists with Dijkstra's algorithm, thus it outperforms IE/UBA in high-density networks. Safar et al. [23] proposed a PINE based method to process CNN queries in road networks. To efficiently find the location of the split point(s) on the path, they used a modified version of the IE algorithm proposed in [14]. However these three methods are just to determine the kNNs of any point on a given path and they assume the data objects are static.

Mouratidis et al. [17] addressed the issue of processing CkNN queries in road networks by proposing two algorithms (namely, IMA/GMA) that handle arbitrary object and query movement patterns in the road network. They can process kNN monitoring over moving objects and query points and re-calculate query results whenever an update occurs. However due to the nature of discrete location updates, the kNNs of the query point within two successive updates are unknown. Thus, these methods would return invalid results between two successive update time stamps. Huang et al. [10] proposed a continuous monitoring method for moving objects in road networks. This method includes the pruning phase and the refining phase, and it can continuously give the kNN result of query q. However, the objects in question are assumed to move with constant speeds.

2.2. Data broadcast algorithms for nearest neighbor search in Euclidean space

A straightforward method to NN search in broadcast environments 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 channel. Obviously, this approach is inefficient in terms of latency and tuning time since all the data points should be examined. There are generally two kinds of methods to address this issue: (1) using an index in the broadcast to improve the performance on the access latency and tuning time; (2) instead of using an index, assigning all the data points to the packets in the broadcast in some order and/or add some additional information in each site in the broadcast.

In [27], Zheng et al. adapted the R-tree index [7] to search CNN in broadcast environments. Additionally, they presented a CNN search algorithm based on the Hilbert-Curve air index. However, since the air index is located at the beginning of each broadcast cycle, the clients have to wait until the beginning of the next broadcast cycle in order to obtain the index segment. Thus, it may incur longer access latency. Hambrusch et al. [8] presented techniques for scheduling a spatial index tree for broadcast in single and double channel environments. The algorithms executed by clients aim to minimize latency and tuning time. However, the algorithms may still incur longer access latency due to the same reason as that of the Hilbert Curve air index. In [21], the authors proposed a new broadcast-based NN processing method. In this method, broadcast data objects are sorted and broadcast sequentially based on their locations. Thus, it is not necessary for the client to wait for an index segment, if it has already identified the desired data items based on the broadcasting order before the associated index segment has arrived. Liu et al. [16] proposed efficient protocols for kNN search using a broadcast R-tree. By adding some additional entries to the index nodes of the R-tree, the method allows the kNN search to start in the middle of a broadcast cycle, thereby reducing the access latency. In [28], the authors developed a generalized search algorithm for CkNNs based on the Hilbert–Curve index in wireless broadcast systems. This is the first study on this issue. Zheng et al. [29] presented a distributed spatial index to support efficient location-based data access in wireless data broadcast systems. This index has a linear yet fully distributed structure that naturally shares links in different search paths and it is very resilient to error-prone wireless communication environment. However, data objects in the above methods are assumed static. Park et al. [20] proposed a client–server architecture for answering CNN queries in wireless broadcast environments, and this method does not require any conventional spatial index. Besides, it can be adapted to handle static or moving objects, and does not require additional information beyond the maximum speed and the location of each object.

However, these previous studies have at least one of the following limitations: (1) they focus on point-to-point communication and do not consider wireless data broadcast environment; (2) they are limited to Euclidean space and cannot handle spatial queries in the road network which is a more realistic scenario. Thus, it is important to devise an efficient method to deal with CNN queries in the road network under data broadcast environments.

3. CN^3B query processing

Fig. 1 gives an illustration of a CN^3B query, where query q moves from q_{start} to q_{end} along a given road namely *query*

line. CN^{3}B query requires that its query result should be updated timely when q keeps moving along its query line. Broadcast-based CNN query processing in real road networks (i.e., CN^{3}B) is different from and more difficult than that in Euclidean space. This is mainly because in road networks, the distance between a query point q and a data point o is determined by the connectivity of the network, as opposed to the positions of q and o in Euclidean space, which are comparatively easy to obtain. Moreover, since most mobile clients (MC) keep moving from time to time on the road network, it is unrealistic to require an MC to store the structure of the road network and the positions of the objects in the road network before the MC launches a query. In other words, the information of both the road network and the data objects being searched have to be broadcast by the server. In view of this, how to organize and broadcast the information on the server side, as well as design efficient algorithms to process queries on the client side, are the major issues that must be addressed for CN^{3}B query processing.

As well known, Voronoi Diagram (VD) is a useful structure for CNN query processing. A Voronoi Diagram (VD) divides a spatial space into several disjoint Voronoi Polygons (VP) where the nearest neighbor of any point inside a VP is the generator point (object) of the VP [18]. NVD is a specialization of VD, where the location of objects is restricted to the edges of the graph and the distance between objects is defined as the shortest path connecting them in the network instead of their Euclidean distance [18]. For a given generator object p in an NVD, its VP includes all the edges (or sub-edges), where all the points on these edges are closer to p than any other generator objects in the network. In this section, we first introduce a straightforward method which uses NVD directly, and point out its inefficiency on CN^{3}B query processing in Section 3.1, then we present our novel method in Section 3.2.

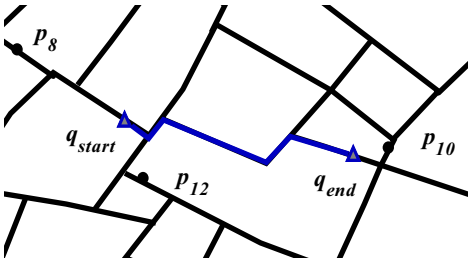

Fig. 1. An illustration of CN^{3}B query in a road network.

3.1. A straightforward method

Given a road network, a common method for processing CNN queries consists of the following three steps:

(1) Construct the NVD diagram based on the objects distributed in the road network.
(2) Utilize a D-Tree [26] to index the VPs of the NVD constructed.
(3) Repeatedly search the D-Tree to locate the VP which contains the query point when it keeps moving along the road network, and the generator object of the VP is the query result.

We call the above D-Tree based Method DTM in the *sequel*. Now we give a concrete example to show how DTM works. Fig. 2(a) depicts a road network, where p_1, p_2, p_3 and p_4 are the data objects and the hollow points are the intersections of the road network. Fig. 2(b) shows the NVD constructed based on the objects distributed in the road network. As can be seen, the green lines divide the whole road network into four VPs (VP_1 to VP_4). Note here b_i ($1 \le i \le 8$) is a border point of the VPs, while v is an auxiliary point which connects adjacent border points. To construct the D-tree index, we can use a large cell to confine the whole NVD and define this cell as the root node of the D-tree. The NVD is recursively partitioned into two sub-spaces until each space contains only one region (VP). As shown in Fig. 2(b), we first partition the whole NVD into two sub-spaces, VP_5 and VP_6, by the polyline $b_1b_2vb_7b_8$. Then VP_5 (VP_6, resp.) is further partitioned into VP_1 (VP_2, resp.) and VP_3 (VP_4, resp.), by the polyline b_3b_4v (vb_5b_6, resp.). In this way, we can get the D-tree index as shown in Fig. 2(c). Next, we can linearize the D-tree index to get an air index, in particular, the air index can be constructed by traversing (starting from the root node) the D-Tree in a level-based order. Notice here for each VP, the pointer always points to their child VPs. After deriving the air index, we can simply interleave it with NVD diagram and data objects on the broadcast channel and then broadcast them. For instance, for the above example, we can derive the air index and the broadcast cycle of the

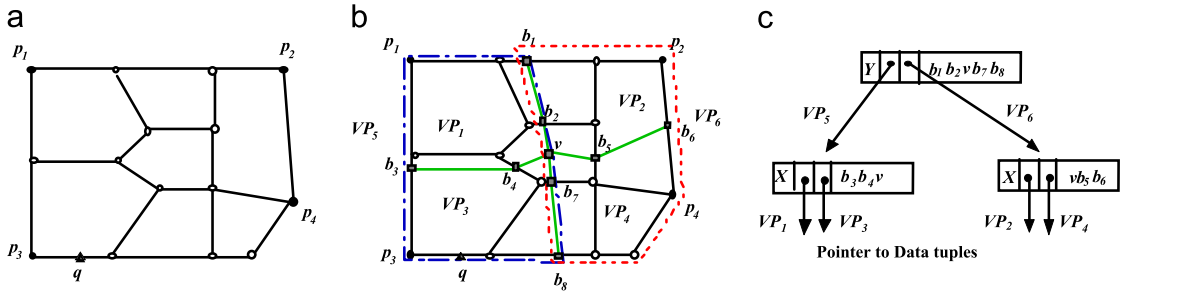

Fig. 2. An example of constructing a D-tree index from a given NVD that depicts a road network. (a) A road network. (b) NUD. (c) D-tree index. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

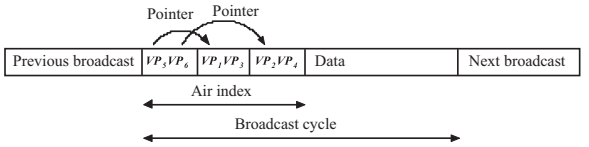

Fig. 3. The air index and the broadcast cycle of the NVD in Fig. 2.

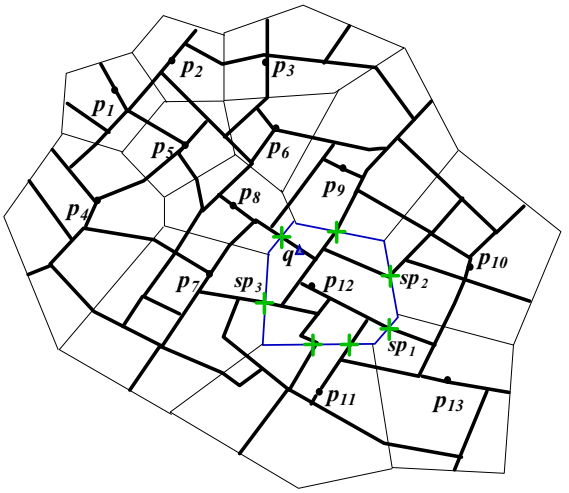

Fig. 4. A road network and the corresponding NVD. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

NVD, as shown in Fig. 3. Till now, we have finished all the issues on the server side, on the client side, the MC can repeatedly tune into the broadcast channel and search the air index to locate the VP which contains the query point when it keeps moving along the road network, and the generator object of the VP is the query result.

Drawback of DTM: Although DTM is an effective scheme for CN^3B query processing, it is not an efficient one. This is because in DTM, executing an NN/CNN search has to start from the root of the D-Tree (i.e., the beginning of a broadcast cycle), which may lead to long access latency. Moreover, since the VPs of NVD are irregular polygons, the partition size of D-Tree is large, which in turn can enlarge the size of the D-Tree index, and this may further result in long access latency and tuning time. To determine which VP a query point belongs to, one would need some reference polylines. For example, to find out that the query q belongs to VP_3, we need two polylines, viz. $b_1b_2vb_7b_8$ and b_3b_4v. However, these two lines cannot be obtained without searching the D-tree index (Fig. 2(c)) that is broadcast from the server. Thus the client has to continuously listen to the broadcast channel to get the index information of the frames which contain the VPs covered by the query line of q. As a result, DTM is not efficient for processing spatial queries in data broadcast environments, which will also be demonstrated in our experimental study in Section 4.

3.2. Our novel method

The inefficiency of DTM mainly steps from that it depends on a tree-based air index. In this section, by fully leveraging the key properties of NVD, we propose a novel method for CN^3B query processing. Firstly, in Section 3.2.1, we construct the *NVD structure* of the underlying road network according to the positions of the data objects in it. Then in Section 3.2.2, we propose an efficient quad partition method to divide the NVD structure into several rectangular areas (called grid cells) and obtain a quad-tree to keep the grid cells of the NVD structure. Thirdly, we propose an NVD-based distributed index, namely *NVD-DI*, to support CN^3B query processing in Section 3.2.3. In Section 3.2.4, we introduce an efficient algorithm on the client side to process CN^3B queries. Finally, we present an analytical model in Section 3.2.5 to study the performance of our proposed query processing method.

3.2.1. NVD structure construction

Figure 4 presents a road network and its corresponding NVD diagram. In this figure, the road network is divided into several VPs and each VP includes a data point (object), which is the generator of the VP. As can be seen, q is located in the VP of p_{12} (bounded by blue lines), hence it is easy to conclude that the *NN* of query q is p_{12}.

Since the query clients are constrained to move on the roads (also called edges) of the underlying network, the actual boundaries between adjacent VPs are some special points, called *split points*. As can be seen from Fig. 4, sp_1, sp_2, and sp_3, each represented by a green cross, are split points. A split point cuts the edge where it lies into two parts, each of which belongs to one of the VPs of two different objects. For example, sp_1 cuts the edge it locates into two edges which belong to the VPs of p_{12} and p_{13}, respectively. Thus, an edge e may belong to several VPs of different objects. We mark each edge of the road network with an attribute *obj_dom* to indicate that this edge belongs to the VP of a certain object. Hereinafter, we call this marked road network which keeps NVD properties *NVD structure*. Note *NVD structure* is different from *NVD diagram*. In particular, An *NVD diagram* consists of a set of Voronoi polygons with different shapes and sizes, which make it hard to be divided evenly in the space partition step. On the contrary, an *NVD structure* consists of a set of edges with the same sizes, and thus can be evenly divided easily. The pseudo code for building an *NVD structure* is given in Algorithm 1. It is not difficult to see that Algorithm 1 has a linear time complexity $O(|S|)$, where $|S|$ is the number of objects in set S.

3.2.2. NVD partition and coding

We now discuss how to partition the *NVD structure* constructed above. There are many space partition methods, such as fixed partition, which divides the search space into grid cells with fixed spatial size, and adaptive partition, which divides the search space into grid cells such that each grid cell contains nearly the same number of data objects. In this subsection, we propose an efficient approach, namely *NVD quad partition*, to partition the *NVD structure* constructed so as to derive an *NVD quad tree*.

Algorithm 1. Building NVD structure.

input : The road network structure and the object set S
output: The NVD structure
1 **for** *each object* $p \in S$ **do**
2 use a Dijkstra-like algorithm to search the road network to find any one of its neighboring objects p', and calculate the central point o between p and p';
3 set the obj_dom of the edges on the route from p (p', resp.) to o with p (p', resp.);
4 **if** *all edges have been marked* **then**
5 exit the loop;

Considering that each broadcast frame is of fixed size, the basic idea of our NVD partition method is to make each transmitted unit in the broadcast channel contain approximately equal amount of information about the underlying road network. Before giving the detail of the partition method, we first introduce a threshold λ that will be used later.

Definition 1. Let s_F be the size of a broadcast frame, s_i be the size of an index table, and s_e be the space size needed for keeping an edge. λ is decided by s_F, s_i and s_e, where $\lambda = (s_F - s_i)/s_e$.

Now we proceed to detail our partition method. At first, we use a large cell C to confine the whole NVD structure and define C as the root node of the NVD quad-tree. If the number of edges overlapping with cell C exceeds the predefined threshold λ, C is split into four sub-cells, 00, 01, 10, and 11, each having the same size, and the sub-cells become the child nodes of C. Besides, the edges of C are distributed into these four sub-cells according to their positions in cell C. In particular, 00, 01, 10, and 11 represent the sub-cells at the upper-left, upper-right, lower-left, and lower-right side, respectively. Note that different parts of a single edge in C can be distributed into different sub-cells of C. For each edge e of C, if the entire or part of edge e overlaps with the sub-cell ij (i, j=0 or 1), we add 1 to the number of edges in cell ij. After all the edges of C have been processed, the numbers of edges in cell ij can be readily obtained.

The above process repeats until every grid cell contains no more than λ edges, and then the quad tree rooted at C is obtained. For ease of query processing at the client side, each node (grid cell) in the quad-tree is assigned an identification (ID) which is a sequence of 0–1 bits, except for the root node C, which has an empty string as its ID. For a non-root node, its ID is obtained by concatenating its parent node's ID with one of the four two-bits 00, 01, 10 and 11, which is determined by the relative position of the node as described in the preceding paragraph. Hereinafter, we call this ID the *GCode* of the grid cell. It is not difficult to see that the length of the *GCode* of a grid cell is twice the level of the cell in the quad-tree. With this naming scheme, we know that for a given NVD, there is a one-to-one relationship between the grid cell and its *GCode*. Hence, given a *GCode*, the area that the grid cell covers can be readily figured out, and, given a space point, the *GCode* of the cell containing the point can also be immediately determined. Moreover, since geographically neighboring cells are also adjacent in their cell numbers, the grid cells of the NVD quad-tree obtained have good locality-preserving behavior. Now let us go back to the running example illustrated in Fig. 4. With our NVD partitioning and naming schemes, the NVD structure is partitioned into multiple numbered grid cells where solid dots represent data objects and rectangles in dotted line represent grid cells, as shown in Fig. 5, and the corresponding NVD quad-tree is given in Fig. 6.

The next step is to organize the NVD quad-tree into the broadcast channel. To ensure that geographically neighboring cells are adjacent in the broadcast channel, we put the leaves of the NVD quad tree into the channel from left to right. In this way, we can obtain the linear sequence of the 31 grid cells for the NVD quad-tree constructed above, as shown in Fig. 7. Note that the linear sequence obtained is in an ascending order of the *GCode* value. It is worth mentioning that since some areas (grid cells) have fewer edges, the edges within a road network are usually unevenly distributed. In order to increase the storage utilization, we merge consecutive cells to form a new one if the sum of the number of edges in these cells remains no larger than λ. Note that if two or more cells are merged, then the first cell's *GCode* will be used as the *GCode* of the merged cell. We again use the above example to illustrate how to conduct the merge operation. Assume

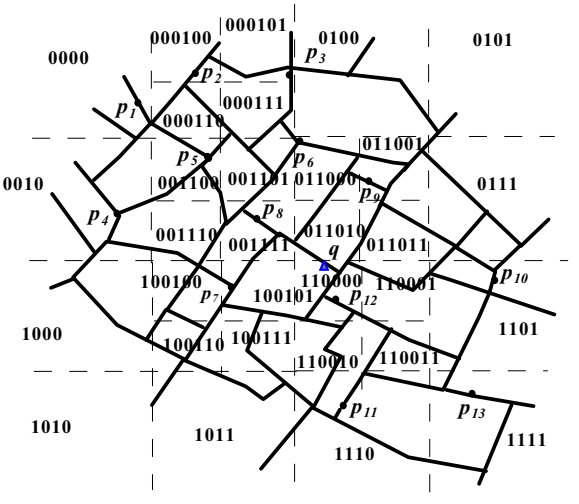

Fig. 5. The grid cells after partition of the NVD structure in Fig. 4.

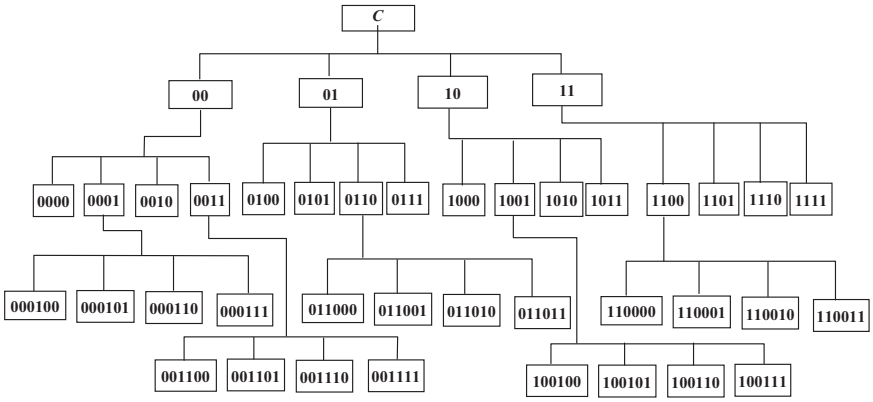

Fig. 6. The constructed NVD quad-tree corresponding to Fig. 5.

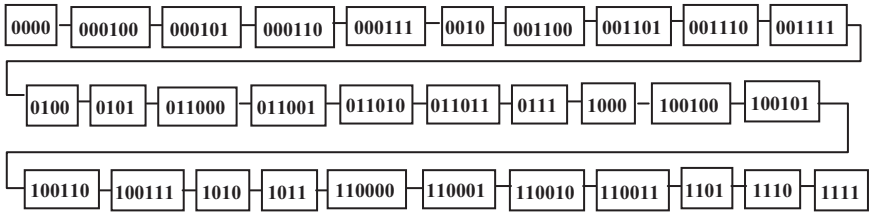

Fig. 7. The linear sequence of the NVD quad-tree in Fig. 5.

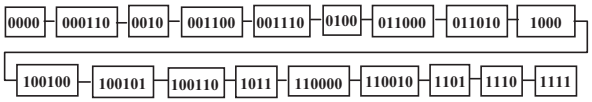

Fig. 8. The linear-compressed sequence of the NVD quad-tree in Fig. 5.

that $\lambda = 10$, since the sum of the number of edges in the continuous three cells 0000, 000100 and 000101 is no larger than 10, these three cells can be merged to form a new cell 0000. Similarly, the cells 0100 and 0101 can be combined to form a new one 0100. After conducting the merging process, we can obtain a new linear-compressed sequence which consists of 18 cells, as shown in Fig. 8.

After the above merging process, some grid cells may contain several smaller grid cells. Moreover, since the grid cells in our system belong to different level of the NVD quad-tree, the grid cells which are not located at the lowest-level of the tree contain several lowest-level grid cells. For ease of processing, for each grid cell, we keep the maximum cell code of the lowest-level cells it contains, namely *MaxGCode*. For example, cell 0000 consists of three cells 0000, 000100 and 000101, so its *MaxGCode* is 000101.

Now we discuss the time complexity of NVD partition and coding. In the first round of partition, each edge will be checked once to determine which sub-cell it belongs to, so the time complexity for this round is $O(r)$, where r is the number of edges in the road network. For a road network whose edges are evenly distributed, it takes $\log_2(r)$ partition round at most to get the final partition result, hence the time complexity is $O(r \times \log_2(r))$. However, for extreme situation where the edges in a road network are pretty concentrated, it may take r round partitions in the worst case, thus the time complexity is $O(r^2)$. The cost for assigning an ID to each grid cell and getting the liner compressed sequence of grid cells are both $O(K)$, where K is the number of grid cells to be processed. In sum, the time complexity of this stage is $O(r^2 + K)$.

3.2.3. NVD-DI index construction

Since the clients do not know in advance when the grid cells they require will arrive, they have to listen to the broadcast channel continuously and keep retrieving all the grid cells being broadcast until the desired ones are located. Obviously, this behavior consumes quite a long tuning time, which in turn can result in large power consumption of mobile clients. For an ordered linear sequence to be broadcast, in order to realize energy-efficient selective tuning, distributed index has been widely accepted as a good choice. Based on the NVD quad-tree constructed above, we present a linear distributed index namely NVD-DI in this section. With NVD-DI, the execution of a CNN search can be started whenever an MC tunes in the broadcast channel. Moreover, it allows clients to keep in sleep mode until the desired frame(s) and the relevant index frame(s) arrive, which can significantly save energy consumption.

Assume that each broadcast frame contains only one grid cell and the total number of grid cells is N. In addition to the information of a grid cell, each frame also has the following information as its header: (1) the *GCode* and *MaxGCode* of this grid cell; (2) a forward Index Table (IT) which keeps n entries in the form of $\langle i, GCode_i \rangle$, where $i \in [0, n-1]$ and $GCode_i$ represents the *GCode* of the grid cell corresponding to the frame which is the 2^ith frame following the current frame in the broadcast cycle. More clearly, the IT contains the *GCodes* of the grid cells corresponding to the 1th, 2th, 4th, …, and 2^{n-1}th frame following the current frame. With this definition, it is not difficult to see that $n = \lceil \log_2(N) \rceil$. Note that the frame size,

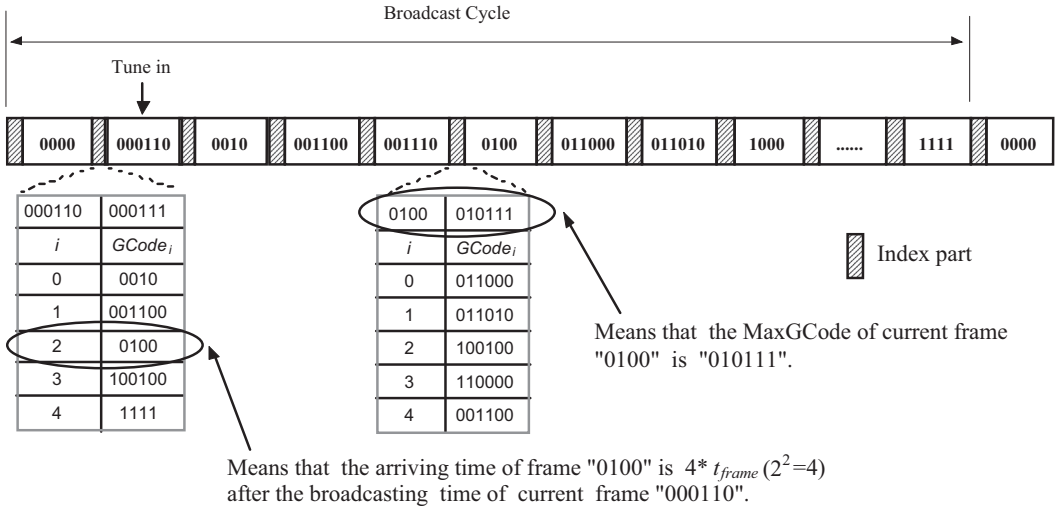

Fig. 9. The NVD-DI structure of the network shown in Fig. 4.

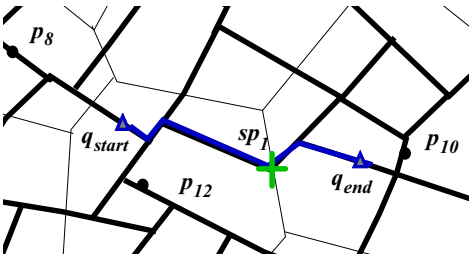

Fig. 10. An example of a CN^{3}B query crossing the boundary of two neighboring VPs.

as well as the time needed to broadcast each frame (called t_{frame}) are both fixed. Hence, given the broadcast order of a frame, its arrival time can be readily obtained. To help an MC calculate the *GCode* of the grid cell where it is located, we define three parameters H, W, and h for each frame, where H and W are the height and width of the road network, respectively, and h is the height of the NVD quad-tree. Thus, for a linear-compressed sequence derived from an NVD quad-tree, the broadcast cycle consists of several frames, where each frame includes the following items sequentially: (1) three parameters H, W, and h; (2) the corresponding index table; and (3) the edge and object information included by the corresponding grid cell. The three organized items are delivered by the server sequentially, and the mobile client can distinguish the bit sequence between the index part and the data part. This is because the three parameters and the index table are placed at the beginning of each frame, and moreover, their sizes are fixed.

Now let us go back to the example given in Fig. 4, from which we obtain the NVD-DI index. Fig. 9 shows the index tables which corresponds to the frames of grid cells 000110 and 0100. Since $N=18$, each index table has $\lceil \log_2(N) \rceil = 5$ entries. Take the index table for frame 000110 as an example, the first entry $\langle 0, 0010 \rangle$ means that the first (2^0th) frame following the current frame has $GCode=0010$, and its arrival time is $1 \times t_{frame}$ after the current frame. The remaining entries can be interpreted in a similar way.

Since we need to construct an IT table for each frame in the broadcast cycle, it is no difficult to see that the time complexity of this step is O(N), where N is the number of frames (grid cells) in a broadcast cycle.

3.2.4. CN^{3}B query processing algorithm on the client side

In this section, we present an efficient algorithm to process CN^{3}B queries on the client side. Since a client q may move continuously within a road network, it makes the retrieval of q's NN a challenge. A CNN query finds the NN object to every point on a given query line.[1] As discussed before, we only report the NN's status change when query q arrives at a split point, as opposed to continuously issuing NN queries when q keeps moving. Fig. 10 gives an example where the query line of q crosses the Voronoi Polygons of objects p_{12} and p_{10}. The split point sp_1, which is the intersection of the query line of q and the boundary of the VPs of p_{12} and p_{10}, cuts the query line into two line segments $q_{start}sp_1$ and sp_1q_{end}, with their NNs being p_{12} and p_{10}, respectively.

We now detail our CN^{3}B query processing algorithm on the client side, denoted by Client. The pseudo-code of Client is presented in Algorithm 2. Table 1 lists the formal definitions off some symbols that will be frequently used later. Given a query line of q, Client performs the following three steps:

(1) Obtain a list of *GCodes* of the grid cells overlapping with the query line (Algorithm 3).
(2) Search the NVD-DI index to locate and retrieve the frames that contain the information of the grid cells with the *GCodes* obtained in step 1) (Algorithm 4).
(3) Get the split points of the query line and the

[1] In a road network, the roads are curves with indefinite shapes. People typically use a sequence of road segments, which are line segments, to simulate an actual road. Since a normal query client moves on the roads of a network, its query line consists of the road segments on its way.

Table 1
Symbols and definitions.

Symbol	Definition
CGCode	Calculated *GCode* used on the client side
SGCode	Actual *GCode* used on the server side
QL	The list of *GCode* s of the grid cells covered by the query line of *q*
WL	The waiting time list for the frames, which keeps more direct index information of the desired grid cells
F_{IT}	The forward index table of frame *F*
$i\%j$	The result of *i* mod *j*

corresponding NNs, according to the edge information stored in these frames. The split points cut the query line into several line segments, and all the line segments together with their NNs form the final query result.

Since step (3) is quite straightforward, in the following, we will only describe the first two steps.

Algorithm 2. Client.

input: the query line of *q*
output: the query result of *q*
1 Invoke Algo. GetGCs; //To obtain an ordered list of grid cells overlapping with the query line of *q*;
2 Invoke Algo. LocateGCs; //To locate and retrieve the frames which contain the information of the list of grid cells obtained previously;
3 Get the split points of the query line and the corresponding NNs, return the final query result of *q*;

Algorithm 3. GetGCs.

input: the query line of *q*
output: *QL*(a list of the GCodes of grid cells overlapping with the query line of *q*)
1 list $QL=\emptyset$;
2 Tune into the channel and retrieve the first frame *F*;
3 Cut the query line of *q* into several line segments by the turning points;
4 **for** *each query line segment s* **do**
5 Calculate the *GCode*(s)of grid cell(s) containing the end point(s) of *s*;
6 Insert the *GCode*(s) obtained into *QL* in ascending order;
7 Return *QL*;

Step (1) In this step, Algorithm 3 – GetGCs is designed to obtain the list of *GCode*s of the grid cells that overlap with the query line of *q*. In particular, it cuts the query line into several line segments by its turning points,[2] and for each line segment, the algorithm obtains the *GCode*(s) of the grid cell(s) containing the end point(s) of the line segment, with the final result stored in *QL*.

Recall that for a given NVD structure, there is a one-to-one correspondence between a grid cell and its *GCode*. Hence, given the coordinates of a point, the *GCode* of the grid cell containing this point can be easily determined. In order to facilitate the query processing, we first assume that the broadcast NVD quad-tree on the client side to be a complete quad-tree with height *h*, and each leaf node at the lowest level represents a grid cell whose covering area is $H/2^h \times W/2^h$. Moreover, we assume that each client lies in a lowest-level grid cell and the calculated *GCode* is a string whose length equals to $2*h$. Hereinafter, we call this lowest-level grid cell supposed on the client side *Cgrid cell*, and use *CGCode* to represent the calculated *GCode*. Fig. 11 (a) gives an example to illustrate our approach, where the query line of *q* overlaps with Cgrid cells 110000, 110001, and 110100, hence $QL=\{110000, 110001, 110100\}$.

It is important to note that the above Cgrid cells are only supposed ones, the actual grid cells broadcast by the server may be located at different levels of the NVD quad-tree. From now on, we call these actual grid cells being broadcast at the server side *Sgrid cell*, and use *SGCode* to represent its *GCode*. It is not difficult to see that the actual NVD quad-tree broadcast by the server is a sub-tree of the complete quad-tree assumed by the client side, and each leaf node represents a Sgrid cell whose covering area is $H/2^i \times W/2^i$, where $i(1 \le i \le h)$ is the level of the cell in the tree. For a Sgrid cell that is not at the lowest level, it covers $2^{2*(h-i)}$ Cgrid cells. Comparing to Fig. 11(a), Fig. 11(b) presents the actual grid cells covered by the query line of the query *q*, i.e., 110000, 110001, and 1101. Note that although the *CGCode* is not always in accordance with the actual *SGCode*, we can still use the *CGCode*s to find out the actual grid cell where query *q* is located. This is because, for a frame *F* being broadcast, we have kept its *GCode* and *MaxGCode*, hence, if the *CGCode* being searched is between *F.GCode* and *F.MaxGCode*, then *F* must contain the information of the desired grid cell.

Step (2) In this step, Algorithm 4 – LocateGCs is performed to retrieve the list of grid cells kept in *QL*, which is in ascending order of the *GCode* value. We use *WL* to record the waiting times of the frames, which keeps more direct index information of the desired grid cells. For example, $WL[i]=2^j \times t_{frame}(0 \le j \le n-1)$ means that the frame, which will be broadcast after $2^j \times t_{frame}$ following the current frame, keeps more direct index information of grid cell *QL*[*i*]. Moreover, we use two variables *next* and *found* to specify the subscript of the element being searched in *QL* and keep the number of the found grid cells, respectively.

Algorithm 4. LocateGCs.

input: *QL* – An ordered list of the GCodes of grid cells covered by the query line of *q*;
F – The first frame retrieved when *q* tunes into the broadcast channel;
output: The frames which contain the grid cells of *QL*;
1 $m=QL.length$;
2 int $WL[m]=\{0\}$;
3 int $found=next=i=0$;
4 **while** $(i<m)$ **do**
5 **if**$(F.GCode \le QL[i] \le F.MaxGCode)$ **then**
6 **repeat**
7 $found++; next=(i+1)\%m; i++;$
8 **until**$(QL[i] \notin [F.GCode, F.MaxGCode])$;
9 break;
10 **for** $(i=next; i<next+m-1-found; i++)$ **do**

[2] Since a normal query client moves on the roads of a road network, the turning points of its query line are the intersections of adjacent road segments on its way.

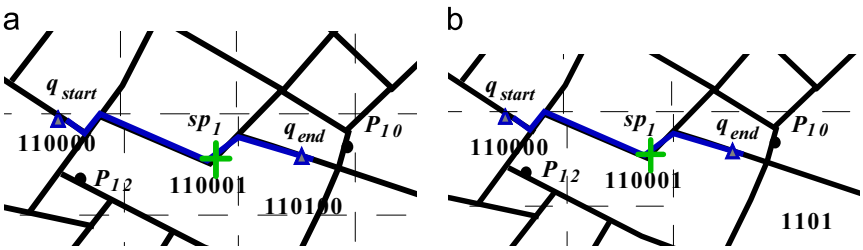

Fig. 11. The grid cells covered by the query line of q. (a) The client grid cells 110,000, 110,001, and 110,100 obtained by running Algorithm 4 and (b) the actual grid cells being broadcast: 110,000, 110,001, 1101.

```
11  | for (j = 0; j ≤ n−1; j++) do
12  |   if (QL[i%m] ∈ [F_IT[j].GCode, F_IT[(j+1)%n].GCode)) then
13  |     if (WL[i%m] < 2^j × t_frame) then
14  |       WL[i%m] = 2^j × t_frame;
15  |       break;
16  Let next be the subscript of the smallest item in WL;
17  Turn into doze mode, sleep for a time period whose length
    equals WL[next];
18  for (k = next+1; k ≤ next+m−1−found; k++) do
19  | WL[k%m] = WL[k%m] − WL[next]
20  Tune into the channel and retrieve the frame F;
21  while (found < m) do
22  | if (F.GCode ≤ QL[next] ≤ F.MaxGCode) then
23  |   repeat
24  |   | found++; next = (next+1)%m;
25  |   until (QL[next] ∉ [F.GCode, F.MaxGCode]);
26  | if found ≥ m then
27  | | break;
28  | Repeat lines 10–15;
29  | Repeat lines 17–20;
30  Return the frames retrieved.
```

We first check whether one or more items in QL are included in the first frame of F (Lines 4–9 in Algorithm 4). If the answer is affirmative, we modify the values of *next* and *found* correspondingly. Note that QL is regarded as a cyclic structure, which means the next item of $QL[m-1]$ is $QL[0]$. Then we use F_{IT} to set WL (Lines 10–15). For each non-found grid cell, if it belongs to $[F_{IT}[j].GCode,\ F_{IT}[(j+1)\%n].GCode)$[3] $(0 \le j \le n-1)$, then it means $F_{IT}[j].GCode$ may keep more direct index information of this frame, and we thus modify the related item in WL correspondingly. Note here we also need to modify the waiting time of each non-found item in WL (Lines 18 and 19) since a time duration of $WL[next]$ has elapsed. Finally, we repeat the following three steps until m grid cells have been found:

(i) Check whether $QL[next]$ is included in the current frame F. Since a frame may contain the information of more than one grid cell, we repeat step (i) until the next item being searched does not belong to frame F (Lines 22–25).
(ii) Repeat lines 10–15 to modify the waiting time of the non-found items in WL.
(iii) Repeat lines 17–20 to get the desired frame.

[3] If $F_{IT}[j].GCode > F_{IT}[(j+1)\%n].GCode$, then $[F_{IT}[j].GCode, F_{IT}[(j+1)\%n].GCode)$ equals to $[F_{IT}[j].GCode$, the $GCode$ of the last frame of a broadcast cycle $]\cup[$ the $GCode$ of the first frame of the broadcast cycle, $F_{IT}[(j+1)\%n].GCode)$.

To illustrate the process of Algorithm 4, we again use query q in Fig. 1 as an example. Note $QL = \{110000, 110001, 110100\}$, thus the lengths of QL and WL are both 3. Assume that the first frame F is 000110, since all the three cells in QL are behind frame F in the broadcast cycle, and 110000 is the nearest one from frame F, we know that frame 110000 is the first one being searched. Moreover, since $F_{IT}[3].GCode(100100) \le QL[0](110000) < F_{IT}[4].GCode(1111)$ and frame 100100 contains more direct index information of the desired frame, $WL[0]$ is set to be $8* t_{frame}$. The other two elements in WL are handled in a similar way. The detail of the above process is shown in phase 1 of Fig. 12. In phase 2, after sleeping for a duration of $8* t_{frame}$, query q wakes up and tunes into the channel to retrieve frame 100100, and WL is modified correspondingly. In phase 3, query q sleeps for a time duration of $4 \times t_{frame}$, and then wakes up to retrieve frame 110000. Since both 110000 and 110001 belong to [110000, 110001], we know that frame 110000 includes the information of cell 110000 and 110001. Note that 110001 is the *MaxGCode* of frame 110000, hence WL is modified according to the IT table of frame 110000. Finally, query q sleeps for a duration of $2 \times t_{frame}$, and wakes up to retrieve frame 1101. Since 110100 belongs to [1101, 110111], and 110111 is the *MaxGCode* of frame 1101, we can conclude that frame 1101 contains the information of cell 110100. Up to now, all the desired frames have been retrieved.

It is straightforward to see the time complexity of the first step is $O(|s|)$, where $|s|$ is the number of edges the query line covers. In the second step, since we need to retrieve m data frames, and when searching for one frame, we need to access $\log_2(N)$ frames in the worst case, where N is the number of frames in a broadcast cycle. Besides, when searching for the IT table of a frame to get the index information of the desired frame, we need to compare n index items in the worst case, where $n = \lceil \log_2(N) \rceil$. Hence, the time complexity of the second step is $O(m \times n \times \log_2(N))$, where m is the number of grid cells covered by the query line. The third step has the same time complexity ($O(|s|)$) as that of step one. Overall, the time complexity of the Client algorithm is $O(|s| + m \times n \times \log_2(N))$.

Till now, we have finished introducing all the details of our novel CN^{3}B query processing method. Before finishing this subsection, we highlight the advantages of our method as follows:

(1) Each query can be finished within one broadcast cycle length, as stated in Theorem 1 below.
(2) The total tuning time is much shorter than $m \times t_u$, where t_u is the average tuning time for retrieving one

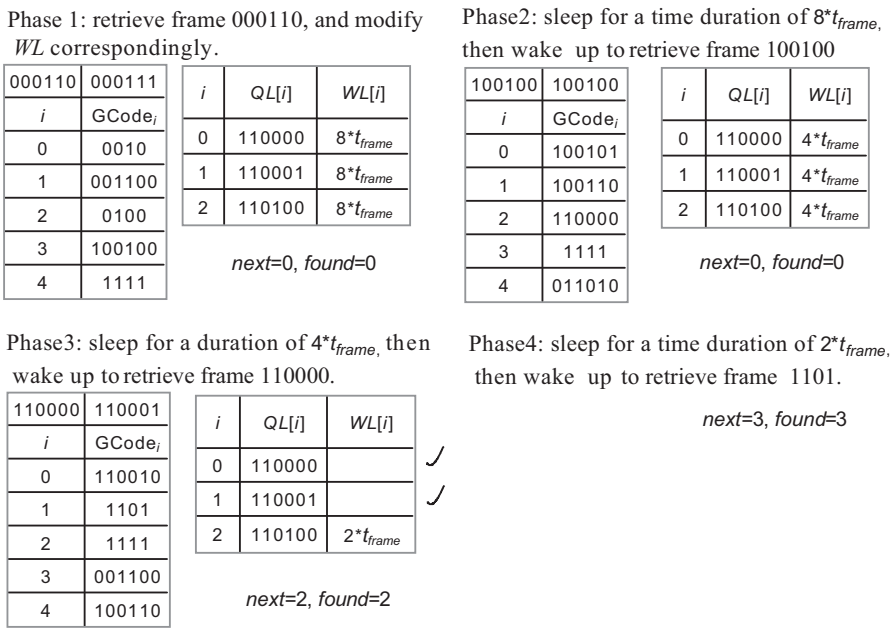

Fig. 12. An example of getting the frames desired for CN[3]B query processing.

grid cell separately. This is because we sort all the grid cells in ascending order of their *GCode*s and we can get more direct index information for grid cells in the back end while searching for grid cells in the front end.

Theorem 1. *With NVD-DI and the* Client *algorithm, each CNN search can be completed within one broadcast cycle.*

Proof. Assume that there are totally m frames overlapping with the query line of q, and P is a randomly selected frame from these m frames. Firstly, the broadcast cycle, which starts from the time instant when q tunes into the broadcast channel, includes every frame. Secondly, assume that the frame P is the t-th frame after the current one ($0 \le t \le N-1$, N is the number of frames in a broadcast cycle). Then, we can calculate which frame(s) will be retrieved in order to get the desired frame P according to the value of t:

- Convert decimal value t into a binary string $x_1x_2\cdots x_k$ ($k \le n$, n is the number of entries in IT). To simplify the discussion, we truncate leading zeros of the string to make $x_1 = 1$.
- Assume the binary string $x_1x_2\cdots x_k$ has l bits which equal to 1 ($l \le k$). We define a function f by $f(i) = h(i \in [1, l])$, where h is the position of the i-th bit in $x_1x_2\cdots x_k$ (counting from left to right) that equals to 1. So if the string is 1011, then $f(1) = 1$, $f(2) = 3$, $f(3) = 4$. According to our Client algorithm, a client wakes up to retrieve a frame after sleeping for a duration of $t_{frame} * 2^{k-f(i)}$ at the i-th iteration, where $i = 1, 2, \ldots, l$. This process will repeat l times. When the client wakes up at the l-th iteration, it will retrieve the t-th frame which is exactly the desired frame P.

Since $t < N$ and $k < \log_2(N)$, we have,

$$\sum_{i=1}^{l} 2^{k-f(i)} \le \sum_{i=1}^{k} 2^{k-i} \le 2^k - 1 \le 2^{\log_2(N)} - 1 < N \quad (1)$$

Thus, after waiting a time duration which is shorter than $t_{frame} * N$ and waking up l times, the frame P is retrieved within a broadcast cycle. Since P is a randomly selected frame, all these m frames can be retrieved in one broadcast cycle, therefore, the theorem follows. □

Actually, when tuning into a broadcast channel to retrieve a frame, Algorithm LocateGCs will modify the index information of the non-found frames according to the IT of the current frame (Lines 13 and 14 of Algorithm 4), which will cut down the tuning time to locate a frame.

3.2.5. Cost model

In this section, we conduct a performance analysis based on the assumption that the clients tune into the broadcast channel randomly and each edge in the road network has the same probability to be accessed. In wireless broadcast environments, tuning time and access time are the two major metrics to measure the access efficiency and energy consumption for mobile clients, respectively. Access latency is the time elapsed from the moment a query is issued to the moment it is answered, while tuning time is the time a mobile client stays in active mode to receive the requested data objects and index information. In what follows, we use the number of access data frames to evaluate the average access time (AAT) and the average tuning time (ATT) for query processing.

1. Average access time

Let N be the total number of frames in a broadcast cycle, S_D be the size (in the number of frames) of data used to include all the edges and objects, S_{NVD-DI} be the size (in

the number of frames) of the NVD-DI index, and m be the number of grid cells covered by the query line. Since NVD-DI is a complete distributed air index, it is obvious the probe-wait time for getting the index table is zero. Moreover, it is not difficult to derive that the average broadcast-wait time for getting the desired grid cells is $(N/2)+m$. By summing the probe-wait time and broadcast-wait time up, we have,

$$AAT = \frac{N}{2} + m = \frac{S_D + S_{NVD-DI}}{2} + m \tag{2}$$

2. Average tuning time

Let s_i be the size of an index table, s_F be the size of a broadcast frame, and m be the number of grid cells covered by the query line. Suppose a client tunes into the broadcast channel and retrieves the first frame F. Among the desired grid cells behind F, suppose the nearest one to F is located in frame $F_i (i \in [0, N])$, where F_i the i-th frame behind F. Note that NVD-DI is a distributed index and retrieving F_i needs 1 search step at least and $\lceil \log_2(N) \rceil$ search steps at most. Since we assume each edge in the road network has the same probability to be accessed, i.e., the probability of selecting any value from $[0, N]$ for i is equal, the average tuning time for accessing one frame, denoted by ATT_{frame}, can be computed as follows:

$$ATT_{frame} = \frac{1}{2} + \sum_{i=0}^{N-1} t(i) \tag{3}$$

where

$$t(i) = \begin{cases} 1 & i = 0 \\ t(i-x) + \frac{s_i}{s_F} & i > 0 \end{cases} \tag{4}$$

The first part of ATT_{frame} is the initial probe time for the first complete frame. Since each frame contains a pointer to the next frame, the initial probe requires retrieval of half of a frame, i.e., $\frac{1}{2}$, on average. The second part of ATT_{frame} ($\sum_{i=0}^{N-1} t(i)$) is the average number of frames accessed for retrieving the desired frame F_i, where x in Formula (4) is the maximum value no larger than i in the set of $\{2^0, 2^1, 2^2, \ldots, \lfloor N/2 \rfloor\}$.

Note that since we sort all the grid cells in ascending order of their cell numbers, we can get more direct index information for grid cells in the back end when searching for grid cells in the front end. Besides, the NVD quad-tree has good locality-preserving behavior. That is, geographically neighboring cells are also adjacent in their cell numbers. Thus, the set of grid cells related to a CN^3B query may list adjacently in the broadcast cycle. As a result, the total average tuning time for a CN^3B query is far smaller than $m \cdot ATT_{frame}$, and we have,

$$\frac{1}{2} + \sum_{i=0}^{N-1} t(i) + m \le ATT \ll \frac{1}{2} + m \cdot \sum_{i=0}^{N-1} t(i) \tag{5}$$

where $t(i)$ is defined in Formula (4).

4. Performance evaluation

This section presents the performance evaluation of our CN^3B query processing method. Since our approach is an NVD-DI based method, we call it NVD-DI-M for short. To our best knowledge, there is no other work on dealing with CNN queries in road networks under broadcast environments in the literature up to now. Hence, we will compare NVD-DI-M with DTM, a method presented in Section 3.1 which runs repeatedly when a query client q keeps moving along its query line segment. For simplicity, we only run DTM whenever a query q leaves the edge it locates and moves to another edge in our experiment study. Moreover, to evaluate the effect of the distributed index on access time prolongation and power saving, we will also compare NVD-DI-M with its another version that does not use NVD-DI index, denoted by NVD-No-DI for convenience.

The performance of NVD-DI-M versus DTM and NVD-No-DI is evaluated by measuring average access time and tuning time for processing CNN queries in road networks. Suppose the bandwidth of the broadcast channel is fixed, we use the number of bytes transferred over the broadcast channel as the time unit, instead of the real clock time. The reason is that given a certain bandwidth, transferred bytes can be easily converted into clock time. Note that here we avoid using the number of packets for comparison since the packet sizes in our experiments are varied to model different real life applications. The effect of the distributed index on access time prolongation and power saving of mobile clients is also evaluated in this section.

4.1. Environments and parameters

The system model in the simulation consists of a base station, a number of clients, and a broadcast channel. The available bandwidth is set to 84 Mbps. Our method is tested in both a real road network and a synthetic 2-dimensional grid network containing $M*M$ cells, where M is a system parameter. The road traffic network of San Francisco Bay Area in America [24], which consists of 175,343 nodes and 223,308 edges, is used as the real road network.

First, the performance of our methods is tested both on real data and uniform data sets in the above mentioned real road network in Section 4.2. Then the performance is evaluated both on real data and uniform data sets in a synthetic grid network in Section 4.3. In particular, a set of data objects obtained through the generator proposed in [2] works as the real data set. Fig. 13 depicts the real road network of San Francisco Bay and the real date objects in it, with roads and data objects represented by blue lines and red points, respectively. The uniform data set contains several data objects which follow uniform distributions. Table 2 lists the parameters for the investigation. The values in bold face are the default values in our experiments and the results presented in the following sections are the average performance of all the queries in the system. In the experiments the packet size is varied from 256 to 2048 bytes. Since the storage space needed for keeping an edge is fixed, varying packet size is equivalent to varying the value of the threshold λ discussed in Section 3.2.2. Moreover, three road networks, San Jose, San Rafael and Oldenburg, are used to test the applicability of our method in Section 4.3.

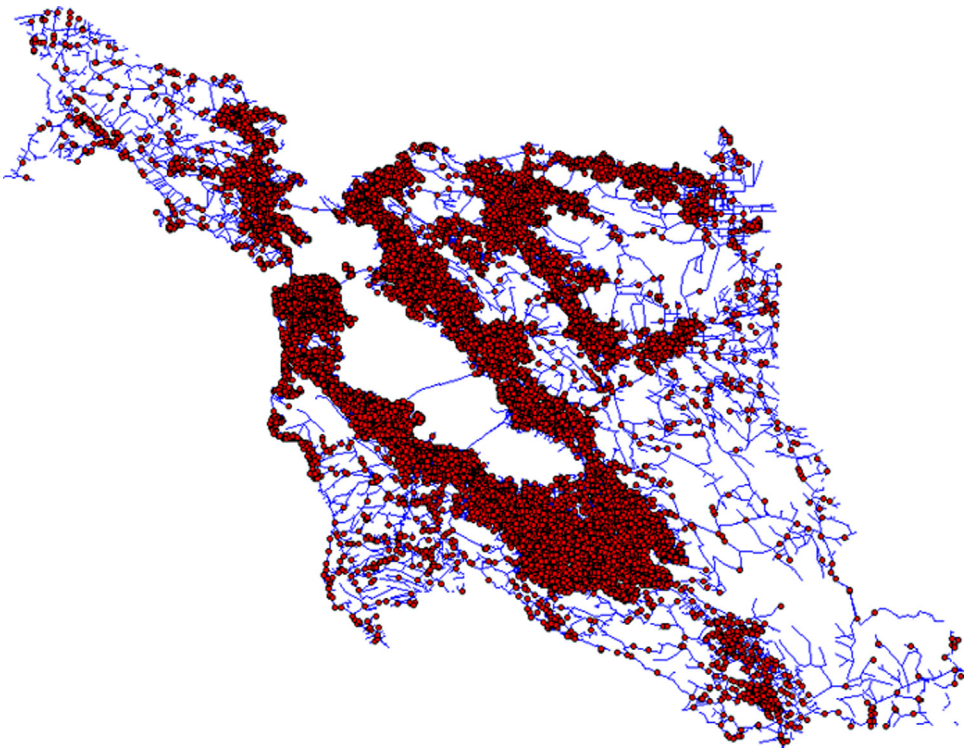

Fig. 13. San Francisco Bay and real data set. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Table 2
Dataset parameters.

Number of queries	10, 50, **100**, 150, 200(k)
Number of objects	5, 10, **20**, 30, 40(k)
Query line length	2, 4, **6**, 8, 10 (edges)
Packet size	256, 512,**1024**, 2048 (bytes)
The number of cells in grid (M*M)	100*100, 200*200, **300*300**, 400*400,500*500
The distribution of objects	Real data, uniform distributed data

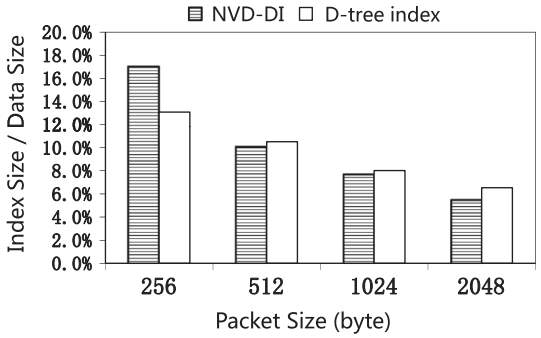

Fig. 14. Normalized index size for real data set.

4.2. Performance on the real road network of San Francisco Bay Area

4.2.1. Access latency

Firstly, NVD-DI-M is compared with NVD-No-DI and DTM in terms of access latency. Access latency is influenced by the length of the broadcast cycle and the number of multiple paths provided by an index. The broadcast cycle is affected by the index size. The larger the index size, the longer the broadcast cycle. Fig. 14 shows the normalized index sizes of our NVD-DI index and the D-tree index for the real data set. This figure tells us that the former outperforms the latter in most cases, while the D-tree index is smaller than our index when the packet size is 256 byte. This is because each frame in our method keeps an IT index table, which will result in a larger normalized index size when the packet size is small. However, NVD-DI is a fully distributed index which allows the query client to start a query processing at arbitrary time instants. Thus the performance of NVD-DI-M, which is based on NVD-DI, is much better than that of DTM even when the packet size is small.

Fig. 15(a) and (b) show the effect of packet size on the access latency of these three methods. It can be seen that NVD-DI-M and NVD-No-DI outperform DTM significantly. On average, NVD-DI-M incurs only 20.39% and 19.65% of access time compared to DTM for the uniform and real data sets, respectively. Similarly, NVD-No-DI incurs only 18.14% and 17.62% of access time compared to DTM for the uniform and real data sets, respectively. This is because: (1) DTM is based on a D-tree air index and its mobile clients need to wait for the beginning of the broadcast cycle to start the search. This leads to a longer latency; (2) DTM repeatedly launches a query when query q moves along the query line. On the contrary, NVD-DI-M is based

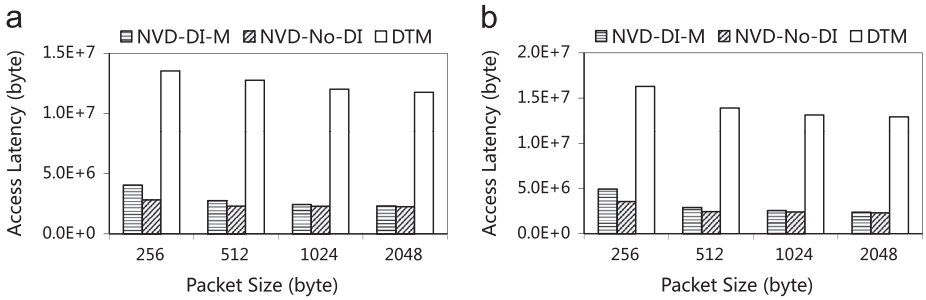

Fig. 15. Access latency for different packet sizes. (a) Uniform data set and (b) real data set.

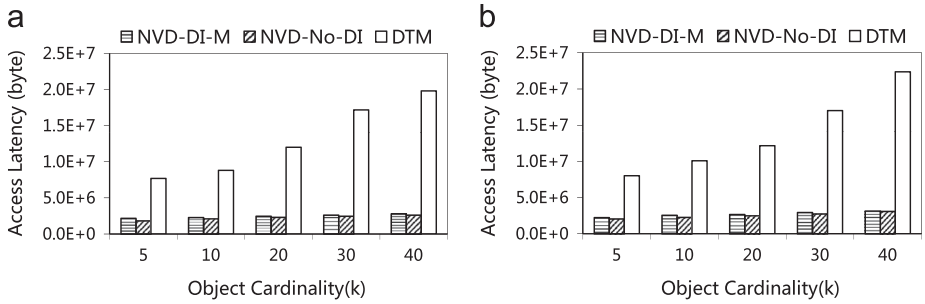

Fig. 16. Access latency for different object cardinalities. (a) Uniform data set and (b) real data set.

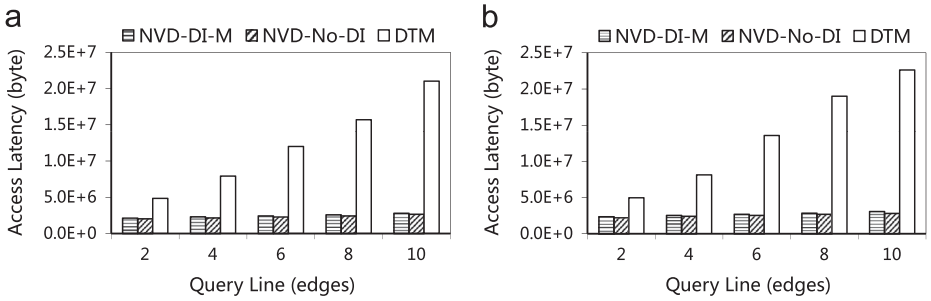

Fig. 17. Access latency for different query line lengths. (a) Uniform data set and (b) real data set.

on the NVD-DI index, which is a distributed index that has sufficient multiple paths for MCs to access the desired data simultaneously. Moreover, by determining the split points on the query line, we can continuously give the NNs of query q, instead of repeatedly issuing and processing queries. For NVD-No-DI, the reason why its performance is a little better than that of NVD-DI-M lies in that it does not use any index.

As shown in Fig. 16(a) and (b), the access latency of these three methods increases with the increment of the object cardinality for both the uniform and real data sets. For DTM, this is because that the number of Voronoi polygons of NVD increases as the object size increases, thus the sizes of NVD and D-tree index increase correspondingly which will, in turn, enlarge the access latency and the tuning time of the method. For NVD-DI-M and NVD-No-DI, the reason lies in that the number of boundaries of adjacent Voronoi polygons increases as the object cardinality increases, which in turn enlarges the sizes of edge table and the NVD structure. However, the increasing trend of NVD-DI-M and NVD-No-DI is smooth compared to that of DTM.

Now we evaluate the effect of query cardinality on the access latency of these three methods for the uniform and real data sets, respectively. The result tells us that the query size does not have an effect on the access time of the methods and we omit the figures since the result is apparent. As shown in Fig. 17(a) and (b), the access latency of these three methods increases with the increment of the query line length for both the uniform and real data sets. For NVD-DI-M and NVD-No-DI, the reason lies in that the number of grid cells overlapped by the query line increases as the query line becomes longer, thus the access time increases. For DTM, the reason is that as the query line becomes longer, more static NN queries should be launched to get the query result of q. However, the growing trend of NVD-DI-M (and NVD-No-DI) is more gentle than that of DTM.

4.2.2. *Tuning time*

In a broadcasting environment, it is desirable to cut down the tuning time so as to save power consumption of the clients. In general, tuning time is dependent on the

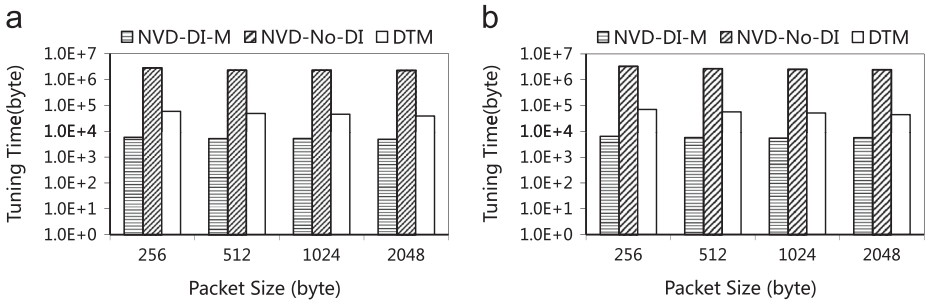

Fig. 18. Tuning time for different packet sizes. (a) Uniform data set and (b) real data set.

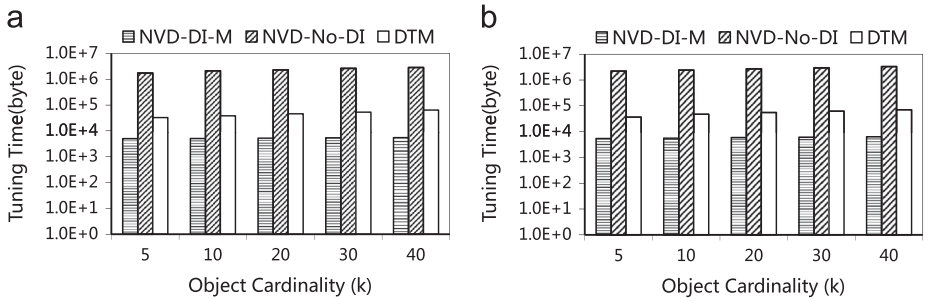

Fig. 19. Tuning time for different object cardinalities. (a) Uniform data set and (b) real data set.

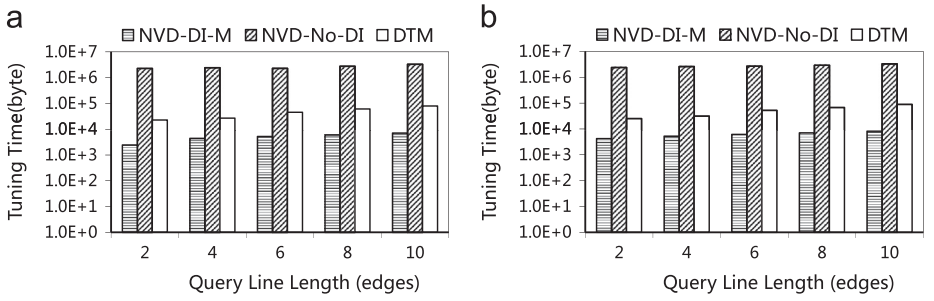

Fig. 20. Tuning time for different query line lengths. (a) Uniform data set and (b) real data set.

index structures and search algorithms adopted, parameters specified in a query, and system parameters such as packet size.

Fig. 18(a) and (b) presents the effect of the packet size on the tuning time performance of these three methods. These two figures show that the tuning time of the methods decreases as the packet size increases, and NVD-DI-M outperforms NVD-No-DI and DTM for both the uniform and real data sets. On average, NVD-DI-M incurs only 11.50% and 10.61% of tuning time compared to DTM for the uniform and real data sets, respectively. However, the tuning time of NVD-No-DI is very high and it incurs 438.05 and 463.12 times of that of NVD-DI-M for the uniform and real data sets, respectively. The reason why the tuning time of NVD-No-DI is so high lies in that it does not use any air index and the mobile clients have to continuously tune into the broadcast channel to retrieve the information needed.

Fig. 19(a) and (b) shows that the object cardinality has an obvious effect on the tuning time of these three methods for both the uniform and real data sets. The tuning time of the methods increases with the increment of the object size, due to the reason very similar to that of Fig. 16(a) and (b). Similarly, the increasing tendency of NVD-DI-M (and NVD-No-DI) is more gentle than that of DTM.

Now we evaluate the effect of query cardinality on the tuning time of these three methods. The result tells us that the query size does not have an effect on the tuning time of the methods and we omit to present the figures since the result is apparent. As shown in Fig. 20(a) and (b), the tuning time of these three methods increases with the increment of the query line length for both the uniform and real data sets. The reasons are similar to that of Fig. 17 (a) and (b). Note that in the above three figures (Figs. 18, 19 and 20), the y-axis is shown in a logarithmic scale.

4.3. *Performance on the road networks of San Jose, San Rafael and Oldenburg*

In this subsection, we use three other road networks, San Jose and San Rafael in America, and Oldenburg in Germany, to evaluate the applicability of our method on different networks. The San Jose (San Rafael and

Oldenburg, resp.) road network consists of 49,482 (16,130 and 6105, resp.) nodes and 63,439 (20,178 and 7035, resp.) edges, and includes 6*k* (2*k* and 0.7*k*, resp.) data objects. Due to space limitation, in this set of experiments, we only give the experimental results of NVD-DI-M by varying the query line length (edges) from 2 to 10, and fixing other parameters (e.g., packet size, query cardinality) at their default values.

Fig. 21(a) and (b) depicts the performance of our NVD-DI-M method on different road networks. As can be seen from these two figures, the access time and tuning time for different networks maintain at a reasonable level and increase with the growth of the query line length. We can conclude that the access time and tuning time are mainly affected by the number of edges and objects in the road network, and less affected by the distribution of the edges in the road network. This is because we have addressed the unbalanced edge distribution problem of real road networks, by proposing an optimization technique to compress the grid cells with small number of edges to form a new cell with large number of edges, as described in Section 3.2.2.

4.4. Performance on synthetic grid network

In this subsection, we compare the performance of the three algorithms with respect to the number of vertices in the road network. A synthetic 2-dimensional grid network is used as the basic network and 300∗300 is the default grid size of the network which includes 10k randomly distributed objects. We vary the grid size of the network from 100∗100 to 500∗500, and the number of objects included in the network varies correspondingly to maintain a constant ratio of object cardinality and edge cardinality. The other parameters, such as packet size, query line length, and query cardinality, are fixed at their default values. Fig. 22(a) and (b) depicts the effect of network size on the access latency performance of these three methods. These two figures show that as the network size increases, the access time of these algorithms increases obviously for both the uniform and real data sets. This is due to the fact that the number of edges and the average size of Voronoi polygons in the system increase as the network size increases, which in turn causes the length of the broadcast cycle to increase.

As shown in Fig. 23(a) and (b), the tuning time of these algorithms increases as the network size increases. For NVD-DI-M, this is because that the number of edges and the size of NVD structure increase as the network size increases. Thus, the number of index items of each distributed index table increases slightly which in turn slightly increases the tuning time. For DTM, the reason mainly lies in that the D-tree index becomes larger as the network size increases, thus incurring more time on searching the D-tree index, which in turn causes the tuning time to increase correspondingly. Moreover, the reason for NVD-No-DI is the same as that of Fig. 22.

4.5. Summary of experimental results

In summary, we revealed the following three observations from our experimental results.

(1) The performance of NVD-DI-M is much better than that of DTM. In particular, it incurs no more than 20.39% of

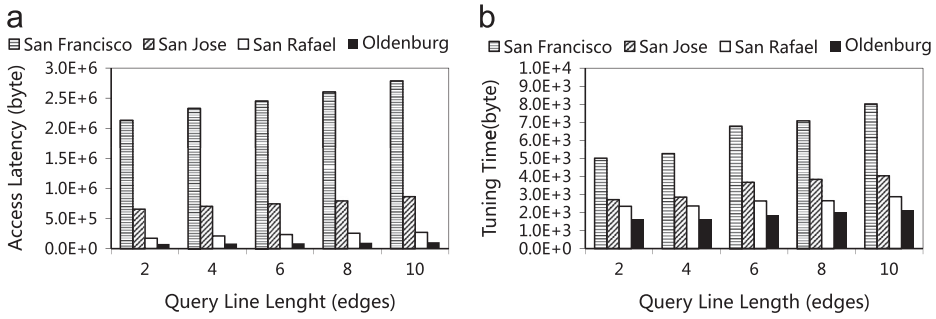

Fig. 21. Access time and tuning time of NVD-DI-M for different road networks. (a) Access latency and (b) tuning time.

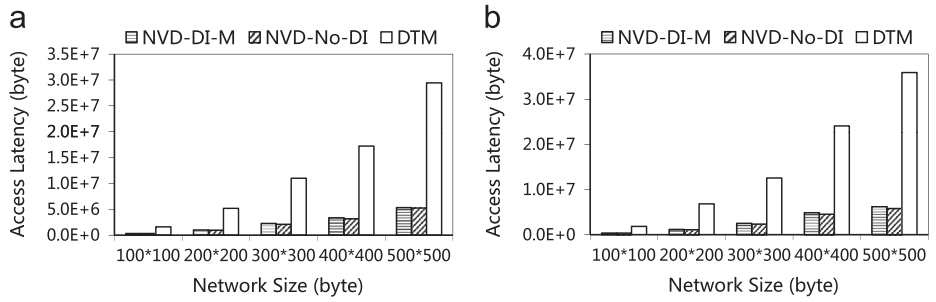

Fig. 22. Access latency for different network sizes. (a) Uniform data set and (b) real data set.

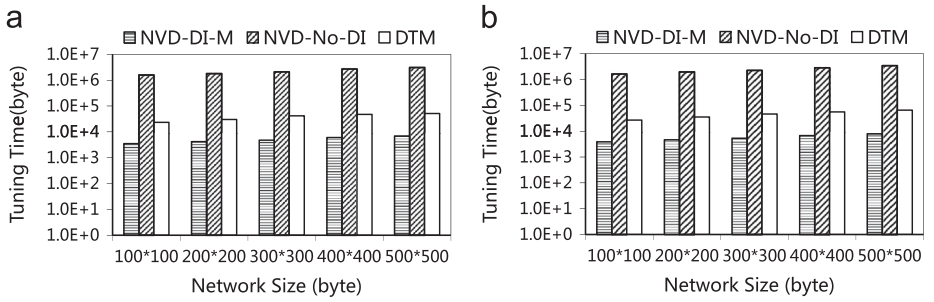

Fig. 23. Tuning time for different network sizes. (a) Uniform data set and (b) real data set.

access time and 11.50% of tuning time compared to DTM.

(2) Using NVD-DI can result in a decrease in tuning time to more than 438 times, at the expense of 10.25% access time prolongation on average. Thus, NVD-DI is a very effective and cost-efficient indexing mechanism.

(3) NVD-DI-M is robust with respect to different values of the packet size, object cardinality, query line length, and network size.

5. Conclusion and future work

In this paper, we addressed the problem of processing CNN queries in road networks under wireless broadcast environments (CN^3B). As far as we know, this work is the first attempt to solve the given problem. At first, we presented an efficient approach, called NVD quad partition, to partition the NVD structure of the underlying network into grid cells. These cells are generally balanced in size and have good locality-preserving behavior. We then proposed an NVD-based distributed index, called NVD-DI, to facilitate efficient CN^3B query processing. Finally, we presented an efficient algorithm on the client side to process CN^3B queries. Experimental study on both real data and uniform data sets demonstrate the efficiency of our method and the associated NVD-DI index.

For future work, we intend to extend our method to process Ck NN ($k > 1$) queries. We also plan to study how to use NVD-DI to process other location-based queries, such as window queries and reverse nearest neighbor queries.

Acknowledgements

This work was substantially supported by National Natural Science Foundation of China under Grants No.61309002, No. 61173049 and No. 61300045, Science Foundation of Hubei Province under Grant No.2012FFB07401, and China Postdoctoral Science Foundation under Grant No. 2013M531696.

References

[1] S. Berchtold, D.A. Keim, H.-P. Kriegel, T. Seidl, Indexing the solution space: a new technique for nearest neighbor search in high-dimensional space, IEEE Trans. Knowl. Data Eng. 12 (1) (2000) 45–57.

[2] T. Brinkhoff, A framework for generating network-based moving objects, GeoInformatica 6 (2) (2002) 153–180.

[3] H.-J. Cho, C.-W. Chung, An efficient and scalable approach to CNN queries in a road network, in: Proceedings of the 31st International Conference on Very large Databases, VLDB Endowment, 2005, pp. 865–876.

[4] Y. Gao, B. Zheng, G. Chen, C. Chen, Q. Li, Continuous nearest-neighbor search in the presence of obstacles, ACM Trans. Database Syst. (TODS) 36 (2) (2011) 9.

[5] Y. Gao, B. Zheng, G. Chen, Q. Li, C. Chen, G. Chen, Efficient mutual nearest neighbor query processing for moving object trajectories, Inf. Sci. 180 (11) (2010) 2176–2195.

[6] Y. Gao, B. Zheng, G. Chen, Q. Li, X. Guo, Continuous visible nearest neighbor query processing in spatial databases, VLDB J. 20 (3) (2011) 371–396.

[7] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In Proceedings of ACM SIGMOD Conference, pp. 47–57, Boston, MA, June 1984. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan Kaufmann, San Mateo, CA, 1988.

[8] S. Hambrusch, C.-M. Liu, W.G. Aref, S. Prabhakar, Query processing in broadcasted spatial index trees, in: Advances in Spatial and Temporal Databases, Springer, 2001, pp. 502–521.

[9] X. Huang, C.S. Jensen, S. Šaltenis, The islands approach to nearest neighbor querying in spatial networks, in: Advances in Spatial and Temporal Databases, Springer, 2005, pp. 73–90.

[10] Y.-K. Huang, Z.-W. Chen, C. Lee, Continuous k-nearest neighbor query over moving objects in road networks, in: Advances in Data and Web Management, 2009, Springer, pp. 27–38.

[11] T. Imielinski, S. Viswanathan, B. Badrinath, Data on air: organization and access, IEEE Trans. Knowl. Data Eng. 9 (3) (1997) 353–372.

[12] C.S. Jensen, J. Kolářvr, T.B. Pedersen, I. Timko, Nearest neighbor queries in road networks, in: Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems, ACM, 2003, pp. 1–8.

[13] M. Kolahdouzan, C. Shahabi, Voronoi-based k nearest neighbor search for spatial network databases, in: Proceedings of International Conference on Very large Databases. VLDB Endowment, 2004, pp. 840–851.

[14] M.R. Kolahdouzan, C. Shahabi, Continuous k-nearest neighbor queries in spatial network databases, in: Proceedings of STDBM, vol. 4. Citeseer, 2004, pp. 18–25.

[15] G. Li, Y. Li, J. Li, L. Shu, F. Yang, Continuous reverse k nearest neighbor monitoring on moving objects in road networks, Inf. Syst. 35 (8) (2010) 860–883.

[16] C.-M. Liu, S.-Y. Fu, Effective protocols for kNN search on broadcast multi-dimensional index trees, Inf. Syst. 33 (1) (2008) 18–35.

[17] K. Mouratidis, M.L. Yiu, D. Papadias, N. Mamoulis, Continuous nearest neighbor monitoring in road networks, in: Proceedings of International Conference on Very large data bases, VLDB Endowment, 2006, pp. 43–54.

[18] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, Wiley Series in Probability and Statistics, vol. 501, John Wiley & Sons, 2009.

[19] D. Papadias, J. Zhang, N. Mamoulis, Y. Tao, Query processing in spatial network databases, in: Proceedings of the 29th International Conference on Very large Databases, vol. 29, VLDB Endowment, 2003, pp. 802–813.

[20] K. Park, H. Choo, P. Valduriez, A scalable energy-efficient continuous nearest neighbor search in wireless broadcast systems, Wirel. Netw. 16 (4) (2010) 1011–1031.

[21] K. Park, M. Song, C.-S. Hwang, Adaptive data dissemination schemes for location-aware mobile services, J. Syst. Softw. 79 (5) (2006) 674–688.
[22] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, ACM Sigmod. Rec. 24 (2) (1995) 71–79.
[23] M. Safar, Enhanced continuous kNN queries using pine on road networks, in: Proceedings of the 1st International Conference on Digital Information Management, IEEE, 2006, pp. 248–256.
[24] F.S. University, ⟨http://www.cs.fsu.edu/lifeifei/spatialdataset.htm⟩, 2013.
[25] Y. Wang, Y. Gao, L. Chen, G. Chen, Q. Li, All-visible-k-nearest-neighbor queries, in: Database and Expert Systems Applications, Springer, 2012, pp. 392–407.
[26] J. Xu, B. Zheng, W.-C. Lee, D.L. Lee, The d-tree: an index structure for planar point queries in location-based wireless services, IEEE Trans. Knowl. Data Eng. 16 (12) (2004) 1526–1542.
[27] B. Zheng, W.-C. Lee, D.L. Lee, Search continuous nearest neighbors on the air, in: Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004, pp. 236–245.
[28] B. Zheng, W.-C. Lee, D.L. Lee, On searching continuous k nearest neighbors in wireless data broadcast systems, IEEE Trans. Mob. Comput. 6 (7) (2007) 748–761.
[29] B. Zheng, W.-C. Lee, K.C. Lee, D.L. Lee, M. Shao, A distributed spatial index for error-prone wireless data broadcast, VLDB J. 18 (4) (2009) 959–986.