On Searching Continuous k Nearest Neighbors in Wireless Data Broadcast Systems

Baihua Zheng, *Member*, *IEEE*, Wang-Chien Lee, *Member*, *IEEE*, and Dik Lun Lee

Abstract—A *continuous nearest neighbor* (CNN) search, which retrieves the nearest neighbors corresponding to every point in a given query line segment, is important for location-based services such as vehicular navigation and tourist guides. It is infeasible to answer a CNN search by issuing a traditional nearest neighbor query at every point of the line segment due to the large number of queries generated and the overhead on bandwidth. Algorithms have been proposed recently to support CNN search in the traditional client-server systems but not in the environment of wireless data broadcast, where uplink communication channels from mobile devices to the server are not available. In this paper, we develop a generalized search algorithm for continuous k-nearest neighbors based on *Hilbert Curve Index* in wireless data broadcast systems. A performance evaluation is conducted to compare the proposed search algorithms with an algorithm based on R-tree Air Index. The result shows that the Hilbert Curve Index-based algorithm is more energy efficient than the R-tree-based algorithm.

Index Terms—Continuous nearest neighbor search, broadcast, indexing, location-based services.

1 Introduction

With the widespread deployment of wireless networks and the fast growing popularity of smart mobile devices, there has been an increasing interest in wireless data services from both industrial and academic communities in recent years. There are two primary approaches for delivering wireless data services: *point-to-point* and *broadcast* (or *point-to-multipoint*) systems [1], [2]. Point-to-point access employs a basic client-server model, where the server is responsible for processing a query and returning the result to the client via a dedicated point-to-point channel. Wireless data broadcast systems, usually without an uplink channel available to the clients, have the server actively pushing data to the clients. The server determines the data and its schedule to be broadcast. A client, without sending any request to the server, simply listens to a broadcast channel to retrieve data based on her queries and, thus, is responsible for query processing.

The point-to-point data access approach is particularly suitable for light-loaded systems where contention for wireless bandwidth and server resources is not severe. However, the overall system performance can deteriorate quickly as the number of users and the system workload increase. Compared with point-to-point access, broadcast is a very attractive alternative [3], [4], [5]. It allows simultaneous access by an arbitrary number of mobile clients without causing interreceiver interference and, thus, can achieve an efficient usage of the server resource and scarce wireless bandwidth. Moreover, clients can retrieve desirable information from the broadcast channel without revealing to the server their operations and intentions and, thus, it preserves user privacy.

Wireless data broadcast services have been available as commercial products for many years, e.g., StarBand (www.starband.com) and Hughes Network (www.direcpc.com). Recently, there has been a push for such systems from the industry and various standard bodies. One example is the MSN Direct Service (www.msndirect.com), based on the smart personal objects technology (SPOT) and the DirectBand Network. With a continuous broadcast network using FM radio subcarrier frequencies, mobile devices (e.g., smart watches and PDAs) can continuously receive timely information such as stock quotes, airline schedules, local news, weather, and traffic information.

In this paper, we focus on location-based data services via wireless broadcast. Indeed, among the various envisaged wireless data services, location-based services are considered the most important ones. While data (or information) is important to users, it is only valuable when available at the right time, *right place*. The demand for location-based data (LBD), e.g., pollution index, local traffic conditions, restaurant locations, navigation maps, weather condition, etc., is tremendous due to the broad application base.

As in many real applications, mobile clients prefer issuing queries while they are moving. Consequently, continuous processing for different queries becomes a more and more important issue. Among them, the *Continuous Nearest Neighbor (CNN)* search is a new and important class of queries that find a set of nearest neighbors corresponding to every point in a given *query line segment*. Every object o in the answer set *dominates* a part of the given line segment, i.e., o is the nearest neighbor to any query point lying on

- *B. Zheng is with the School of Information Systems, Singapore Management University, Stamford Road 80, Singapore 178902. E-mail: bhzheng@smu.edu.sg.*
- *W.-C. Lee is with the Department of Computer Science and Engineering, The Pennsylvania State University, 360D Information Sciences and Technology Building, University Park, PA 16802. E-mail: wlee@cse.psu.edu.*
- *D.L. Lee is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: dlee@cse.ust.hk.*

Manuscript received 17 Jan. 2005; revised 17 Aug. 2006; accepted 18 Oct. 2006; published online 7 Feb. 2007.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0011-0105.
Digital Object Identifier no. 10.1109/TMC.2007.1004.

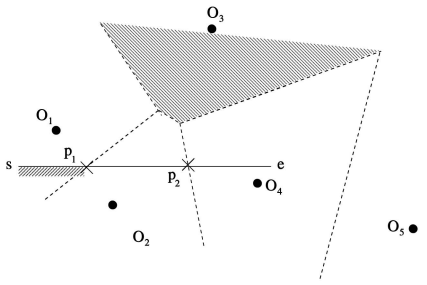


Fig. 1. Example of CNN search.

that part of line segment. An illustrative example is given in Fig. 1, in which the answer set to the query line $\overline{se}$ contains three objects, namely, O_1, O_2, and O_4. O_1 dominates the shadowed line segment $\overline{sp_1}$, while O_2 dominates $\overline{p_1p_2}$ and O_4 dominates $\overline{p_2e}$. Those partial line segments (we call them *valid scopes*) and their corresponding nearest data objects are returned as the answer set. p_1 and p_2 are called *split points* since they are the points at which the nearest objects along the line segment change [6]. Examples of CNN search are everywhere in our daily life. For example, a visitor touring in Manhattan may want to find all the nearest Chinese restaurants along Fifth Avenue from her current position to Central Park. A wildlife observer in Yellow Stone National Park may issue a query via a wireless mobile device to find the nearest observation points where she can most likely see wolves along the trail she is hiking.[1]

Due to the mobility of users and their devices, the query submission point may change continuously, which makes retrieval of location-based data a challenge. This issue of query continuity is particularly important for navigation and tour guide applications. Continuously issuing nearest neighbor queries while moving is obviously not a feasible strategy. Thus, efficient algorithms for processing CNN queries are required. As *energy conservation* is one of the most critical issues for mobile clients, our solution aims at reducing a mobile client's energy consumption when it accesses the data for query processing, i.e., improving energy consumption with comparable access latency performance.[2]

There are some existing studies on CNN search in the traditional client-server systems [6], [7], [8], [13]. However, these algorithms have the following constraints: 1) they require an uplink channel from a mobile client to the server and, hence, are not applicable to wireless data broadcast systems, and 2) they do not address the important issue of energy conservation. This paper, to the best of the authors' knowledge, presents the first study on enabling energy efficient continuous search of k-nearest neighbors (CkNN) in wireless data broadcast systems.[3] The primary contributions of this study cover the following aspects:

1. While a trail may not be a straight line, it can be decomposed into multiple line segments.

2. Access latency is also very important. Since there is no approach which can optimize both at the same time, we focus on energy efficiency in this paper. In the simulation section, BOTH the tuning time performance and access latency will be presented, which gives readers a full picture of the relative trade-offs between them so readers can decide what is the best for their applications.

3. A preliminary report of this study appeared in [17].

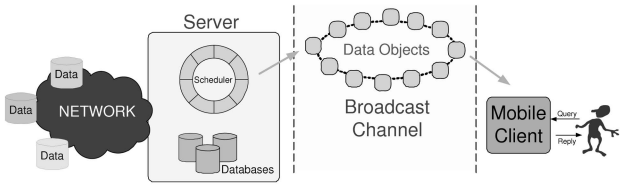


Fig. 2. A wireless data broadcast system.

- An energy-efficient search algorithm based on the Hilbert Curve (HC) index is developed to support CkNN queries for wireless data broadcast systems.
- A set of heuristics and claims that serve as the core of our CkNN search algorithm are identified and formally proven.
- A theoretical analysis is performed to estimate the performances of the proposed CkNN search algorithm under different index organizations.
- An extensive simulation is conducted to evaluate the performance of the proposed CkNN search algorithm using both synthetic and real data sets.

The rest of this paper is organized as follows: Section 2 presents background, system model, and related work to our study. Section 3 describes the proposed CkNN search algorithm based on the Hilbert Curve index. Implementation issues on query partitioning and index organization are discussed in Section 4. Section 5 evaluates performance of the proposed CkNN search algorithm and another algorithm modified from an existing work [6]. Finally, Section 6 concludes this paper and points out directions of the future work.

2 Support Location-Based Queries via Wireless Broadcast

Location-based query processing plays a key role in supporting location-based services. As described earlier, CkNN search is an important class of location-based queries. To facilitate our discussion of the CkNN search problem in wireless data broadcast systems, in this section, we first describe the general system model, assumptions, and constraints of wireless data broadcast systems. Next, we discuss the issues faced by adopting a spatial air index in wireless data broadcast systems and review some related work.

2.1 System Model

A wireless data broadcast system consists of three parts: 1) the communication mechanism, 2) the broadcast server, and 3) the mobile clients. Fig. 2 shows a high-level overview of the system model. Broadcast channels are the main communication mechanism. Without loss of generality, we assume that only one broadcast channel with limited bandwidth is allocated for the applications in our study and there is no uplink channel for clients to send requests or feedback to the server. In addition, all the information is disseminated to the clients in the unit of *pages*. The broadcast server is interfaced with other data sources via high speed networks and, thus, can be considered as a logical data source for all the mobile users in the system. The server is responsible for determining and scheduling

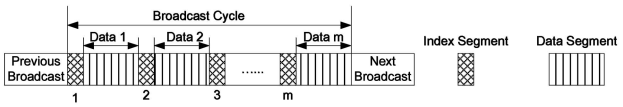


Fig. 3. *(1, m)* interleaving scheme.

the data for broadcast. Thus, we assume that the server has full knowledge of all the data objects under broadcast. It *periodically* disseminates data objects to its clients via the shared broadcast channel. A data object consists of a set of searchable attributes and a content body. Since we are focusing on location-based queries in this paper, we simply assume the searchable attributes consist of geospatial coordinates. A complete broadcast of data objects is called a *broadcast cycle*. From the viewpoint of users, wireless broadcast is perceived as a *linear* stream of data objects flowing along the time axis. Logically, there are no specific start and end objects for a broadcast cycle, which may start at any data object and end at the next appearance of the same data object. Content updates to the data objects are reflected between successive broadcast cycles.

The mobile clients, usually having limited power supply and storage space, also play an important role in the system due to the client-side processing. Each client has to continuously monitor the broadcast channel to receive the data objects of interests (as specified by user queries). Blindly checking every data object broadcast on air obviously consumes a lot of energy of mobile clients. To address this issue, *air indexing* techniques have been proposed [4], [5]. The basic idea is to provide auxiliary index information that annotates the broadcast data objects. Based on index information (on indexed attribute values, arrival schedule, length of data items, etc.) broadcast along with data objects, mobile clients are able to selectively skip unauthorized or unwanted objects by slipping into *doze mode* and switching back to *active mode* only when the data of desire arrives. This technique, alleviating the workload of the server and reducing energy consumption of mobile clients, is particularly important for wireless data broadcast.

Existing studies suggest that the air index information should be interleaved with the data objects in order to reach a good performance [4], [5]. Thus, in this paper, we adopt the *(1, m)* interleaving scheme, where an optimal m partition of the data set in a broadcast cycle can be derived [4]. As shown in Fig. 3, index information is interleaved with the m data partitions (called *segments*). Some existing work proposed replicating the frequently accessed data objects based on user access patterns, but that is out of the scope of our study. We focus on the general design of search algorithms based on air index structures which do not assume specific knowledge of access patterns. Thus, we assume a flat broadcast (i.e., a data object is broadcast only once in a cycle) in this paper.

This study, similar to the existing work in the literature, uses *access latency* and *tuning time* as the primary performance metrics.[4] The former is the time elapsed between the moment when a query is issued to the moment when all the requested data are received. The latter represents the period of time a client has to be active in order to finish one query. Most existing work assumes that the setup time for tuning into the broadcast channel and switching into the doze mode is negligible [4]. Therefore, the tuning time can be regarded as the major metric to evaluate power consumption. However, as pointed out in [9], mobile devices do require a nonnegligible overhead in terms of time and energy dissipation to go from doze mode to active mode and vice versa. Therefore, frequent on/off switching actually incurs significant power consumption. We also extend our evaluation to count the average number of switches incurred by a query and the energy consumed by switch operations in our evaluation.[5]

2.2 Location-Based Query Processing

To facilitate access of location-based data in mobile computing, we assume that positioning technology (e.g., GPS) is available for the clients to obtain their own positions. Based on client location and other available information (e.g., moving speed, direction, and cached query results), location-based applications such as navigation can estimate the frequency and timing for reissuing queries. Thus, in this paper, we focus only on the query processing algorithms. To simplify our discussion, we consider only one homogeneous data type in this study, i.e., the data objects broadcast on air have the same data type as an ATM. In other words, the channel is assumed to continuously broadcast information related to different ATMs and the clients are issuing queries like "Where are the nearest ATMs on my way to JFK airport along I-678?" Since a wireless broadcast channel can be logically divided into multiple subchannels, data objects of other data types (e.g., restaurants or hospitals) can be broadcast in other allocated subchannels. While complicated applications may issue complex queries that involve multiple data types, our study represents an initial step toward this direction. Finally, we assume that the locations of data objects seldom change.

To support location-based queries such as $C\!k\mathrm{NN}$ in wireless data broadcast systems, index information on location attributes of data objects is broadcast to facilitate client-side query processing. Thus, well-known spatial indices (e.g., R-trees) are candidates for air indexing. However, the unique characteristics of wireless data broadcast make the adoption of existing spatial indices inefficient (if not impossible). Specifically, traditional spatial indices are designed to cluster data objects with spatial locality. They usually assume resident storage (such as disk and memory) and adopt search strategies that minimize I/O cost. This is achieved by *backtracking* index nodes during the search. However, the broadcast order (and, thus, the access order) of index nodes is extremely important in wireless broadcast systems because the data and index are only available to the client when they are broadcast on air. Clients cannot randomly access a specific data object or index node but have to wait until the next time it is broadcast. As a result, each backtracking operation extends the access latency by one more cycle and, hence, becomes a constraint in wireless broadcast scenarios.

Fig. 4 depicts an example. Assume that a nearest neighbor search algorithm based on R-tree [10] first visits the root node, then node R_2, and, finally, R_1, while the

4. Since data objects are typically measured in terms of *pages*, we use it as the unit of tuning time. Thus, the channel capacity and transmission rate do not need to be considered in this study.

5. Both a transition from active mode to doze mode and vice versa are counted as a switch.

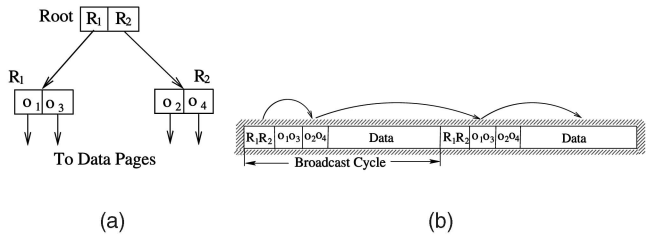


Fig. 4. Linear access on wireless broadcast channel. (a) R-tree index. (b) Branch-and-bound search.

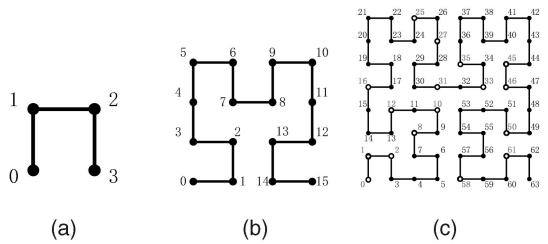


Fig. 5. Hilbert curves of order 1, 2, and 3. (a) H_1. (b) H_2. (c) H_3.

server broadcasts nodes in the order of the root, R_1, and R_2. If a client wants to backtrack to node R_1 after it retrieves R_2, it will have to wait until the next cycle because R_1 has already been broadcast. This significantly extends the access latency and it occurs every time a navigation order is different from the broadcast order.

2.3 Related Work

Several CkNN search algorithms for client-server systems have appeared in the literature [6], [7], [11]. In [11], a sampling technique is employed to perform normal kNN searches at some predefined sampling points and then approximate a range to bound all the possible right answers. However, its accuracy depends pretty much on the predefined sampling points. In order to enable an exact search, Tao and Papadias devised two search algorithms for CkNN queries based on the R-tree. The first algorithm is based on the concept of time-parameterized (TP) queries, which treat a query line segment as the moving trajectory of a query point [7]. Therefore, the k nearest objects to the moving query point are valid only for a limited duration and a new TP query is issued to retrieve new nearest objects once the valid time of the current query expires, i.e., when a split point is reached. While the TP approach avoids the drawbacks of sampling, it is an incremental algorithm that needs to issue nkNN queries in order to obtain the final answer set, where n is the number of split points along the query line segment. The second algorithm, proposed later in [6], navigates R-tree based on certain heuristics. The whole answer set is obtained within one single navigation of R-tree.

More recently, the continuous window (CW) algorithm has been proposed to solve the CkNN problem of moving objects with update [12]. Their focus was on minimizing the update cost caused by the objects continuously changing their positions. Observing that window queries are easier to maintain on moving objects than kNN queries, the CW algorithm filters candidate objects using a within-window query around the query point. Since the within-window query tries to bound at least k objects, only those bounded objects are under consideration when computing the kNN query. Similarly, the research presented in [13], [14] is also motivated by the frequent update operation issued by the moving objects. However, they proposed different approaches, adopting incremental evaluation and shared execution strategies, to address the performance issue. Since the update operation and query processing at server site will be significantly degraded to become the bottleneck of the system performance, some scalable and efficient algorithms have been proposed.

On one hand, all of the proposed algorithms for CkNN search are only suitable for systems with resident storage but not for wireless data broadcast systems, which only support sequential access but not random access. On the other hand, most existing air indices are not designed to answer location-based queries. Our study uniquely addresses the CkNN search problem in wireless data broadcast systems.

3 Search CkNN on Hilbert Curve Air Index

To adapt to the linear streaming property of the wireless data broadcast channel, the *Hilbert Curve (HC) index* was proposed to disseminate location-based data via wireless broadcast [27], [16]. In this section, we first briefly describe the basic idea of the HC index and then introduce the new CkNN search algorithm based on the HC index. The algorithm presented in this paper supports generalized CNN queries, namely, *continuous-k-nearest-neighbor* (CkNN) queries.

3.1 Hilbert Curve Index

A space-filling curve is a continuous path that visits every point in a k-dimensional grid exactly once without crossing itself. Well-known space filling curves, including the Z-curve, the Gray-coded curve, the Peano curve, and the Hilbert curve, are different in the order in which the points in the grid space are visited [26]. The *Hilbert curve* is chosen to build the HC index due to its *optimal locality* [18].

Like other space-filling curves, the Hilbert curve maps points from a multidimensional space to a one-dimensional space. Fig. 5a shows the basic Hilbert curve of order 1. To derive a curve of order i, each vertex of the basic curve is replaced by a curve of order $(i-1)$, which may be strategically rotated and/or reflected to fit the new curve. The Hilbert curves of orders 2 and 3 are depicted in Fig. 5b and Fig. 5c. The numbers, called *index values* in this paper, represent the visiting orders of different points in the Hilbert curve. For example, curve H_2 illustrates a (4×4) grid, where the point $(1, 1)$ has the index value 2. The objects whose positions are denoted by hollow circles in Fig. 5c constitute a running example to be used in later descriptions of our CkNN search algorithm based on the HC index.

Given the mapping function of the Hilbert curve, it is easy for a client to perform a conversion between coordinates and HC index values [19]. Let n be the number of bits assigned to represent a coordinate, the expected time for the conversion is $O(n^2)$. Since n is a preset system constant, the conversion can be done in a constant time. In our previous study, search algorithms for window queries and kNN searches have been developed (please refer to [27] for details).

TABLE 1
Terminology Definition

Notation	Description
$dis_p(q, q')$	Euclidean distance between two points q and q'
$bisc(q, q')$	the perpendicular bisector of line segment connecting points q and q'
$NN(q)$	the nearest neighbor of the query point q
$kNN(q)$	the k nearest neighbors to the query point q, with $kNN(q)[i]$ the ith nearest neighbor to q
$CkNN(\overline{se})$	the answer set containing all the k nearest neighbors to any point in the segment $\overline{se}$
$cir(o, r)$	the circle centered at point o and having r as the radius

3.2 Search Algorithms for Ck$NN Queries

The basic idea for processing a Ck$NN query is to first bound an approximated search range that includes all the candidates. Thereafter, a refining process is conducted to filter out unqualified candidates. The detailed algorithm comprises three steps: 1) Obtain the k-nearest neighbors to the two endpoints of the query line segment. Based on these objects, an approximated search range that bounds all the objects in the final answer set is determined. 2) Obtain a candidate set by issuing a window query based on the approximated search range. 3) Examine the candidate set to obtain the exact answer set. Before going into the details, we first develop two heuristics for processing the Ck$NN search. Table 1 defines the notations for discussion and description of our algorithms.

Heuristic HC1. *For a given query line segment $\overline{se}$, $kNN(s) = \{O_{s_i}, i \in [1,k]\}$ and $kNN(e) = \{O_{e_i}, i \in [1,k]\}$. Then, $\{O_{s_i}, O_{e_i}, i \in [1,k]\} \subseteq CkNN(\overline{se})$.*

Proof. Since s and e are two points on the query line segment, their k-nearest neighbors are part of the final answer set. □

Next, we observe that, if two endpoints of the line segment share the same set of k-nearest neighbor objects, the whole line segment is dominated by those k-nearest neighbors.

Heuristic HC2. *For a query segment $\overline{se}$, if*

$$kNN(s) = kNN(e) = \{O_i, i \in [1,k]\},$$

then $CkNN(\overline{se}) = \{O_i, i \in [1,k]\}$.

Proof. By mathematical induction.

Basic Step. If $k = 1$, the above heuristics can be shown with a *Voronoi Diagram* [20], which partitions a space into disjoint *Voronoi Cells (VCs)* based on locations of data objects in the space. $VC(O_i)$ represents the VC corresponding to the object O_i. For any query point, it must be located within one VC, say $VC(O_i)$, and object O_i must be the nearest neighbor to that query point. As shown in Fig. 1, the Voronoi Diagram partitions the space into five parts denoted by the dashed line, according to the positions of given objects. The shadowed polygon is the corresponding VC of object O_3, i.e., O_3 is the only nearest neighbor to any query point inside the shadowed polygon. Based on computational geometry, VCs are convex [20]. Since $NN(s) = NN(e) = O_1$, both endpoints and, hence, the entire query line segment $\overline{se}$, lie inside the VC of object O_1. Therefore, object O_1 is the nearest neighbor to any query point along the query line segment.

Inductive Step. If we assume that the heuristic is true for $k = m (m \geq 1)$, we are going to prove that it is also true for $k = m + 1$. If the heuristic is not true for $k = m + 1$, there must be at least one point $p \in \overline{se}$ such that $(m+1)NN(p) \neq (m+1)NN(s)$. Since we have known from the assumption that $mNN(p) = mNN(s)$, the $(m+1)$th nearest neighbor of p must be different from that of s (or e). Assume that objects o' and o_{m+1} are the $(m+1)$th nearest neighbors to point p and point s (and e), respectively. If we remove all the objects $o \in mNN(s)$ from the data set, then $NN(s) = NN(e) = o_{m+1}$ and $NN(p) = o'$. Based on the features of the Voronoi Diagram (as shown in the Basic Step), object o_{m+1} must be object o'. As a result, $(m+1)NN(p)$ actually equals $(m+1)NN(s)$ and this inductive step is complete. Thus, by the principle of mathematical induction, the heuristic is true for all $k \geq 1$. □

In Step 1 of the Ck$NN search algorithm, the k-nearest neighbors to the endpoints of the given line segment are obtained and included in the final answer set (based on Heuristic HC1). If both endpoints share the same set of k-nearest neighbors, the final answer set can be returned directly without further processing (based on Heuristic HC2). Otherwise, a radius, guaranteeing at least k objects within that radius to every point along the query line segment, is obtained according to Algorithm 1. Thereafter, a search range bounding all the candidate objects is determined based on Algorithm 2.

Algorithm 1 Finding Maximal Distance
Input: $kNN(s)$, $kNN(e)$, $\overline{se}$, k;
Output: the minimum radius $\mathcal{D}_{max}$;
Procedure:
1: $\phi = kNN(s) \cup kNN(e)$; $cur = s$;
2: $\mathcal{D}_{max} = 0$; $kNN(cur) = kNN(s)$
3: **while** $kNN(cur) <> kNN(e)$ **do**
4: $next.x = e.x$;
5: **for** each object $o \in kNN(cur)$ **do**
6: **for** each object $o' \in (\phi - kNN(cur))$ **do**
7: $p =$ intersection $(bisc(o, o'), \overline{se})$;
8: **if** $(p.x > cur.x)$ and $(p.x < next.x)$ **then**
9: $next = p$;
10: $\mathcal{D}_l = dis_p(kNN(cur)[k], cur)$;
11: $\mathcal{D}_r = max\_dis(kNN(cur), next)$;
12: $\mathcal{D}_{max} = MAX(MAX(\mathcal{D}_l, \mathcal{D}_r), \mathcal{D}_{max})$; $cur = next$;
13: $kNN(cur) =$ findKnn$(cur,$ candidate set $\phi, k)$;
14: $\mathcal{D}_l = dis_p(kNN(cur)[k], cur)$;
15: $\mathcal{D}_r = dis_p(kNN(e)[k], e)$;
16: $\mathcal{D}_{max} = MAX(MAX(\mathcal{D}_l, \mathcal{D}_r), \mathcal{D}_{max})$;
17: return $\mathcal{D}_{max}$;

Algorithm 1 assumes that the kNN objects to the two endpoints, $kNN(s)$ and $kNN(e)$, are known. Those two sets of objects form a candidate set ϕ. It is motivated by the fact that, if a circle around a point p contains k objects

of the set ϕ, this circle must contain at least k objects of the whole data set. As a result, its k-nearest neighbors must be included as well.

The algorithm first checks $kNN(s)$ and $kNN(e)$. If they are different, $\overline{se}$ is not dominated by the same set of objects and there is at least one split point between points s and e. A sweeping line approach is adopted to find the split point(s), which works as follows: For any object o from $kNN(s)$ and any object o' from $(\phi - kNN(s))$, the intersection between the bisector $bisc(o, o')$ and query segment $\overline{se}$, denoted by q', is the point by passing which object o' will replace o as one of the kNN objects to point q'. Among those intersections, the one m having the shortest distance to point s is the next split point. Segment $\overline{sm}$ is dominated by $kNN(s)$ and the process continues with segment $\overline{me}$ as the query segment until the $kNN(m)$ contains the same objects as $kNN(e)$.

The split points partition the whole query segment $\overline{se}$ into several subsegments, with each dominated by a set of objects. For a subsegment l, the maximal distance, denoted by $\mathcal{D}$, between any point of l and its kNN objects could be detected (please refer to Claim 1 for the detailed theoretical proof). Among all the detected $\mathcal{D}$s, $\mathcal{D}_{max}$ is set to the maximal one. For any point p along the segment $\overline{se}$, it is guaranteed that the circle centered at p with $\mathcal{D}_{max}$ as its radius will contain at least k objects.

Claim 1. *Let $\mathcal{D}_s(k)$ be the distance between s and its kth NN object $kNN(s)[k]$, $\mathcal{D}_e(k)$ be the distance between e and its kth NN object $kNN(e)[k]$, and $\mathcal{D}(k)$ be the maximal of $\mathcal{D}_s(k)$ and $\mathcal{D}_e(k)$. If $kNN(s) = kNN(e)$, $\forall q \in \overline{se}$, $dis_p(q, kNN(q)[k]) \leq \mathcal{D}(k)$.*

Proof. By mathematical induction.

Basic Step. When $k = 1$, the fact that $NN(s)$ equals $NN(e)$ means that both endpoints s and e are within the Voronoi Cell of the object $NN(s)$. Therefore, object $NN(s)$ must be the nearest object to any point q on the segment $\overline{se}$. Since

$$dis_p(q, NN(s)) \leq MAX\big(dis_p(e, NN(s)), dis_p(s, NN(s))\big),$$

the claim is true.

Inductive Step. Let us assume that the claim is true for k and prove that it is true for $(k+1)$. Suppose $kNN(s) = \{o_i, i \in [1, k]\}$ and

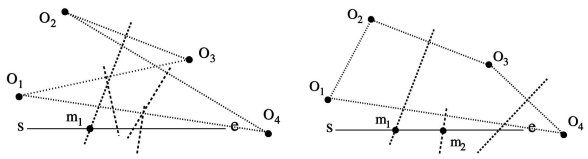
$$(k+1)NN(s) = kNN(s) \cup \{o_{k+1}\}.$$

Randomly selecting a point q from the segment $\overline{se}$, object $o' \in (k+1)NN(s)$ is the farthest one away from q compared with the other objects within $(k+1)NN(s)$. If $o' \in kNN(s)$, $dis_p(q, o') \leq \mathcal{D}(k)$ (based on the assumption). Since $\mathcal{D}(k+1) > \mathcal{D}(k)$, $dis_p(q, o')$ must be smaller than $\mathcal{D}(k+1)$. Otherwise, o' must be object o_{k+1}. Based on the distance between a given point and any point of a line segment,

$$dis_p(q, o_{k+1}) \leq MAX(dis_p(s, o_{k+1}), dis_p(e, o_{k+1})) = \mathcal{D}(k+1).$$

The inductive step is complete.

As a result, for any query point q on the segment $\overline{se}$, at least k objects in the set $kNN(s)$ are within the circle $cir(q, \mathcal{D}(k))$. Therefore, Claim 1 is proved. □



cur	$2NN(cur)$	$2NN(e)$	$\mathcal{D}_{max}$
s	$\{O_1, O_2\}$	$\{O_3, O_4\}$	$dis_p(s, O_2)$
	report: $\overline{sm_1}$ is dominated by $\{O_1, O_2\}$		
m_1	$\{O_1, O_3\}$	$\{O_3, O_4\}$	$dis_p(s, O_2)$
	report: $\overline{m_1 m_2}$ is dominated by $\{O_1, O_3\}$		
m_2	$\{O_3, O_4\}$	$\{O_3, O_4\}$	$dis_p(m_2, O_4)$
	report: $\overline{m_2 e}$ is dominated by $\{O_3, O_4\}$		

(c)

Fig. 6. Determination of maximal distance D_{max}. (a) Detection of m_1. (b) Detection of m_2. (c) Intermediate results.

Fig. 6 shows an example (where $k = 2$) to illustrate the process of finding D_{max}. Before the algorithm starts, 2NN objects to both endpoints are detected, i.e., $2NN(s) = \{O_1, O_2\}$ and $2NN(e) = \{O_3, O_4\}$. Therefore, the candidate set ϕ contains four objects. First, the algorithm starts with s as the cur point. Two objects, O_1 and O_2, are in the set $2NN(cur)$ and another two objects, O_3 and O_4, form the set $(\phi - 2NN(cur))$. As depicted in Fig. 6a, there are four intersections. Point m_1 is chosen as the next split point since it is closest to the cur point. Next, the algorithm continues with m_1 as the cur point. Similarly, the intersections are found and the one, m_2, having the shortest distance to cur point is set as the next split point (see Fig. 6b). Since $2NN(m_2)$ equals $2NN(e)$, the algorithm is finished and the $dis_p(m_2, O_4)$ is returned as the detected maximal distance $\mathcal{D}_{max}$. Fig. 6c illustrates the steps for finding split points and D_{max}.

Algorithm 2 CkNN Search Range
Input: $kNN(s)$, $kNN(e)$, $\overline{se}$, k;
Output: search range $\mathcal{R}$;
Procedure:
1: let P_l be the left-most point with $P_l \in kNN(s) \cup kNN(e)$;
2: let P_r be the right-most point with $P_r \in kNN(s) \cup kNN(e)$;
3: $radius =$ Finding Maximal Distance $(kNN(s), kNN(e), \overline{se}, k)$;
4: $P_1.x = P_l.x$; $P_1.y = P_l.y + radius$;
5: $P_2.x = P_l.x$; $P_2.y = P_l.y - radius$;
6: $P_3.x = P_r.x$; $P_1.y = P_r.y + radius$;
7: $P_4.x = P_r.x$; $P_2.y = P_r.y - radius$;
8: return the rectangle $\mathcal{R}$ bounded by P_1, P_2, P_3, and P_4;

Given the query line segment and $\mathcal{D}_{max}$, an intuitive approach to determine the final search range is to incorporate all the search circles, as shown in Fig. 7a. Because the answer set $kNN(s)$ has already been detected, the left side of the search range could be further shrunk to the smallest x-coordinate of the found objects, i.e., O_1. Similarly, the right side could also be refined by the largest x-coordinate of the found objects, i.e., O_4. Fig. 7b shows the

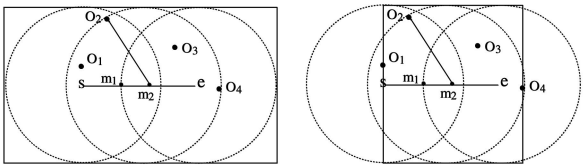


Fig. 7. Approximated search range for C2NN. (a) Before refinement. (b) After refinement.

final search range which is represented by the solid-line rectangle. Algorithm 2 gives the pseudocode.

Once the search range is returned, a window query is issued to complete the second step, i.e., retrieving the candidates set. The only step left is filtering, i.e., how to distinguish the real answers from the rest. Employing the sweeping line approach used in Algorithm 1, split points are detected one by one and the query line segment is partitioned into smaller subsegments with each dominated by a set of objects (based on Heuristic HC2). The detailed pseudo-code is provided by Algorithm 3 and the proof to verify that the algorithm will return and only return right answers is shown in Claim 2. Finally, Algorithm 4 summarizes those three steps and gives a complete description of the search algorithm.

Algorithm 3 Filtering
Input: query line segment $\overline{se}$, $kNN(s)$, $kNN(e)$, candidate answer set ϕ, k;
Procedure:
1: $s'.x = e.x$;
2: **for** each object o in $kNN(s)$ **do**
3: **for** each object o' in $(\phi - kNN(s))$ **do**
4: $p = \text{intersection } (bisc(o, o'), \overline{se})$;
5: **if** $(p.x > s.x)$ and $(p.x < s'.x)$ **then**
6: $s' = p$; $O_{rep} = o$; $O_{next} = o'$;
7: $kNN(s') = (kNN(s) - \{O_{rep}\}) \cup \{O_{next}\}$;
8: denote segment $\overline{ss'}$ is dominated by $kNN(s)$;
9: Filtering(segment $\overline{s'e}$, $kNN(s')$, $kNN(e)$, ϕ, k);

Claim 2. *Algorithm 3 will return and only return qualified objects.*

Proof. Assume that there is an object $o \in CkNN(se)$ which is not detected by Algorithm 3. Since object o is part of the answer set, there must be at least one point $p \in \overline{se}$ with $o \in kNN(p)$. As explained, the filtering algorithm is completed only when the segment $\overline{se}$ is successfully partitioned into several subsegments with each fully dominated by one set. Consequently, the subsegment containing point p will also have a corresponding set η containing its k-nearest neighbors. According to Claim 1, object o must be bounded by the search range and, hence, inside the candidate set. Therefore, the detected answer set η must cover the object o and our assumption is not satisfied. On the other hand, the returned object must be one of the kNN objects to at least one point q along the segment $\overline{se}$. Therefore, it must be part of the answer set. The proof is completed. □

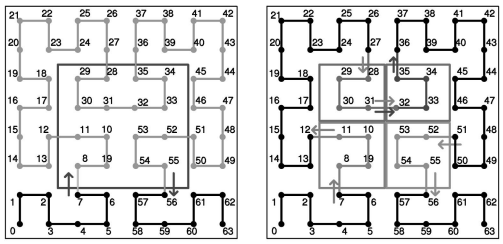


Fig. 8. Improvement introduced by search space partition. (a) Before partition. (b) After partition.

Algorithm 4 CkNN Search
Input: query line segment $\overline{se}$, k, data set $\mathcal{S}$;
Procedure:
1: $kNN(s) = \text{findKnn}(s, \mathcal{S}, k)$; $kNN(e) = \text{findKnn}(e, \mathcal{S}, k)$;
2: **if** $(kNN(s) == kNN(e))$ **then**
3: report that segment $\overline{se}$ is dominated by $kNN(s)$;
4: **else**
5: $\mathcal{R} = \text{C}k\text{NN Search Range}(kNN(s), kNN(e), \overline{se}, k)$;
6: $\phi = \text{Window Query}(\mathcal{R}, \mathcal{S})$;
7: $\text{Filtering}(kNN(s), kNN(e), \overline{se}, \phi)$;

4 Implementation Issues

In this section, we discuss two enhancements of HC index implementation, query partitioning and index organization, to facilitate efficient processing of CkNN search. The former strategy allows clients to partition the query window into smaller subwindows. As each subwindow can be covered by the Hilbert curve with a lower order, the spatial locality is improved. As a result, the tuning time performance is improved. The latter aims at obtaining optimal access latency. The HC index is proposed to answer multiple location-based queries. As different queries request various number of scans of the index, different index organizations can be used. We develop a theoretical model to analyze the access latency performance under different index organizations. Therefore, an index segment can include various B^+-trees. If the detailed access patterns of clients are available, the organization that produces the optimal access latency can be derived.

4.1 Query Partitioning

The data locality on the Hilbert curve has a great impact on the performance of query processing. If the nearby points in the original search space are rather far away from each other along the curve, the search range of the window query will be relatively large and, hence, many points need to be checked. A motivating example is depicted in Fig. 8a in which the solid rectangle represents a query window. Employing the window search algorithm, all the points with index values between 8 and 55 need checking. Obviously, this range contains many points outside the window.

It can be observed from Fig. 5 that the order i curve is derived from order $(i-1)$ curves. If a query window crosses several order $(i-1)$ curves, it has a higher

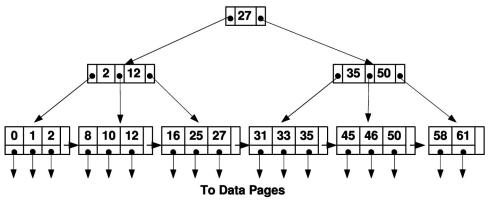


Fig. 9. B^+-tree of Hilbert Curve values.

probability to contain many more points than necessary due to the low locality of the points near the boundary of the $(i-1)$ curves. Therefore, one solution is to partition the query window into subwindows. If each subwindow can be covered by curves of order $(i-1)$, or even lower orders, the search range could be dramatically refined. Suppose the client partitions the window into four subwindows as shown in Fig. 8b; then, the new search range only consists of objects whose HC values fall within ranges [8, 11], [28, 35], or [52, 55].

Each client, based on its available resource and capability, determines the partition degree. For example, a client may take a 2×2 partition and process a window query based on four order $(i-1)$ curves, where i is the order of original curve covering the whole search space. In short, the client can apply $2^p \times 2^p$ partition against 2^p order $(i-p)$ curves.

4.2 Index Organization

A B^+-tree is employed to store HC index values [21]. Given a set of data objects, the B^+-tree is built bottom up. The fan-out of the tree node is decided by the page capacity. The leaf nodes of the B^+-tree contain the sorted HC values of all the objects along with corresponding pointers, which facilitates the continuous access at the leaf level. Each object is indexed and identified by its HC index value, and the associated pointer presents the broadcast time of the data page containing the object itself. If the fan-out is three, the B^+-tree for the running example is depicted in Fig. 9.

Based on the HC index, different types of queries require different number of index scans. A window query can be answered by scanning the HC index once, while a kNN query requires two scans and a CkNN query needs to scan the HC index three times. Based on this observation, several alternatives can be considered in organizing the index segments. One solution is to include an original B^+-tree without any alternation. The advantage of this scheme is the simplicity and small index storage cost. In this case, however, the kNN and CkNN search process may need to go across two and three index segments, respectively. If some data objects in the final answer set are broadcast between the scanned index segments, they will be missed and, hence, significantly extend the access latency (since the clients have to wait until the next cycle to retrieve them). The second solution is to duplicate the B^+-tree by broadcasting it twice within the same index segment, which ensures that the kNN searches can be finished within one (longer) index segment. Although this solution doubles the size of the index segment compared with the first solution, it saves the client's average access latency for kNN searches (and also the CkNN queries). The last solution is to replicate the B^+-tree three times within one index segment. This enables clients to answer all the supported queries by scanning an index segment only once. To serve multiple types of location-based queries, we have to consider the overall performance. Thus, in the rest of this section, we conduct a performance analysis of different organization schemes for different queries. Notations used in the derivation are summarized in Table 2.

TABLE 2
Notation for Analysis of Index Organizations

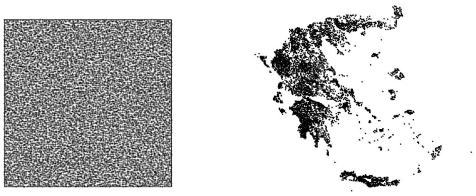
Notation	Definition
D	the number of pages used to include all the objects information
I	the number of pages forming one index segment
m^*	optimal m (i.e., $\sqrt{D/I}$) to minimize the average access latency [4]
$Cycle$	the number of the pages, including both data and index, within one broadcast cycle, $Cycle = D + m^* \times I$
lat	the average latency which is the duration between the client's issuing a query and the client's receiving answers
$\{w, k, c\}$	superscripts representing window queries, kNN queries, and CkNN queries, respectively
p	the possibility that some answers are missed due to the fact that clients could not find the answer by scanning the index segment once
q	the possibility that a client will issue a particular kind of query
$\lvert k\rvert$	the number of objects asked to be returned by a kNN query
$\lvert c\rvert$	the number of objects returned by a CNN query

The following analysis is conducted based on the assumption that each object has the same access probability and clients tune into the channel randomly. As we mentioned before, index organization schemes only affect the average access latency, but not the tuning time. Consequently, we only include the cost model of access latency, as shown in (1). Subscript $i \in [1,3]$ stands for repeating time of the B^+-tree within one index segment.

$$\begin{aligned} lat_i^t = {} & \frac{1}{2} \times \left(I_i + \frac{D}{m_i^*} \right) + \frac{1 - p_i^t}{2} \times Cycle_i \\ & + p_i^t \times \left(1 + \frac{n^t - i}{m_i^*} \right) \times Cycle_i, \\ & i \in [1,3], \quad t \in \{w, k, c\}, \quad n^k = 2, \quad n^c = 3, \end{aligned} \tag{1}$$

$$p_i^t = \begin{cases} 0 & \text{if } t = w \text{ and } i \in [1,3] \\ \left(1 - (1 - \frac{1}{m^*})^{|k|}\right) / \binom{m^*}{1} & \text{if } t = k \text{ and } i = 1 \\ 0 & \text{if } t = k \text{ and } i \in [2,3] \\ \left(1 - (1 - \frac{(3-i)}{m^*})^{|c|}\right) / \binom{m^*}{1} & \text{if } t = c \text{ and } i \in [1,2] \\ 0 & \text{if } t = c \text{ and } i = 3. \end{cases}$$

The first term of the equation is the latency of the initial probe, i.e., the period starts when the client first tunes into the broadcast channel till the moment it receives the first index segment. The second term of the equation is the average waiting time for the answers which are assumed to be uniformly distributed within the broadcast cycle under the situation that p is zero. Since some searches cannot be completed within one index segment, the clients have to



(a) (b)

Fig. 10. Data sets for performance evaluation. (a) UNIFORM. (b) REAL.

probe in the next or even the following index segments to finish the queries. This is caused by the possibility that some answers are broadcast between the scanned index segments and, thus, are missed from the current broadcast cycle ($p \neq 0$). Consequently, the clients suffer from waiting until the next broadcast cycle to receive those answers. Given a distribution of the queries submitted by the clients, we can approximate the average performance of different indices based on the expected access latencies of different schemes. The expected access latency is derived as in (2).

$$lat_i = q^w \times lat_i^w + q^k \times lat_i^k + q^c \times lat_i^c \quad i \in [1,3] \text{ and } q^w + q^k + q^c = 1.0. \tag{2}$$

5 Performance Evaluation

This section evaluates the performance of the proposed CkNN search algorithm on HC Index. Two data sets, as shown in Fig. 10, are used in the evaluations. In the first data set (UNIFORM), 10,000 points are uniformly generated in a square euclidean space. The second data set (REAL) contains around 6,000 cities and villages of Greece, which is extracted from the point data set available in [22]. The simulation model is implemented using *CSIM*, a discrete-event-based simulation package (www.mesquite.com).

We implement the CkNN search algorithms based on R-tree for comparison [6]. As explained in Section 2, the algorithm is designed for disk-based R-tree. Consequently, it is not suitable for direct use in wireless data broadcast due to the backtracking operations which may incur a longer access latency. Thus, we make the following changes to adapt the algorithm to the wireless data broadcast environment: In short, the branches of R-tree are visited sequentially based on broadcast order, rather than on the order dynamically determined by heuristics. As the locations of data objects are known a priori, the STR packing scheme is employed to build R-tree in order to achieve the best performance (denoted as *STR R-tree* in the later presentation) [23]. Furthermore, the Hilbert curve is also adopted as an alternative in packing R-tree [24], which is also implemented in our simulation (denoted as *HC R-tree*).

R-tree, including both STR R-tree and HC R-tree, is broadcast in a depth-first order because breath-first broadcast requires a client to maintain a queue during query processing to keep track of the distance information between the query point and all the nodes in the same level in order to prune the unnecessary branches. The queue requires a large memory space, which may not be available to all the mobile devices. As for the HC index, B^+-tree is broadcast in a breath-first order to facilitate the continuous access of linked leaf nodes.

TABLE 3
System Parameters

Parameter	Setting	Parameter	Setting
Capacity	256 bytes	*Bandwidth*	256 kpbs
DataSize	1024 bytes	*FloatSize*	4 bytes
PointerSize	2 bytes	*QueryLengthRatio*	0.1

System parameters for our evaluation are defined in Table 3, together with the default settings. Unless explicitly stated, the default values are used. The fixed size of a page is denoted as *Capacity* and the available bandwidth of the public channel is denoted by *Bandwidth*. The content body of each object occupies *DataSize* bytes. Two floating-point numbers are used to represent a two-dimensional coordinate, and the same number of bits are used for an HC index value. The size of a floating-point number is *FloatSize*. The bandwidth occupied by a pointer is denoted by *PointerSize*. Finally, *QueryLengthRatio* defines the ratio of the length of query line segments used in CkNN searches to the side length of the whole search space.

The ability to address the scalability issue is a major strength of wireless broadcast systems. However, we only model one client to capture the client-side processing in our simulation since the number of clients does not affect the system performance (i.e., an arbitrary number of clients can access the broadcast at the same time and there are no requests/feedback sending from clients to the server). One million queries are issued randomly in our simulation and the average performance is shown in the following figures. The tuning time presented here is the active period to search in the index. Given a fixed *Bandwidth* and *DataSize*, the tuning time for retrieving qualified objects is a constant and is independent of indices deployed. Therefore, the cost for object retrieval is not presented in the simulation. Similarly, the initial probe is not counted because its average again is a constant. Each index segment under R-tree, either STR R-tree or HC R-tree, contains one R-tree, while, under the HC index, it consists of three B^+-trees. Although there are other index organization schemes as described in Section 4.2, we duplicate B^+-tree three times to simplify the description. Access latency under different organization schemes will be presented later in Section 5.5.

The performance evaluation is organized as follows: Section 5.1 first validates the enhancement achieved by query partitioning. Section 5.2 evaluates the tuning time performance of CkNN search algorithms under different indices. Section 5.3 examines the access latency of CkNN search algorithms by varying bandwidth overhead preallocated for index segments within a broadcast cycle. This experiment provides important guidance from the system planning perspective since the service providers may choose to allocate a fixed budget of bandwidth for indexing. On the other hand, it is important to find out how the CkNN search algorithms (and indices) fare with each other when the constraint of index bandwidth preallocation is released. Thus, Section 5.4 evaluates the access latency performance under the optimal configuration of bandwidth allocation for index and data segments. The access latency presented in the previous two sections is based on the assumption that three B^+-trees form an index segment, which provides the worst-case access latency. In Section 5.5, we assume a

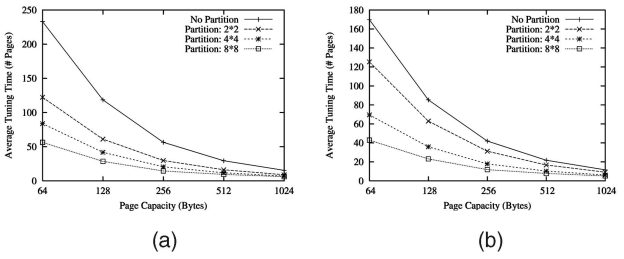


Fig. 11. Improvement of query partitioning. (a) UNIFORM. (b) REAL.

general scenario in which clients may submit different kinds of queries, including window queries, kNN queries, and CkNN queries. The access latency under different index organizations and various access patterns is thereafter presented. Finally, we extend the simulation to evaluate the extra energy overhead caused by switches, which tends to be neglected by existing work in the literature.

5.1 Query Partitioning

Window query is a fundamental operation for the processing of complex location-based queries (e.g., kNN and CkNN) on the HC index. The performance of window queries based on different degrees of query partitioning is depicted in Fig. 11. Here, the size of the square query window is set to 1 percent of the whole search space. A $2^p \times 2^p$ partition will check against several Hilbert curves of order $(i - p)$ rather than the original curve of order i. This experiment shows that query partitioning does improve the tuning time (as shown in the figure) but has no effect on access time (figure not shown to save space). Using the original HC index (denoted by *No Partition* in the figure) as the baseline for comparison, partition 2×2, 4×4, and 8×8 consume only 53, 37, and 29 percent of the tuning time of No Partition on UNIFORM. A similar result is also observed from the REAL data set. For the rest of the experiments, Partition: 8×8 is adopted as the default partitioning scheme.

Since the clients have to determine the boundary index values for multiple subquery windows when employing query partitioning, there is a small computational overhead incurred at clients. It is worth noting that the number of subqueries incurred due to query partitioning is not proportional to the number of subspaces corresponding to Hilbert curves of lower order. This is because some windows may not overlap with every subspace. Our experiments show that an average of 1.23 subwindow queries are produced per window query in Partition 2×2 and 1.54 subwindow queries are produced per window query in Partition 4×4.

5.2 Tuning Time Performance

In this section, we present the average tuning time of answering a CkNN query based on different indices. In addition to system parameters such as page capacity, tuning time performance is dependent on the index structures adopted, search algorithms, and parameters specified in a query. However, the index organization models, i.e., how the index segment is interleaved with data segments, do not affect tuning time. Thus, in this section, we compare the tuning time of different indices by varying the page capacity, number of nearest objects k, and the *QueryLengthRatio*, separately.

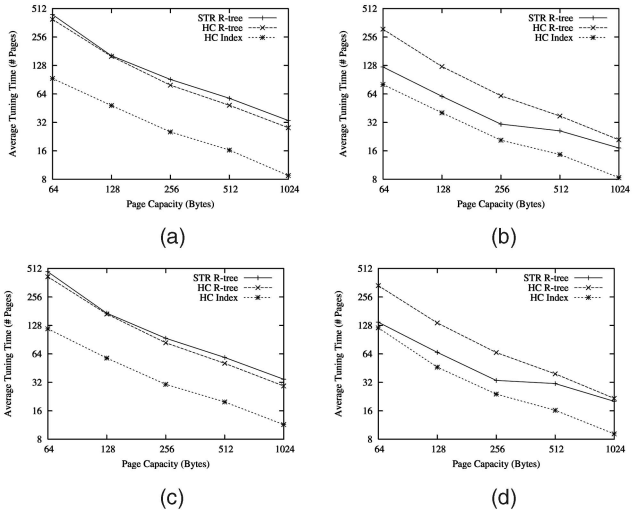


Fig. 12. Tuning time of CkNN queries versus *Capacity*. (a) UNIFORM ($k = 1$). (b) REAL ($k = 1$). (c) UNIFORM ($k = 5$). (d) REAL ($k = 5$).

We first perform a sensitivity test on page capacity. In this experiment (see Fig. 12), the *QueryLengthRatio* is set to 0.1 and k is set to 1 and 5. The HC index outperforms STR R-tree and HC R-tree significantly on UNIFORM. As shown in Fig. 12a, the HC index consumes only 27 percent and 30 percent of the tuning time consumed by the STR R-tree and HC R-tree, respectively. The HC index also outperforms the R-tree variants noticeably when the data distribution is not uniform. As shown in Fig. 12b, the HC index consumes only 61 percent and 34 percent of the tuning time consumed by STR R-tree and HC R-tree, respectively. When $k = 5$, the tuning time under the HC index still performs constantly better than that of other indices. Take the REAL data set as an example; the HC index consumes only 65 percent and 38 percent of the tuning time of STR R-tree and HC R-tree.

Next, we study the impact of query line length on tuning time by setting $k = 1$ and $Capacity = 256$ bytes. In this experiment (see Fig. 13), the *QueryLengthRatio* is varied from 0.025, 0.05, 0.1, to 0.2. As expected, more page accesses are requested when the length of the query line increases. This is because a CkNN query with a longer query line asks for more objects and, hence, needs a larger search space. The HC index performs consistently well. It consumes only 28 percent and 32 percent of the average tuning time consumed by STR R-tree and HC R-tree, respectively, on UNIFORM. The HC index on average consumes 75 percent

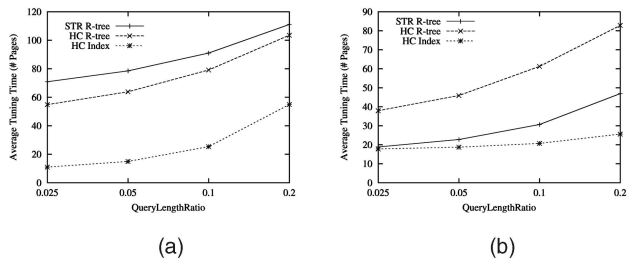


Fig. 13. Tuning time of CNN queries versus *QueryLengthRatio*. (a) UNIFORM. (b) REAL.

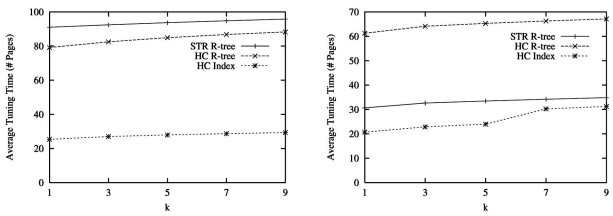


Fig. 14. Tuning time of $C k\mathrm{NN}$ queries. (a) UNIFORM. (b) REAL.

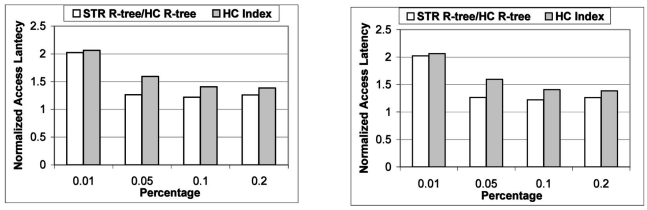


Fig. 15. Access latency versus various *percentages*. (a) UNIFORM. (b) REAL.

and 38 percent of the tuning time consumed by STR R-tree and HC R-tree, respectively, on the REAL data set.

Finally, Fig. 14 shows the experimental results on different data sets by fixing $Capacity = 256$ bytes and $QueryLengthRatio = 0.1$, and varying k from 1 to 9. As shown, the tuning time of all indices is not very sensitive to k; it only slightly increases as k increases. Taking UNIFORM as an example, when k is increased from 1 to 9, the tuning time performance on average is reduced by 10 percent for all three indices. This experiment also shows that the HC index has a better performance for complex $C k\mathrm{NN}$ queries in comparison with STR R-tree and HC R-tree. For UNIFORM, the HC index consumes 30 percent the tuning time of STR R-tree and 33 percent tuning time of HC R-tree, respectively. For the REAL data set, it consumes 90 percent tuning time of STR R-tree and 46 percent tuning time of HC R-tree, respectively.

5.3 Fixed Bandwidth Allocation for Index

While the primary goal of this study is to minimize energy consumption at mobile clients, the air index has a significant impact on access latency which is highly dependent on how much bandwidth is available for index segments and how index segments are interleaved with data segments. From the perspective of system deployment and operations, fixed amounts of bandwidth may be budgeted for index segments; thus, in this section, we examine the access latency performance of $C k\mathrm{NN}$ algorithms given a predetermined index bandwidth (which implicitly determines the number of index segments to be interleaved with data segments in a broadcast cycle).

In this experiment, we vary *Percentage*, the ratio of preallocated index bandwidth to overall *Bandwidth*, to test its impact on access time. In other words, index segments within one broadcast cycle occupy *Percentage***Bandwidth* bandwidth. Therefore, m, the number of index segments broadcast in a broadcast cycle, can be derived accordingly. Since the replication of index segments does not have an impact on the tuning time resulted from index scanning, we do not plot the figures for tuning time.

We take an estimated optimal access time, i.e., a half of the broadcast cycle which consists of only data objects, as the base line for comparison. For clarity of presentation, the experimental results are normalized based on this optimal access time. As shown in Fig. 15, both R-tree and the HC index share a similar access latency when *Percentage* is very small. This is because a small *Percentage* results in a small m. For example, when *Percentage* is 0.01, both R-tree and the HC index have only one index segment (i.e., $m \approx 1$). Therefore, different indices share similar initial probe time and broadcast cycle and, hence, access latency. As *Percentage* increases, more index segments are broadcast within one cycle to reduce the initial probe time. Although a large m incurs a longer broadcast cycle, the client benefits more from the improved initial probe time. However, when m reaches a certain value, the lengthened broadcast cycle starts to dominate the access latency and, hence, causes a longer access latency. It is also observed that an index with smaller size has a more frequent appearance within one cycle. R-tree, owning to the smaller size, has a better access latency than the HC index.

5.4 Access Latency under an Optimal Configuration

In this section, we examine the access time of compared search algorithms and indices under optimal configuration of index and data bandwidth allocation. The minimal access latency under an optimal m (denoted by m^*) is presented in Fig. 16. In contrast to Section 5.3, here, we assume that sufficient bandwidth is available for broadcasting the index segment m^* times within a broadcast cycle. Fig. 16 shows the access latency under different page capacities. The query parameters examined previously, i.e., the number of nearest objects and the length of query line, have no effect on access time and, thus, are not discussed here.

As shown in the figure, indices expand the broadcast cycle and result in a longer access latency. Take UNIFORM as an example; the R-tree index, including both STR R-tree and HC R-tree, causes a 23 percent access latency overhead, while the HC index incurs a 39 percent access latency overhead. Compared to the experiment discussed in Section 5.3, the access latency under an optimal configuration is obviously better than the access time resulting from predetermined index bandwidth allocation.

5.5 Access Latency under Different Index Organizations

Based on the performance presented in previous sections, R-tree has a better access latency than the HC index. This is

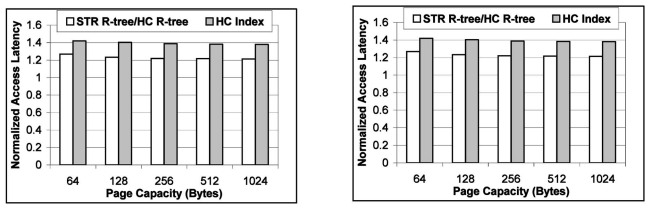
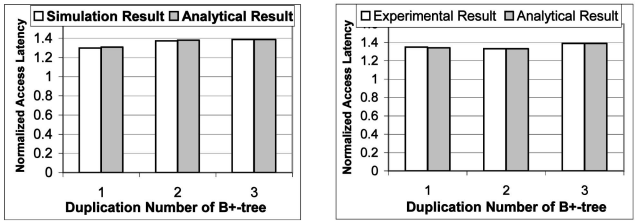


Fig. 16. Access latency under optimal configuration. (a) UNIFORM. (b) REAL.



(a) (b)

Fig. 17. Access latency versus index organization. (a) $p_w = p_k = p_c = \frac{1}{3}$. (b) $p_k = \frac{1}{2}$, $p_w = p_c = \frac{1}{4}$.

because we duplicate B^+-tree three times within one index segment, which significantly increases the occupied bandwidth of the index. Therefore, the presented access latency for the HC index is the worst case scenario. In real applications, each index segment under the HC index can broadcast B^+-tree once or twice, but not necessarily three times, to shorten the access latency. As described in Section 4.2, the number of B^+-trees included in an index segment is determined based on the detailed access patterns.

In Fig. 17, the performance of access latency under different organization schemes is presented. To save the space, only the results based on UNIFORM are depicted. We further assume there is enough bandwidth allocated to index segments and, hence, the access latency is evaluated based on the optimal m. As expected, the simulation results demonstrate a perfect match with analytical results derived from (1) and (2). The average difference between simulation results and analytical results is around 4 percent, which further verifies the accuracy of the analytical model. In the simulation, all the default system parameters are applied, and the k for kNN search is one, window size is 0.01 of the original search space, and k for CkNN search is again one.

It is obvious as access patterns change that the optimal index organization scheme that minimizes the average access latency also varies. For example, when a uniform access pattern is applied (i.e., $p_w = p_k = p_c$, as shown in Fig. 17a, the index organization to broadcast B^+-tree once within the index segment optimizes the performance. However, if most of the queries issued are NN searches, the index segment to duplicate B^+-tree twice works the best (as shown in Fig. 17b). Therefore, the analytical model developed in Section 4.2 provides the system administrator a guidance to select the corresponding index organization model.

5.6 Switching Overhead

Air indexing techniques are based on the idea of keeping the mobile clients in doze mode as long as possible and only switching them back to active mode when the data of interest are broadcast. While most of the existing work in the literature neglected the energy overhead caused by mode switches, there is a general concern of whether the energy saving obtained from turning mobile devices into doze mode can outweigh the switching overhead, particularly when traversing an index will obviously incur many switches.

In order to verify that air indexing techniques do save energy, we conduct a simulation to count the average number of switches incurred by a query in this section. In this simulation, a transition from active mode to doze mode and vice versa are both counted as switches. This experiment also allows us to compare the impact of switches on various indexing schemes and search algorithms.

TABLE 4
Number of Switches under CNN Search
($QueryLengthRatio = 0.1$)

Capacity(Bytes)		64	128	256	512	1024
UNIFORM	STR R-tree	436	117	49	25	14
	HC R-tree	305	82	36	16	10
	HC Index	48	34	21	17	12
REAL	STR R-tree	143	55	26	15	11
	HC R-tree	250	63	25	12	7
	HC Index	43	31	19	16	10

Table 4 summarizes the average number of switches for answering CkNN queries. We observe that, in most cases, the CkNN search algorithm based on the HC index results in a much smaller number of switches when compared to the CkNN algorithms based on STR R-tree and HC R-tree. This is because the B^+-tree in the HC index is broadcast in breath-first order. A window query which retrieves objects within a linear range can be answered by traversing the B^+-tree from the root node to the leaf nodes once. Since range search under the HC index is deterministic, the number of switches is only dependent on the height of the tree. Once a leaf node which covers the smallest HC value of a range is reached, the rest of the objects within the range are sequentially obtained from the leaf nodes. Since window query is the fundamental operation commonly used in other queries (e.g., kNN and CkNN), the number of switches is independent of k in kNN queries or the length of the query line segment for CkNN queries. Taking the UNIFORM data set as an example, the HC index only incurs 47 percent switches of STR R-tree and 68 percent switches of HC R-tree.

To more precisely compare the performance of the CkNN search algorithms under evaluation, we obtain their average power consumptions on UNIFORM (see Fig. 18). It is observed that the typical setup time for a mobile device to start or tune into active mode is in the order of $100\ \mu s$ [9]. Since the setup time is device dependent, we use different setup times, ranging from $100\ \mu s$, $1\ ms$, to $10\ ms$. We assume that *Capacity* is 256 bytes and the *Bandwidth* is 256 kbps. As a result, receiving a page takes $8\ ms$. When setup time reaches $10\ ms$, a switch takes a longer time than receiving a page. This can be regarded as the worst-case scenario. Let P_{on} and P_{off} denote the power consumption in active and doze modes, respectively. Also, let T_{on}, T_{off}, and T_{set} denote the time spent in active, doze, and setup (i.e., switching) during CkNN query processing. Finally, we use N_{swi} to denote the number of switches incurred. We adopt the numbers from the Hobbit chip [4], which consumes $250\ mW$ in the active mode and $50\ \mu W$ in the doze mode, and Proxim RangeLAN2 [9], which requires $1.5\ W$ in transmit mode, $0.75\ W$ in receive mode, and $0.01\ W$ in doze mode, to approximate the energy consumption in answering a CkNN query. Here, we only consider the power consumed in searching the index, denoted by $\mathcal{P}$ in (3), excluding the power consumed in the initial probe step

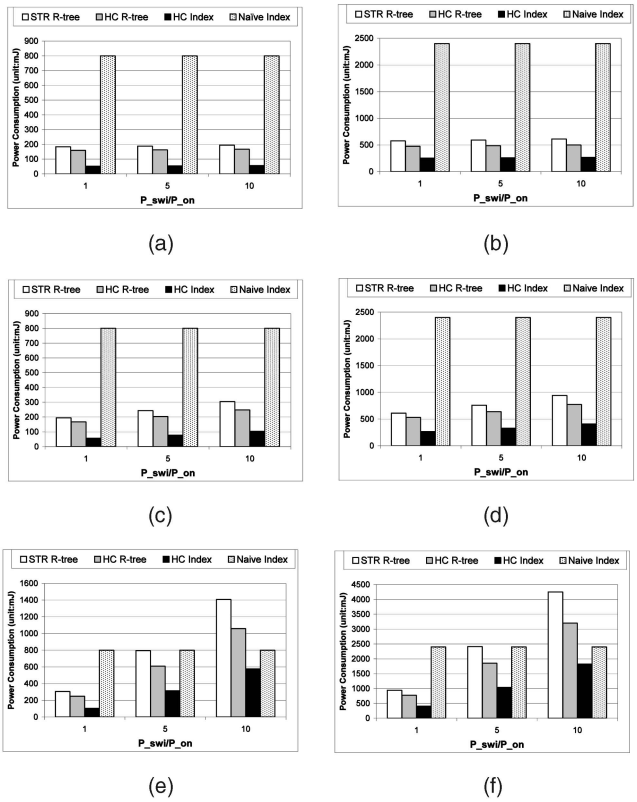


Fig. 18. Power consumption (*Capacity* = 256 bytes, UNIFORM). (a) $T_{set} = 100\ \mu s$, Hobbit. (b) $T_{set} = 100\ \mu s$, Proxim. (c) $T_{set} = 1\ ms$, Hobbit. (d) $T_{set} = 1\ ms$, Proxim. (e) $T_{set} = 10\ ms$, Hobbit. (f) $T_{set} = 10\ ms$, Proxim.

and the object retrieval step. Since different devices have different power consumptions in switching, we assume that P_{swi} equals $\alpha \times P_{on}$, where α is set to 1, 5, and 10.

$$\mathcal{P} = T_{on} \times P_{on} + T_{set} \times N_{swi} \times P_{swi} + T_{off} \times P_{off}. \quad (3)$$

Wireless data broadcast with no index obviously will consume more energy than broadcast with an index. In the experiment, we adopt a simple indexing mechanism which does not require any switching (denoted as *Naive Index*) as the base line for comparison. This index is formed by a set of 2-tuple $\langle p_i, t_i \rangle$, with $i \in [1, N]$, where p_i is the location of object O_i and t_i is the broadcast time of O_i. Therefore, the naive index occupies $\mathcal{N} = N \times (2FloatSize + PointerSize)/Capacity$ pages. Under this scheme, mobile clients process CkNN queries by scanning those $\mathcal{N}$ pages to obtain the broadcast time of the qualified data objects.

The simulation result reveals some important findings. First, air indices do reduce power consumption. Although CkNN searches based on various indices require switches, they significantly reduce the number of pages that need to be downloaded. Therefore, the power saving obtained by retrieving only relevant pages is far more than the extra power incurred by switches. Second, the HC index provides the best performance in terms of power consumption because the CkNN search algorithm based on the HC index requires the least tuning time as well as the smallest number of switches. When $P_{on} = P_{swi} = 250\ mW$, $P_{off} = 50\ \mu W$, the HC index only consumes 19 percent power of STR R-tree, 39 percent power of HC R-tree, and 9 percent power of naive index under UNIFORM. Even when P_{swi} is 10 times higher than P_{on}, the HC index still consumes the least energy. For the other settings and data distributions (not shown here due to space limitations), the HC index also significantly outperforms the other indices.

Finally, it can be observed that, as setup time becomes longer and the ratio of P_{swi} to P_{on} becomes larger, the energy consumed by the switch operation starts to dominate the whole energy consumption. In this case, the broadcast server may consider adopting the naive approach to save clients' energy if THE key objective is energy conservation for the majority of the mobile clients.

6 Conclusion

With the advent of wireless networks and the popularity of mobile devices, the pervasive computing era will soon arrive. Wireless data broadcast, which allows simultaneous access by an arbitrary number of clients, is a very efficient and scalable information dissemination method. In this paper, we address the problem of answering CkNN queries in wireless data broadcast systems.

The characteristics of wireless data broadcast make direct adoption of existing spatial index structures a challenge [15], [25], [16]. In order to enable energy efficient retrieval of location-based data in the wireless data broadcast systems, the Hilbert curve index has been proposed [16], [27]. In this paper, we develop a new search algorithm to support CkNN search based on the HC index. Furthermore, we address two implementation issues by proposing: 1) a query window partitioning strategy to improve the spatial locality of Hilbert curve and 2) three index organization schemes to facilitate the processing of different queries. A theoretical model is developed to guide the choice of the most suitable organization scheme for broadcasting index information. Finally, a comprehensive simulation is implemented to evaluate the performance of the proposed CkNN search algorithm based on the HC index, compared against that of an algorithm based on revised R-tree variants. The result shows that the proposed CkNN search algorithm outperforms its R-tree counterparts significantly in terms of tuning time, for both the synthetic data set and the real data set. This study also takes into account the incurred overhead for switching between doze and active modes. Our experiments show that the energy savings by CkNN search algorithms (for both the HC index and R-tree index) outweigh the incurred switching overhead, with the HC index significantly outperforming R-tree. Generally speaking, the HC index has superior tuning time performance but also has to pay off the gain with a longer access latency. Since no index can optimize both tuning time performance and access latency performance, the HC index still is the best choice for applications with a priority on tuning time performance.

In this paper, we have assumed that all the data objects are logically stored in the broadcast server. As for the next step, we will consider the scenarios where the data objects are distributed in a network of broadcast servers. We are looking into this problem to develop cooperation strategies among the broadcast servers.

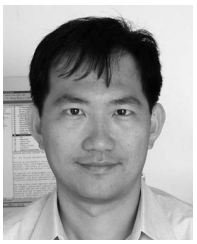
ACKNOWLEDGMENTS

Dik Lun Lee was supported in part by grants from the Research Grant Council, Hong Kong SAR, China (Grant No. 616005). Wang-Chien Lee was supported in part by US National Science Foundation Grants IIS-0328881, IIS-0534343, and CNS-0626709.

REFERENCES

[1] Q.L. Hu, D.L. Lee, and W.-C. Lee, "Optimal Channel Allocation for Data Dissemination in Mobile Computing Environments," *Proc. 18th Int'l Conf. Distributed Computing Systems,* pp. 480-487, 1998.
[2] Q.L. Hu, D.L. Lee, and W.-C. Lee, "Performance Evaluation of a Wireless Hierarchical Data Dissemination System," *Proc. Fifth Int'l Conf. Mobile Computing and Networking,* pp. 163-173, 1999.
[3] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments," *Proc. Int'l Conf. Management of Data,* pp. 199-210, 1995.
[4] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Data on Air—Organization and Access," *IEEE Trans. Knowledge and Data Eng.,* vol. 9, no. 3, pp. 353-372, May-June 1997.
[5] W.-C. Lee and D.L. Lee, "Using Signature Techniques for Information Filtering in Wireless and Mobile Environments," *J. Distributed and Parallel Databases,* vol. 4, no. 3, pp. 205-227, July 1996.
[6] Y. Tao, D. Papadias, and Q. Shen, "Continuous Nearest Neighbor Search," *Proc. 28th Int'l Conf. Very Large Data Bases,* pp. 79-96, 2002.
[7] Y. Tao and D. Papadias, "Time Parameterized Queries in Spatio-Temporal Databases," *Proc. Int'l Conf. Management of Data,* pp. 334-345, 2002.
[8] X. Xiong, F. Mokbel, and W.G. Aref, "SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases," *Proc. 21st Int'l Conf. Data Eng.,* pp. 643-654, 2005.
[9] A. Wang, S. Cho, G. Sodini, and P. Chandrakasan, "Energy Efficient Modulation and MAC for Asymmetric RF Microsensor Systems," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 106-111, 2001.
[10] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," *Proc. Int'l Conf. Management of Data,* pp. 47-54, 1984.
[11] Z. Song and N. Roussopoulos, "K-Nearest Neighbor Search for Moving Query Point," *Proc. Seventh Int'l Symp. Spatial and Temporal Databases,* pp. 79-96, 2001.
[12] G. Iwerks, H. Samet, and K. Smith, "Continuous K-Nearest Neighbor Queries for Continuously Moving Points with Updates," *Proc. 29th Int'l Conf. Very Large Data Bases,* pp. 512-523, 2003.
[13] F. Mokbel, X. Xiong, and W.G. Aref, "SINA: Scalable Incremental Processing of Continuous Queries in Spatio-Temporal Databases," *Proc. Int'l Conf. Management of Data,* pp. 623-634, 2004.
[14] X. Xiong, F. Mokbel, W.G. Aref, S. Hambrusch, and S. Prabhakar, "Scalable Spatio-Temporal Continuous Query Processing for Location-Aware Services," *Proc. 16th Int'l Conf. Scientific and Statistical Databases,* p. 317, 2004.
[15] J. Xu, B. Zheng, W.-C. Lee, and D.L. Lee, "Energy Efficient Index for Querying Location-Dependent Data in Mobile Broadcast Environments," *Proc. 19th Int'l Conf. Data Eng.,* pp. 239-250, 2003.
[16] B. Zheng, W.-C. Lee, and D.L. Lee, "Spatial Queries in Wireless Broadcast Systems," *ACM/Kluwer J. Wireless Networks,* vol. 10, no. 6, pp. 723-736, Dec. 2004.
[17] B. Zheng, W.-C. Lee, and D.L. Lee, "Search Continuous Nearest Neighbor on the Air," *Proc. First Int'l Conf. Mobile and Ubiquitous Computing: Networks and Services,* pp. 236-245, 2004.
[18] C. Gotsman and M. Lindenbaum, "On the Metric Properties of Discrete Space-Filling Curves," *IEEE Trans. Image Processing,* vol. 5, no. 5, pp. 794-797, May 1996.
[19] D. Moore, "Hilbert Curve," http://www.caam.rice.edu/dougm/twiddle/Hilbert, 2002.
[20] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 1996.
[21] D. Comer, "The Ubiquitous B-Tree," *ACM Computing Surveys,* vol. 11, no. 4, pp. 121-137, 1979.
[22] Spatial Datasets in 2D Space, http://www.rtreeportal.org/datasets/spatial/greece/cities_loc.zip, 2006.
[23] S.T. Leutenegger, J.M. Edgington, and M.A. Lopez, "STR: A Simple and Efficient Algorithm for R-Tree Packing," *Proc. 13th Int'l Conf. Data Eng.,* pp. 497-506, 1997.
[24] I. Kamel and C. Faloutsos, "On Packing R-Trees," *Proc. Second Int'l Conf. Information and Knowledge Management,* pp. 490-499, 1993.
[25] B. Zheng and D.L. Lee, "Information Dessimination via Wireless Broadcast," *Comm. ACM,* vol. 48, no. 5, pp. 105-110, May 2005.
[26] H. Sagan, *Space-Filling Curves.* Springer, 1994.
[27] B. Zheng, W.-C. Lee, and D.L. Lee, "Spatial Index on Air," *Proc. First Int'l Conf. Pervasive Computing and Comm.,* pp. 297-304, 2003.

Baihua Zheng received the PhD degree in computer science from the Hong Kong University of Science and Technology. She is a member of the IEEE and the ACM. Currently, she is an assistant professor in the School of Information Systems at Singapore Management University. Her research interests include mobile and pervasive computing and spatial databases.



Wang-Chien Lee received the BS degree from the Information Science Department, National Chiao Tung University, Taiwan, the MS degree from the Computer Science Department, Indiana University, and the PhD degree from the Computer and Information Science Department, the Ohio State University. He is an associate professor of computer science and engineering at Pennsylvania State University. Prior to joining Penn State, he was a principal member of the technical staff at Verizon/GTE Laboratories, Inc. Dr. Lee leads the Pervasive Data Access (PDA) Research Group at Penn State University which performs cross-area research in database systems, pervasive/mobile computing, and networking. He is particularly interested in developing data management techniques (including accessing, indexing, caching, aggregation, dissemination, and query processing) for supporting complex queries in a wide spectrum of networking and mobile environments such as peer-to-peer networks, mobile ad hoc networks, wireless sensor networks, and wireless broadcast systems. He has served as a guest editor for several journal special issues on mobile database-related topics, including *IEEE Transactions on Computers*, *IEEE Personal Communications Magazine*, *ACM MONET*, and *ACM WINET*. He was the founding program committee cochair for the International Conference on Mobile Data Management. He is a member of the IEEE and the ACM.

Dik Lun Lee received the MS and PhD degrees in computer science from the University of Toronto, Canada. He is a professor in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology and was an associate professor in the Department of Computer Science and Engineering at the Ohio State University. He was the founding conference chair for the International Conference on Mobile Data Management. His research interests include information retrieval, search engines, mobile computing, and pervasive computing. He served as the chairman of the ACM Hong Kong Chapter in 1997.