11-2004

Spatial Queries in Wireless Broadcast Systems

Baihua ZHENG

Singapore Management University, bhzheng@smu.edu.sg

Wang-Chien LEE

Hong Kong University of Science and Technology

Dik Lun LEE

Hong Kong University of Science and Technology

DOI: https://doi.org/10.1023/B:WINE.0000044031.03597.97

Spatial Queries in Wireless Broadcast Systems

Baihua Zheng Wang-Chien Lee[†] Dik Lun Lee[‡]

Singapore Management University, 469 Bukit Timah Rd, Singapore 259756
bhzheng@smu.edu.sg
[†] The Penn State University, University Park, PA 16802
wlee@cse.psu.edu
[‡] Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong
dlee@cs.ust.hk

Abstract

Owing to the advent of wireless networking and personal digital devices, information systems in the era of mobile computing are expected to be able to handle a tremendous amount of traffic and service requests from the users. *Wireless data broadcast*, thanks to its high scalability, is particularly suitable for meeting such a challenge. Indexing techniques have been developed for wireless data broadcast systems in order to conserve the scarce power resources in mobile clients. However, most of the previous studies do not take into account the impact of location information of users. In this paper, we address the issues of supporting spatial queries (including window queries and kNN queries) of location-dependent information via wireless data broadcast. A linear index structure based on the Hilbert curve and corresponding search algorithms are proposed to answer spatial queries on air. Experiments are conducted to evaluate the performance of the proposed indexing technique. Results show that the proposed index and its enhancement outperform existing algorithms significantly.

Keywords: location-dependent spatial queries, wireless broadcast, index structure, pervasive computing

1 Introduction

As envisaged by Mark Weiser a decade ago, pervasive computing has gradually become an essential part of our daily lives. Various research and engineering effort on communications, services, infrastructures, and devices has been put forth to bring this vision of pervasive computing towards a reality. While engineers and researchers around the world continue chip away various technical obstacles to this vision, there are still a lot of challenges to be overcome [29]. Among those, *scal-*

ability is one of the most important issues because of a dramatic increase in the number of users and applications involved.

Context-awareness has been seen as a critical element of pervasive computing, which refers to the special capability of an information infrastructure to recognize and react to real-world context. Context refers to information that describes conditions or situations where computation may adapt accordingly [1]. Location is an intuitive but important class of context information. Thus, location-awareness is naturally the first step to facilitate context-awareness. In pervasive computing environments, the *mobility* of users, applications, and data objects brings in another dimension of complexity to query processing in information systems. For example, an enquiry regarding to traffic condition should be answered based on the current location of the user (i.e., the query issuer). Because of the fact that a user may move, many existing techniques for processing spatial data and queries may no longer be efficient or workable since they are mostly based on the assumptions of fixed data and query points. Thus, scalability and mobility issues represent two primary challenges for the information systems in the pervasive computing era.

In this paper, we explore the scalability and mobility issues of supporting *location-dependent spatial queries (LDSQs)* in pervasive computing environments. Wireless broadcast, which has long been used for radio and TV signal transmission, is a natural solution to address the scalability issue. The smart personal objects technology (SPOT), recently announced by Microsoft at 2003 International Consumer Electronics Show (CES), has further exploited the feasibility of using wireless data broadcast in the pervasive computing era [8]. With a continuous broadcast network (called DirectBand Network) using FM radio subcarrier frequencies, SPOT-based devices such as watches and alarms, can continuously receive timely, location-aware, personalized information.

There are studies in the literature showing that wireless data broadcast is an efficient approach to disseminating information to an arbitrarily large number of mobile information consumers [2, 10, 11]. However, most of the previous studies only addressed the dissemination via wireless broadcast without taking into account the impact caused by location information. In this paper, as a first step towards this direction, we address the issues of supporting LDSQs in wireless broadcast systems. We assume that information objects with spatial properties (i.e., location) are disseminated on wireless data channels (similar to radio or TV channels). Information consumers (e.g., devices or applications) retrieve information objects from the wireless data channels to answer queries from their users. To simplify our discussions, in the context of this paper, we consider only the mobile devices as the information consumers and assume that the mobile devices know their current locations[1]. Thus, LDSQs can be processed locally at the devices, without submitting the location information to the server.

As we will detail later in the paper, the specific characteristics of LDSQs and broadcast systems

[1]With rapid technical advances and cost reduction of GPS and network triangulation technologies, this is a reasonable assumption.

introduce many new challenges that make it impossible to directly employ existing techniques. For example, the answer to a query may be sensitive to the location where the query is issued. Thus, it may not always be valid to the same query issued from a different location. Moreover, instead of being always available when stored in disks or memories, the index information is only available when it is on broadcast. In a previous work, we have exploited some aspects of location-dependent query processing. Grid-partition index focuses on nearest-neighbor search in wireless broadcast environments [36], while D-tree is aiming at supporting point queries in a non-overlapped spatial solution space [34]. In a recent paper [35], we propose a general indexing structure for provisioning the most common spatial queries, including window queries and k nearest neighbor queries. This represents the first effort on supporting spatial queries in a wireless broadcast environment. In this paper, we extend the previous work. Our contributions are four-fold:

- A new research direction of provisioning spatial information and supporting spatial queries in the wireless data broadcast systems is identified and studied. Particularly, the location-dependencies of query results to their users are exploited.
- A new index structure based on the Hilbert curve is proposed.
- A cost model is developed to measure the performance of the proposed index and to provide technical insights.
- A simulation is conducted to compare our proposal with state-of-the-art indexes, using both synthetical data and real data.

The rest of this paper is organized as follows. A brief review of the broadcast-based and traditional solutions for spatial queries is provided in Section 2. In Section 3, we analyze the requirements for supporting spatial queries in wireless data broadcast systems. Based on these requirements and our analysis, a new index structure based on the Hilbert curve, a space filling curve, is proposed. A simulation based performance evaluation is conducted in Section 4. Finally, we conclude this paper in Section 5.

2 Preliminary and Related Work

To the best of our knowledge, this is the first work on supporting general location-dependent spatial queries in wireless data broadcast environments. In the following, we briefly review the existing work related to location-dependent spatial queries and wireless data broadcast.

2.1 Location-Dependent Spatial Queries

Due to the mobility of users in pervasive computing environments, the answer to a spatial query may be dependent on the location of its consumer (i.e., the mobile device). Thus, we term this kind of queries as location-dependent spatial queries. In this paper, we concentrate on two common classes of spatial queries, namely, *window queries* and *k-nearest-neighbor (kNN)* search. We assume that the data objects are represented as points in the spatial space. Window queries find the objects that are located within a given window, which is a rectangle in a 2-dimensional space. kNN queries, as the term indicates, return k objects in the spatial space (which contains n objects where $n >= k$) closest to a given query point.

R-tree [14] and its variants, such as R$^+$-tree [32] and R*-tree [4], provide a good solution to window queries in traditional disk-based spatial databases. Based on certain heuristics, the objects close to each other are grouped into leaf nodes of the trees. Internal nodes use the minimal-bounding rectangles (MBRs) to represent areas that cover all the sub-areas represented by their children nodes. Therefore, a window query is processed by visiting the tree nodes that overlap with the query window. If the objects are available a priori, some packing algorithms can be employed to build R-tree to achieve a better search performance.

kNN search, a well-known problem in computational geometry, was first formulated by Minsky and Papert in 1969 [26]. In the 1990s, researchers in spatial databases became interested in this problem [28, 31] and gradually extended it into a high-dimensional space, such as image similarity comparison and content-based retrieval in multimedia applications [5, 15]. In addition, some studies have developed approximation algorithms for the problem [12]. Based on the the number of scans on a dataset, the kNN algorithms can be classified into two categories: single-step search and multi-step search.

Single-Step Search This category of algorithms searches answers based on associated index structures. They scan the dataset, D, only once. Several different approaches are described in the literature. Branch-and-bound algorithms use distance-based heuristics to determine the next node to visit and to prune branches that would never be visited, and the variants mainly differ in searching orders and the metrics used to prune the branches [7, 17, 28]. Incremental algorithms report the qualified spatial objects one by one to facilitate pipelined query processing, especially for complex queries involving proximity [16]. Some approaches directly use the Voronoi-Diagrams, which provide the solution space of kNN search for a fixed k [3].

Multi-Step Search There are situations in practice where a complex similarity distance between two spatial objects is very expensive to obtain. To address this issue, multi-step algorithms have been proposed to use cheaper filter distance functions to first obtain candidate sets (in the filter steps) for exact evaluation later in the refinement steps. Korn et al. proposed an adapted algorithm [23]. First, a set of k primary candidates was selected based on stored statistics to obtain the upper bound d_{max}, which can guarantee that there are at least k objects within the distance d_{max} from the query point q. Next, a range query was executed on the dataset to retrieve the final candidates. An extended version of this algorithm was proposed in [31] in which d_{max} was adapted every time a candidate object was checked.

Given a spatial query, such as a kNN query, a naive way of finding the answer set is to use brute force by checking each object and choosing those that satisfy the conditions to form the final answer set. However, this kind of naive solution incurs a large search space, resulting in a long search time. In order to answer spatial queries more efficiently, algorithms are usually designed according to two common objectives: 1) reducing the size of retrieved data set; and 2) optimizing the representation of data objects. The former is realized by intelligent filtering algorithms that can detect nearly all the not-qualified objects, and the latter is achieved by an efficient representation scheme to use less bits to represent the basic information of the objects. In general, various methods have been employed to achieve these objectives [12]. For example, heuristic distance information can help to prune the impossible nodes in order to reduce the number of visited nodes [28], and the hashing strategy is used in [33] to limit the search space.

2.2 Wireless Data Broadcast

Generally speaking, there are two basic approaches to delivering information to mobile clients: 1) on-demand access. A mobile client issues a request to the server. The server locates the appropriate data and returns it to the mobile client. 2) broadcast. Data are broadcast on one or more wireless channels open to the public. After a mobile client receives a query from its user, it tunes into the broadcast channel and receives data in accordance with specified query conditions. Hybrid approaches that combine the above two have also been configured [18].

Compared to on-demand access, broadcast has the advantage of scaling up to serve an arbitrary number of clients without incurring additional cost at the server site. Hence, it is a promising and desirable dissemination method for the future pervasive computing environment where the client base is expected to be huge.

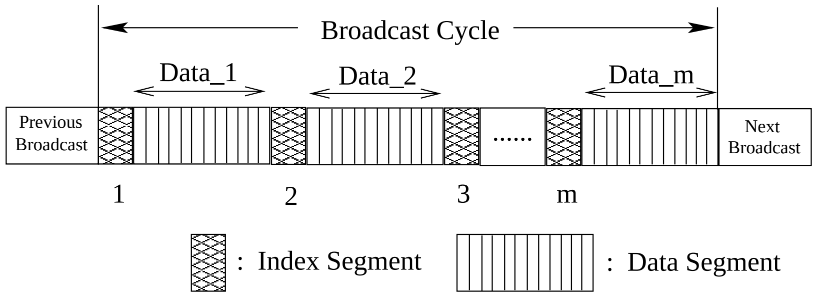


Figure 1: Data and Index Organization on the Broadcast Channel Using the (1, m) Interleaving Technique

To facilitate information services on wireless broadcast channels, index information is typically broadcast along with the data objects. Studies show that interleaving index information with data objects may assist mobile clients to filter out unwanted information during query processing and reduce consumption of mobile clients' battery power [20, 24]. By looking up the index, the mobile client is able to predict the arrival time of the desired data and only needs to tune into the broadcast channel when the requested data arrives. Broadcast organizations that properly interleave index and data objects on the broadcast channel can significantly improve energy efficiency at the minimal expense of access efficiency. Figure 1 illustrates a well-known broadcast organization called $(1, m)$ interleaving technique [20]. As shown, the whole index is broadcasted preceding every $\frac{1}{m}$ fraction of the broadcast cycle, the period of time when a complete set of data objects is disseminated. By replicating the index for m times, the waiting time for reaching the root node of a forthcoming index segment can be reduced. To further reduce the power consumption, each index node (except for the root) and each data object maintains a pointer to the root of the next index. The access protocol for receiving information off a broadcast channel involves the following steps:

Initial probe: The client tunes into the broadcast channel and determines when the next index will be broadcast. It then turns into the power saving mode until the next index arrives.

Index search: The client searches the index. It follows a sequence of index nodes (by selectively tuning into the broadcast channel) to locate the desired data objects and to determine when to tune into the broadcast channel to receive them. It waits for the arrival of the data in the power saving mode.

Data retrieval: The client tunes into the channel when the desired data arrives and downloads the data.

In wireless communications, a bit stream is normally delivered in the unit of *packet* (or *frame*), for the purposes such as error-detecting, error-correction, and synchronization [20]. For example, in the GPRS network a packet can contain the data of up to 1600 bytes [6]. As a result, data are accessed by clients also in the unit of packet, similar to the concept of *page* in traditional databases. In the following description, we use page, rather than packet (frame), for its generosity.

2.3 Observations

There are some factors making access to location-dependent spatial information in wireless broadcast systems different from that in traditional databases. These differences introduce new research challenges and motivate our work and they are summarized below.

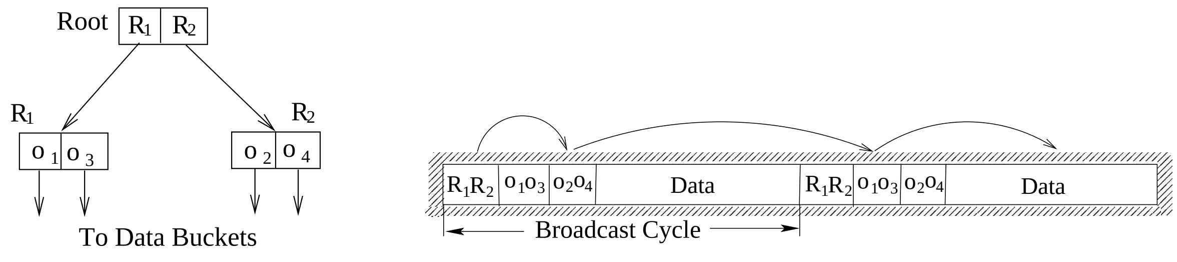
Resource Constraints Due to their portability, mobile devices have various inherited limitations, such as small user-interfaces, limited storage spaces, and scarce battery power. Among these, power supply is particularly important since it keeps the devices and applications running. Hence, algorithms designed to run on mobile devices should cater to all these limitations.

Mobility The unlimited mobility of clients makes the access to location-dependent information a new and challenging topic, especially when a client issues a LDSQ associated with her position. Existing query processing strategies in traditional databases have not taken into consideration this mobility and changing location issues. Therefore, new information access and dissemination schemes need to be devised.

On Air Index An important characteristic of wireless data broadcast is that the index information is on air. Index information is available to the clients only when it is currently being broadcast. Hence, when an algorithm traverses the index packets in an order different from the broadcast sequence, it has to wait for the next time the packet is broadcast. In contrast, the index for traditional databases is stored in resident storages, such as memories and disks. Consequently, it is available anytime. Since nearly all the existing index structures and algorithms devised for traditional databases do not consider the time-series characteristics of the air index, they cannot be easily deployed in wireless broadcast environments. An example of the well-known R-tree index is given in Figure 2. Assuming that the query processing algorithm first visits the node R_2 and then R_1, and that the server first broadcasts node R_1 then node R_2, the access latency is significantly extended since the node R_1 is not available until the next cycle.

As a conclusion, a good index structure and associated algorithms serving spatial queries in a wireless broadcast environment should incur small space cost, take linear broadcast order into account, and perform the search efficiently (in terms of power and access).

Regarding the performance metrics, *tuning time* and *access latency* are employed in this study. The former is the time spent by a client listening to the broadcast channels, logically representing the client's power consumption. It includes the time to search in the index and the time to download the desired data. Since the time used to download data is fixed for a given environment, we only measure the index search time for this metric. The latter is the time duration from the point that the client requests some data to the point that the desired data is received. Both the tuning time and the access latency are measured in terms of number of page accesses [19, 20].



(a) R-tree Index (b) Branch-and-Bound Search

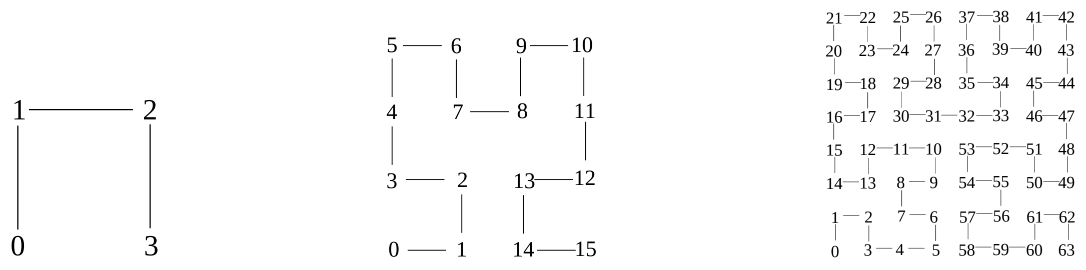
Figure 2: Linear Access on Wireless Broadcast Channel

3 Hilbert-Curve Index Structure

In wireless data broadcast, data packets are sequentially delivered on the broadcast channel. Thus, organizing data in a way such that the users can efficiently retrieve data is critical. In response to the *linear* property of wireless broadcast channels, we propose a new index structure, based on a space-filling curve, to facilitate the processing of LDSQs by linear scanning of the dataset (rather than random traversal of the index nodes in accordance with some heuristics). 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, and the Peano curve, are different in the order in which the points in the grid space are visited. The *Hilbert curve*, due to its *optimal locality* [13], is chosen in this paper to build an index for LDSQs. In this section, we first explain the basic idea of using the Hilbert curve for wireless data broadcast. We then go into the details of the proposed algorithms for answering LDSQs.

Like many other space-filling curves, the Hilbert curve maps points from a multi-dimensional space to a one-dimensional space. *Locality* is an important metric for choosing space-filling curves. A mapping from n dimensions to m dimensions (where $m < n$) is considered to have good locality if points that are close to each other in original n-dimensional space are also close to each other after being mapped onto an m-dimensional space. Considering the nearest neighbors along the grid axes only, each grid point has $2n$ nearest neighbors in the original space. However, it will only have $2m$ nearest neighbors after being mapped onto an m-dimensional space. Therefore, for a mapping from n dimensions to one dimension, such as the Hilbert curve, the best that we can hope for is to have two of the $2n$ nearest neighbors remain as nearest neighbors in the one-dimensional space. This subjective criteria is met by the Hilbert curve, which guarantees a good locality feature.

Figure 3(a) 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 Figure 3(b) and 3(c), respectively. The number, called *index value* in this paper, represents the order of visits at different points in the Hilbert curve. For example, point $(1,1)$ has the index value 2 in curve H_2.



(a) H_1 (b) H_2 (c) H_3

Figure 3: Hilbert Curves of Order 1, 2 and 3

To address the issue of denoting the nodes, the Hilbert curve needs to allocate sufficient number of bits to represent the index values in order to guarantee that each of the point in the original space has a distinct value. Let c_i be the number of bits used for a coordinate in the ith dimension of the targeted m-dimensional space. A total of $\sum_{i=1}^{m} c_i$ bits need to be allocated to represent an index value in the m-dimensional space. As such, each point in the original space can be properly represented in the new space.

Given the mapping function of the Hilbert curve, it is easy for a client to perform a conversion between coordinates and Hilbert-Curve index values. 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 pre-set system constant, the conversion can be done in a constant time. Readers who are interested in the details of a conversion algorithm may refer to [27].

Even though the Hilbert curve has been previously employed in spatial indexes [22, 21], what we propose in this paper is fundamentally different from existing ones. First, while geometrical information, such as MBRs of spatial objects, is maintained in those previously proposed indexes, we only consider the pure linear index value. Second, the search algorithms in the previous studies are typically based on geometrical distribution and certain heuristics derived from the index (e.g., R-tree). As we pointed out earlier, this strategy may not be applicable in a wireless broadcast environment. In this paper, we use index values to guide searches and only use geometric information for filtering in the final step.

The Hilbert curve facilitates the linear scan of objects in a multi-dimensional space. Thus, the problem is reduced to a question of how to answer LDSQs based on data broadcast in the order of the Hilbert curve. In the next two sub-sections, we answer this question by proposing two algorithms for window queries and kNN queries, two of the most important classes of LDSQs.

3.1 Window Queries

To process a window query based on Hilbert-Curve index, the basic idea is to decide a candidate set of points along the Hilbert curve which includes all the points fallen within the query window. These points are retrieved to filter out those fallen outside the window. There is an existing algorithm that can determine the first or last point lying on the boundary of a box visited by the Hilbert curve, with a time complexity of $O(n^d)$ [27]. Here, n is the number of bits used to represent a coordinate in one dimension and d is the dimensionality of the search space. As in Figure 6(a), regarding the dashed-line rectangle as the bounding box, the first point is point a and the last point is b, sorted according to their occurring orders on the Hilbert curve. If this dashed-line rectangle is the query window, all the points inside this query window should lie on the Hilbert curve segmented by points a and b. In other words, the values of points a and b bound the Hilbert curve values of all the candidate points.

In the following, we prove that the set of points bounded by the largest and the smallest Hilbert curve index values on the query window contains all the points satisfying the query.

Claim 1: For a given window, the point p inside the query window that has the largest Hilbert-Curve index value must be lying on the bounding box of the query window.

Proof: Assume that there is a point p' inside the query window which has a larger index value

Algorithm 1 Window Queries

```
Input: a query window $w$, sorted objects' indexes;
Output: objects within query window;
Procedure:
1: $w$ = overlap ($w$, search space); $result = \varnothing$;
2: compute_bounding_points($w$, $first$, $last$); //$first$ and $last$ mean the smallest and largest index value;
3: for any object whose index value $v$ within the range [$first$, $last$] do
4:     $p$=index_to_coor($v$);
5:     if within_window ($w$, $p$) == 1 then
6:         $result = result \cup \{p\}$;
7:     end if
8: end for
9: return $result$;
```

than p. Since the Hilbert curve is a continuous path to visit every point in the search space, there must be a point p'' outside of the query window, having a larger index value than p' [2]. Considering a line connecting p' and p'', it must intersect the bounding box on point q. Since the index values of the points on the Hilbert curve are monotonously increasing, the index value of q which is between p' and p'' must be larger than that of p'. Consequently, the index value of q is larger than p, which has the biggest value according to our statement. Hence, the previous assumption fails and the point p having the largest value should be on the bounding box. Therefore, our claim is proven.

□

Similarly, the point within the query window that has the smallest index value is guaranteed to be on the boundary. The detailed algorithm for processing window query is presented in Algorithm 1. In order to simplify our discussion, a two-dimensional space is assumed and the derived conclusion can be easily extended to a high-dimensional space.

In summary, for a given window query, the intersection of the query window and the whole search space serves as the real query window. Taking the query window bounded by the dashed line in Figure 6(a) as an example, the steps employed to answer a window query are illustrated as follows. First, the first and last points on the bounding box are detected whose values bound the range of all the candidate objects along the Hilbert curve, i.e., point a and point b. Second, all the objects having value between 9 and 54 are detected. Finally, a filtering mechanism is employed to find the objects belonging to the real answer set, via checking the coordinates of the object with the query window.

[2]Otherwise, p' is the last point of the Hilbert curve and must be on the boundary of the original search space, therefore it cannot be inside the given window.

3.2 kNN Queries

A general strategy employed by any kNN algorithm is to determine a search space that is not too small to lose any correct answer and not too big to perform any unnecessary search. It is ideal to know the exact distance between the query point q and its k-th nearest neighbor o_k ($dis(q, o_k)$), because the circle having q as the center and $dis(q, o_k)$ as the radius bounds the necessary and sufficient search space that contains all the answers. However, it is difficult to obtain the exact distance $dis(q, o_k)$ and an alternative is to estimate the distance. As described in Section 2, some optimistic algorithms use tight prediction to avoid any unnecessary search and some pessimistic ones use a loose estimation to avoid any loss of the requested objects. In our algorithm, a range estimation algorithm is proposed based on the locality property of the Hilbert curve. The pseudocode is given in Algorithm 2.

Algorithm 2 kNN Queries

```
Input: a query point $q$, sorted objects' indexes;
Output: k nearest neighbors;
Procedure:
1: $index_q = coor\_to\_index(q)$;
2: locate the $i$th object $o_i$ who has the nearest index value ($index_i$) to $index_q$;
3: $begin$ = MAX(0, $i$-$k$/2);
4: for j = $begin$, $rad$ = 0; j =< ($begin + k$); j++ do
5:   $p$=$index\_to\_coor(index_j)$; $rad$ = max($rad$, distance($p$, $q$));
6: end for
7: let $squ$ be the bounding square centered at $q$ and having $2rad$ as side length;
8: $result\_set$ = window_query($squ$); $answer\_set = \varnothing$; $max\_dis = 0$;
9: for each object $o_p$ in the $result\_set$ do
10:   $dis$ = distance($o_p$, $q$)
11:   if ($answer\_set$ is not full) then
12:     $answer\_set = answer\_set \cup \{o_p\}$; $max\_dis$ = max($max\_dis$, $dis$);
13:   else
14:     if $dis < max\_dis$ then
15:       replace the farthest object in the $answer\_set$ with $o_p$ and update $max\_dis$ correspondingly;
16:     end if
17:   end if
18: end for
19: return $answer\_set$;
```

As the first step, the k nearest objects to the query point along the Hilbert curve are found and a minimal circle centered at query point and containing all those k objects is constructed. The MBR of that circle, having at least k objects and definitely causing no loss, serves as the search range. As the example depicted in Figure 4, the query point q wants to find its 4 nearest neighbors. Locating q at the Hilbert curve, the 4 nearest objects along the Hilbert curve can be detected, which are objects having values of 5, 6, 8, and 9 respectively. According to those four objects, a bounding circle denoted by dashed-line can de determined that is guaranteed to contain at least those 4 objects inside. Based on that circle, a window query, denoted by the dotted-line square, is issued to find all the candidate objects, whose number is at least 4. Finally, the real 4 nearest objects can be found by comparing the distance.

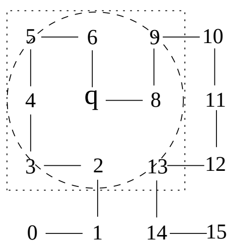


Figure 4: An Example for kNN Search

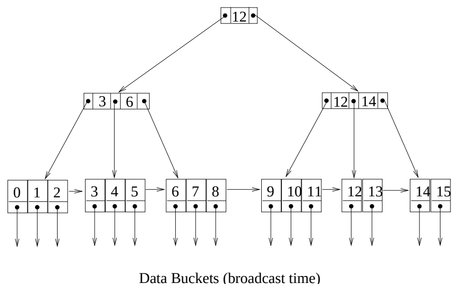


Figure 5: B$^+$-tree of Hilbert Curve Values

One question left is whether the range returned by searching the Hilbert curve is too loose, since a loose range results in the unnecessary search and a poor performance. Thanks to the fact that the Hilbert curve is close to optimal locality [13], the kNN objects should lie near the query point along the Hilbert curve. Consequently, we assume that the bounding rectangle only introduces limited extra search. Later simulation results provide evidences for this assumption.

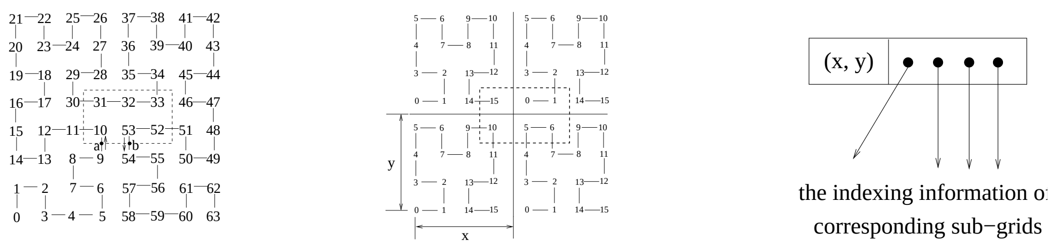
The other problem of this search algorithm is that the indexing information has to be replicated in the broadcast program to enable twice-scanning. For the first scanning, the values of objects along the Hilbert curve are broadcast to enable the query point to decide the search range according to its location and requested k. In the second part, the similar information is broadcast again to allow the clients to retrieve answers based on the search range. In the later experiment section, we can find that duplication results in a larger index storage cost, which has a direct impact on clients' access latency.

3.3 Search Improvement

The locality of the Hilbert curve is a major factor to decide the performance of our algorithms. If the nearby points in the original search space have big differences among their index values, the window query will have to check much more points than necessary. A motivating example is depicted in Figure 6(a) in which the dashed rectangle represents a query window. Employing our original algorithm, all the points whose index values between 9 and 54 should be checked. Obviously, this range actually contains many points outside the window.

It can be observed from the Hilbert curve that the order i curve is derived from order $(i-1)$ curve. If a query window crosses several order $(i-1)$ curves, it has a higher 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 whole space into several disjoint grids and each grid has its own Hilbert curve. For a 4-grid partition, each grid only needs an order $(i-1)$ curve, rather than a part of order i curve for the whole search space. Figure 6(b) shows that, after partitioning, the candidate set contains much fewer objects. We only need to check the points whose index values are between 0 and 2 for the grid in quadrant 1, and points whose values are between 14 and 15 for the grid in quadrant 2, and so on.



(a) Before Partition

(b) After Partition

(c) Extra Index after Partition

Figure 6: Improvement Introduced by Search Space Partition

Besides this advantage, the partition can also reduce the representation size. If the original space needs n bits to represent the index value of an object, it only requires $(n-2)$ bits for 4-grid partition in a two-dimensional space. In order to make a full use of this kind of reduction, we partition the space into the number that is powered by 2. The improvement over that based on original un-partitioned Hilbert curve is clear and its advantage will be further shown in later simulation results.

After partitioning, the original space is divided into several disjoint sub-grids and the detailed partition information is attached to the upper-level index. In other words, the server first broadcasts the span of the sub-grid of the original space. In our example shown in Figure 6(b), the values of x and y are broadcast. As shown in Figure 6(c), there are also 4 pointers following the pair of (x, y), which point to the corresponding sub-grids. Assuming the covered area of the original space is ((0, 0), ($span_x$, $span_y$)), and let $num_x = \lceil span_x/x \rceil$ and $num_y = \lceil span_y/y \rceil$, the ith sub-grid can be decided by its lower left vertex v_{ll} and upper right vertex v_{ur}, with v_{ll} being $(x \times (i\%num_x)$, $y \times \lceil i/num_y \rceil)$ and v_{ur} being $(x \times (i\%num_x + 1)$, $y \times (\lceil i/num_y \rceil + 1))$.

For a window query, the original query window should be partitioned into several disjoint sub-rectangles. For each grid, the overlapping between the query window and the grid produces the sub-rectangle. The revised algorithm for window query is given in Algorithm 3.

We can observe that the larger the number of partitions, the fewer the number of objects checked, since the partition itself preserves the objects' locality. However, as mentioned before, objects in

Algorithm 3 Revised Window Queries

```
Input: a query window $w$, sorted objects' indexes;
Output: objects within the query window;
Procedure:
1: $result = \varnothing$; //initialization;
2: for each grid having bounding box $b$ do
3:     $w' = \text{overlap}(w, b)$;
4:     if $w' \neq \varnothing$ then
5:         $result = result \cup \text{window\_query}(w', \text{sorted indexes of objects within } b)$;
6:     end if
7: end for
8: return $result$;
```

broadcast environments are accessed in the unit of pages, rather than by objects themselves. Hence, a smaller number of objects searched does not necessary result in a smaller number of pages accessed, which depends on the paging strategy employed to organize index information. The other fact is that the client has to perform multiple window searches, while it only processes one window query in the original space without any partition.

3.4 Cost Model

Tuning time and access latency are the two basic measures for an index. Considering the access latency, it can be theoretically obtained by the well-known (1, m) organization algorithm presented in [20]. It only depends on the index size which can be easily obtained when an index is built. This sub-section derives the tuning time performance for various kinds of spatial queries based on the Hilbert-Curve index we propose. In order to facilitate the description, some notations are defined in Table 1.

Notation	Definition
n	the number of bits assigned to a coordinate in one dimension
$p_{x,y}$	the query issued at the position (x, y)
hi_{first}	the first value of Hilbert-Curve index to search
hi_{last}	the last value of Hilbert-Curve index to search
v_i	the ith index value of the sorted Hilbert-Curve index
$loc(x, y)$	the position of point q (x, y) in the sorted Hilbert-Curve index
$cost(v_1, v_2)$	the number of nodes visited in a B$^+$-tree to retrieve data ranging from v_1 to v_2
t_r	the time used to obtain the approximated search range
t_o	the time used to retrieve the objects whose Hilbert curve values are within a given range

Table 1: Definition of Notations

In our implementation, a B^+-tree is used to store the index value. The leaf level contains the sorted Hilbert-Curve index of all the objects, and each object is represented by its index value and a pointer pointing to the next broadcast time of the page containing the real information of that object. As shown in leaf level of Figure 5, the object having value of 0 is denoted by 0 and a pointer pointing to the real data bucket containing information of that object. According to the traditional bottom-up construction method, B^+-tree can be easily built based on the given leaf nodes. The fan-out of the node is decided by the page capacity. Assume that the leaf nodes containing all the objects in Figure 4 and the fan-out of a node is 3, the corresponding B^+-tree is shown in Figure 5.

In a two-dimensional space, $p_{x,y}$ denotes the query point of a kNN query or the lowerleft vertex of a query window. Since n is dependent on the span of the query space, 2^n limits the possible largest coordinate along one dimension. Considering the index search time, it includes the time to obtain the approximated search range of index values and the time to retrieve the objects whose index values are within that range:

$$T = T_r + T_o = \sum_{y=1}^{2^n} \sum_{x=1}^{2^n} p_{x,y}(t_r + t_o) \tag{1}$$

$$t_r = \begin{cases} 0 & \text{a window query} \\ cost(v_{i-k/2}, v_{i+k/2}) & \text{a } k\text{NN query issued at } p_{x,y} \ \& \ \imath = loc(x,y) \end{cases} \tag{2}$$

$$t_o = cost(hi_{first}, hi_{last}) \tag{3}$$

For a window query, the client itself can decide the search range for a given query window and no t_r is required to obtain the range. For a kNN query, the client first checks the sorted Hilbert-Curve index according to the value of the query point to decide the necessary search range. Secondly, it navigates the dataset again to obtain the final results. In Equation (3), hi_{first} and hi_{last} bound the index values of the requested search range. In the cost model, the position that a query issued is assumed to be uniformly distributed and $p_{x,y}$ can be approximated by 2^{-2n} in a 2-dimensional space.

In addition to predicting the performance for a given dataset, the cost model can also be employed to provide some guidance for choosing a suitable partition of the search space. As we explained in Section 3.3, partitioning the search space actually is a tradeoff between the reduced tuning time and increased computation cost at clients. For a window query, it will be turned to multiple sub-window queries and the client has to compute the search range for each sub-window query. Whether the reduced tuning time deserves the enhanced computation burden is application-oriented, since different applications have their own concerns and various devices have diverse limitations. Therefore, a cost function can be employed to choose a suitable partition, which is defined in Equation 4.

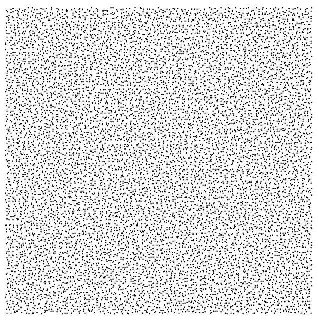
$$cost_i = \alpha \frac{t_i}{t_0} + \frac{c_i}{c_0} \tag{4}$$

Notations t_i and c_i represent the tuning time and computation burden of a specific partition P_i respectively, and P_0 means no partition. Parameter α is to assign different weights to those two factors, according to concerns of real applications. The partition has the smallest cost serves as the best choice for a given application.

A greedy algorithm can be used to obtain a good partition for a given search space[3]. Let ρ_i equal $\frac{c_i}{c_0}$, ρ_i will definitely increase when the number of sub-grids, i, becomes larger. It can be expected that once i reaches some value, ρ_i will be larger than the currently smallest value, $cost_j$, for some j, $1 <= j < i$. Since $cost_k$ is larger than ρ_i for any k bigger than i, the greedy hunting can be stopped and partitioning the space into $j \times j$ is the best choice according to our implementation. Hence, the algorithm is guaranteed to return the optimal decision after a limited number of searching.

4 Performance Evaluation

This section evaluates the performance of the proposed Hilbert-Curve index by comparing it to that of traditional indexes. Two datasets are used in the evaluation, as shown in Figure 7. In the first dataset (UNIFORM), $10,000$ points are uniformly generated in a square Euclidean space. The second dataset (REAL) contains 5848 cities and villages of Greece, which is extracted from the point dataset available from [9]. A discrete-event based simulation package, *CSIM* [30], is used to implement the model.


(a) UNIFORM

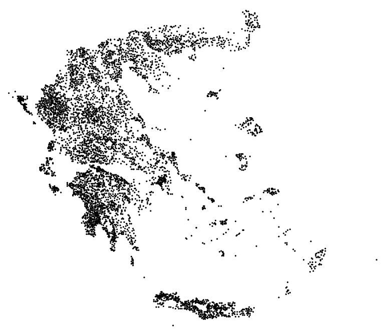

(b) REAL

Figure 7: Datasets for Performance Evaluation

For the existing index, we evaluate the search algorithm based on R-tree. Since the objects are available a priori, the STR packing scheme is employed to build R-tree (denoted as *STR* R-tree in the later presentation) in order to provide a fair performance comparison [25]. To achieve a better search performance for kNN queries, the original R-tree based search algorithms first sort the MBRs according to the query location [17, 28]. As explained in Section 2, such an algorithm

[3]This greedy algorithm is not the only choice.

is not suitable for air indexing, since it causes an extremely large access latency. We make the following revision. No matter where the query location is, the MBRs are accessed sequentially, while impossible branches are pruned according to the *mindist* and *minmaxdist* based heuristics as in the original algorithms (see [28] for details). R-tree is broadcast in a depth-first order to facilitate rollback operations. A packing algorithm using Hilbert curve for R-tree is available [21]. It is also implemented in our simulation in order to evaluate the impact introduced by the Hilbert curve on R-tree which is denoted as *Hilbert-Curve* R-tree.

The system model in the simulation consists of a base station, lots of clients, and a public channel for broadcast. *WinSideRatio* defines the ratio of the query window's side length to the side length of the whole search space. The available bandwidth is denoted as *BroadcastBand*, and the fixed size of a page is denoted as *Capacity*. The size of a data instance is *DataSize*. Two floating-point numbers are used to represent a two-dimensional coordinate, the same amount of bits are for an index value on the Hilbert curve. The size of a floating-point number is *FloatSize*. Table 2 summarizes the configuration parameters of the system model and also the default settings.

Parameter	**Description**	**Setting**
WinSideRatio	ratio of average side length of a query window to that of the search space	0.1
BroadcastBand	bandwidth of the broadcast channel	1000 Kbps
Capacity	the fixed size of the access unit	2^6 - 2^{12}
DataSize	size of a data instance	1024 bytes
FloatSize	size of a floating-point number	4 bytes

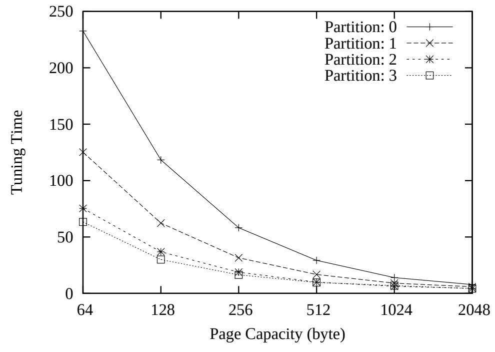
Table 2: Configuration Parameters of the Execution Model

While scalability is a major strength of wireless broadcast systems, we only simulate the action of a client since the number of clients does not affect the system performance. $1,000,000$ queries are issued randomly in our simulation. The figures depict the average performance. The results are presented in the following sub-sections in details.

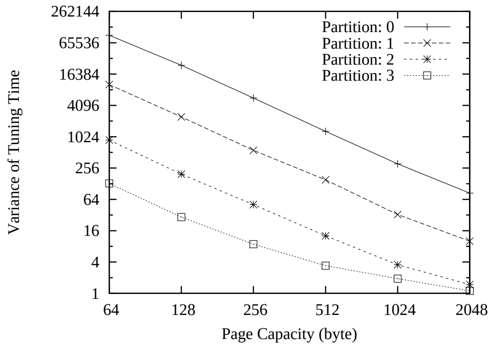
4.1 Improvement of Partitioning the Search Space

As mentioned above, the search performance is dependent a lot on the locality of the Hilbert curve. From the observation, partitioning the search space into smaller grids can reduce the probability that two nearby points have a large difference between their index values. The performance improvement obtained by partitioning the UNIFORM dataset is depicted in Figures 8(a) and 8(b). As explained before, one advantage of partition is to reduce the representation size. Consequently, we partition the space into p_i sub-partitions along ith dimension, given p_i equals to 2^{q_i}. Hence, the representation size for the index value of the Hilbert curve can be decreased by the summation of q_i; that is, $\sum_{i=1}^{m} q_i$, for an m-dimensional space. The number following the word **Partition** is q_i. In current implementation, all the dimensions are partitioned into the same number of sub-partitions, therefore q_i is same for all different i. For instance, *Partition:1* means that the original two-dimensional space is divided into $2^1 \times 2^1$ grids. Without explicit specification, Partition 2 services as the default setting in later simulations.

From the result, we can observe that the performance (in terms of tuning time) is increased a lot due to the partition. However, the computational overhead on clients are increased since it has to determine the boundary index values for multiple sub-query windows, rather than for a single query window in the original situation. Setting the tuning time of Partition 0 as the baseline for performance comparison, partition 1, 2, and 3 outperforms partition 0 on UNIFORM dataset by 66%, 86%, and 93%, respectively. The result of the REAL dataset is similar, which is omitted due to space limitation. The other observation is that the improvement is more significant for smaller page capacity. For large pages, although the number of objects visited is large without any partition, the number of pages accessed is not necessarily large since the data access is in the unit of page.



(a) Tuning Time



(b) Variance of Tuning Time

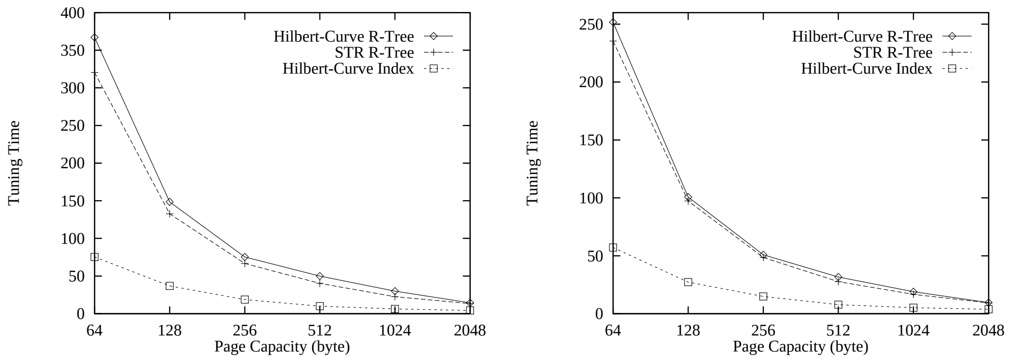
Figure 8: Tuning Time and its Variance of UNIFORM Dataset based on Hilbert-Curve Index

The performance improvement obtained by employing partitioning does not only occur in terms of the average tuning time but also in terms of the variance of tuning time. It means that partitioning improves the stability of system performance, which is desirable since the user may lose patience if the response time of a query is beyond the average that he expects from his experience of using the system.

4.2 Window Queries

As described before, window query is a common and important query type in spatial databases, which is frequently used to evaluate the performance of spatial indexing structures. In this subsection, the performance of the newly proposed Hilbert-Curve index, compared with various traditional methods, is presented. By fixing the size of query windows and varying page capacity from 64 to 2,048 bytes, Figure 9 shows the comparisons. Figure 10 shows the performance under fixed *Capacity* and varying *WinSideRatio*.

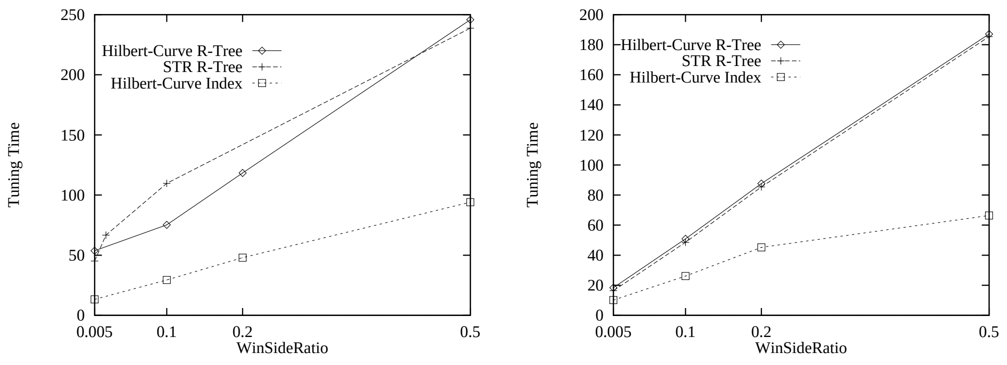
The most obvious observation obtained is that Hilbert-Curve index with Partition 2 has the best performance, and it outperforms the other two methods significantly. For UNIFORM dataset, the improvement in terms of tuning time is about 69.5% and 67.2% over STR R-tree and Hilbert-Curve tree, respectively. For REAL dataset, the advantage of Hilbert-Curve index is also dramatic. Compared to STR R-tree and Hilbert-Curve tree, the improvement is 53.4% and 55.6% in terms of the tuning time. As mentioned previously, partitioning the space does improve the search performance, while it requires multiple sub-window queries which will increase the computation complicity at clients site. However, this kind of overhead is very limited. Take the window query with *WinSideRatio* set to 0.1 as an example, the average number of sub-window queries issued by a client is 1.78 for the UNIFORM dataset, and it is 1.75 for the REAL dataset. The reason is that only when a query window passes the partition lines, the window needs to be partitioned into sub-window. Especially the window size is usually very small compared with the size of the search space.



(a) UNIFORM Dataset (b) REAL Dataset

Figure 9: Tuning Time of Datasets for Fixed-Size Window Queries

In order to provide a comprehensive evaluation, the simulation results obtained based on varied query window size are also included. The value of parameter *WinSideRatio* is changed from 0.05, to 0.1, to 0.2, and finally to 0.5. As expected, the Hilbert-Curve index structure with Partition 2 has the best performance. The performance gain becomes more obvious as the size of the query window increases. Here, the page capacity is set to 256 bytes. The difference of the window size affects the performance of R-tree based indexes more distinctively. The tuning time changes from 16 to 185 for STR R-tree, and from 18 to 187 for Hilbert-Curve R-tree. For the Hilbert-Curve index of Partition 2, it changes from 10 to 66 for Partition 2. All the comparison information above is based on the REAL dataset. The result on the UNIFORM dataset is similar. Therefore, we may conclude that the size of the query window has less impact on the performance of the Hilbert-Curve index than that of the others and that the Hilbert-Curve index provides a more stable performance for window queries.



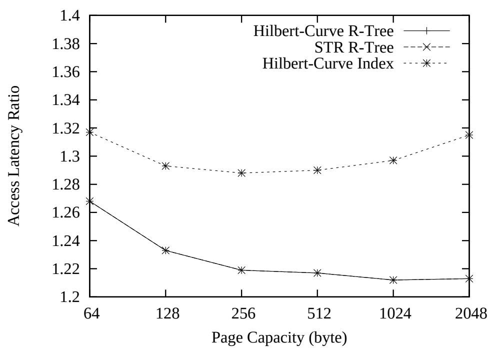
(a) UNIFORM Dataset (b) REAL Dataset

Figure 10: Tuning Time of Various-Size Window Queries

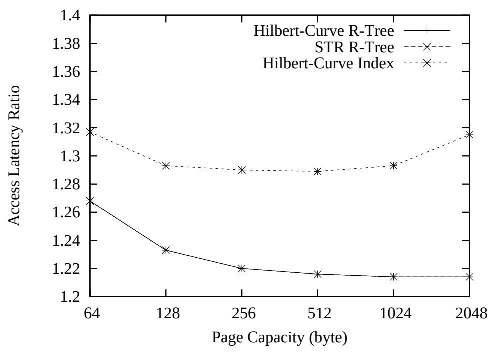
4.3 *k*NN Queries

*k*NN query is one of the most representative spatial queries. It returns k objects that are nearest to the query point. When k is set to 1, it is the famous nearest neighbor search. In this section, the performance of different indexing structures for *k*NN search is compared. First, we evaluate their performance for traditional NN problem, then for fixed k, and finally for various settings of k.

The storage cost is not a big issue in the traditional disk-index environment, with the enlargement of the disk capacity and reduction of its price. However, in broadcast systems, all the index information has to occupy some bandwidth for transformation and therefore affects the access latency of the clients. Consequently, the index size is preferred to be small. Since our algorithm devised to solve *k*NN queries should scan the index value twice, with the first time to decide the necessary search boundary and the second time to obtain the candidate objects, its index size will be larger than that of R-tree. Hence, the expected access time of clients is checked first to guarantee that the Hilbert-Curve index does not introduce too long latency and its performance is depicted in Figure 11. The famous (1, m) index organization algorithm is employed here to achieve the optimal access latency [20]. Assuming the access latency of the scheme having no index information as 1, R-tree introduces the access latency about 1.23 and Hilbert-Curve index incurs the latency about 1.30 for the UNIFORM dataset. For the REAL dataset, the result is nearly the same. Hence, we can make the conclusion that the index overhead caused by Hilbert-Curve index is acceptable and similar as it of other existing ones.



(a) UNIFORM Dataset



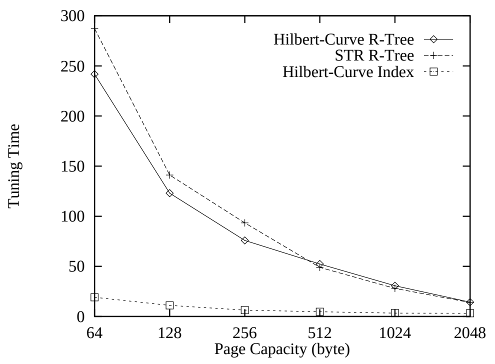
(b) REAL Dataset

Figure 11: Access Latency of Different Indexing Structures for kNN Queries

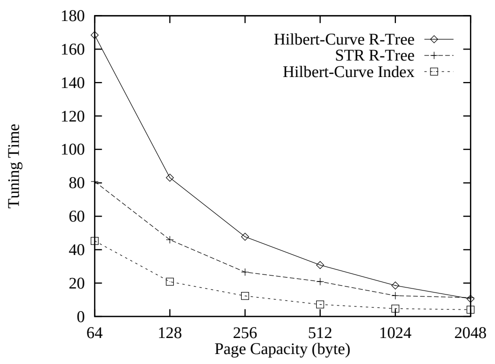
4.3.1 Nearest-Neighbor Queries (k=1)

Nearest-neighbor search is a special case of kNN queries. It has been frequently used as a test case to evaluate the performance of index structures proposed for kNN problems. Figure 12 depicts its performance for both UNIFORM and REAL datasets.

Obviously, Hilbert-Curve index can provide a better performance when partitioned into several sub-grids. It outperforms STR R-tree and Hilbert-Curve R-tree for about 66.4% and 64.5%, respectively, on UNIFORM dataset. While the outperformance for REAL dataset is not so significant, only 18.7% and 46.0% respectively. Figure 13 depicts a comparison of the variance of the tuning time. Hilbert-Curve index with partitioning outperforms others significantly.



(a) UNIFORM Dataset

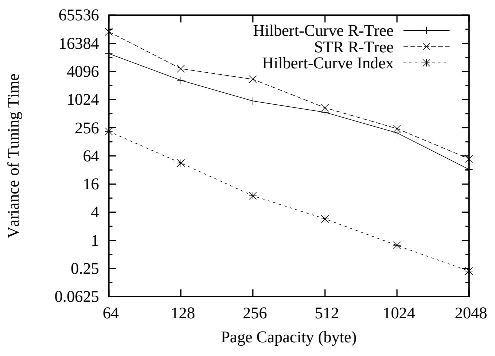


(b) REAL Dataset

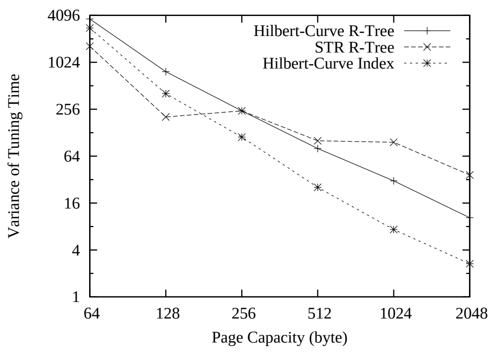
Figure 12: Tuning Time of Nearest Neighbor Queries

4.3.2 k Nearest Neighbor Queries

For the general kNN search, Figure 14 shows the performance of various methods by setting k to 10 for both datasets. Figure 15 depicts the simulation results of kNN queries with various values



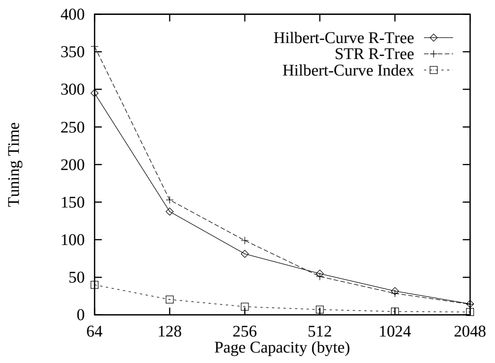
(a) UNIFORM Dataset



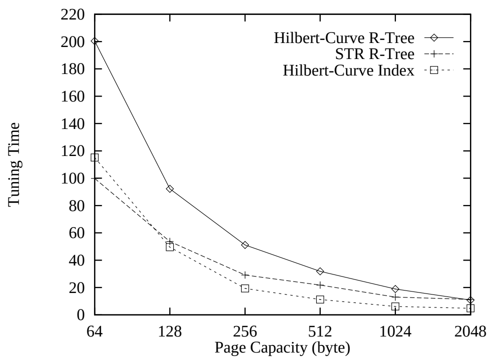
(b) REAL Dataset

Figure 13: Variance of Tuning Time of Nearest Neighbor Queries

of k and page capacity set to 256 bytes for both datasets.



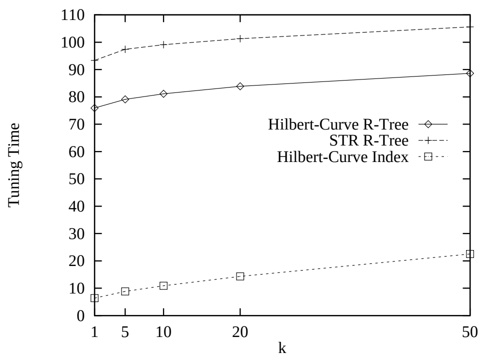
(a) UNIFORM Dataset



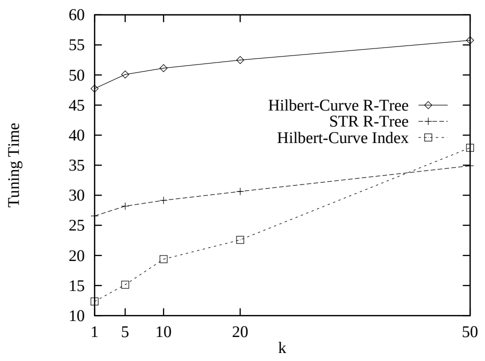
(b) REAL Dataset

Figure 14: Tuning Time of kNN (k=10) Queries

Considering the kNN query, with a fixed value of k or varying values of k, the Hilbert-Curve index with partition performs the best in nearly all the cases. The first observation is that Hilbert-Curve index with partition always achieves the best performance for UNIFORM dataset, while it somehow performs a little bit worse than STR R-tree in the REAL dataset, especially for the small page capacity or large value of the request k. The second observation is that for the comparison between STR R-tree and Hilbert-Curve R-tree, the former works better for the REAL dataset and the latter makes some gain in the performance of the UNIFORM dataset. The reason causing these two behaviors is that the Hilbert curve has a better location locality for the dataset that uniformly distributed, compared to the skew distributed dataset. From the previous simulation result, Hilbert-Curve index always achieves a much better performance for the window query for both datasets. Therefore, the only reason that it works worse than STR R-tree for kNN search is that the approximated range window is larger than necessary which results in a worse performance due to that superfluous search.



(a) UNIFORM Dataset



(b) REAL Dateset

Figure 15: Tuning Time of kNN Queries having Various k

5 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. This paper addresses the problem of answering location-dependent spatial queries via broadcast channels. We first discuss the specific characteristics of broadcast environments and conclude that existing indexing structures are unsuitable for this new environment. Then a new index structure based on a space-filling curve, the Hilbert curve, is proposed to enable a linear broadcast (and later scanning) of objects in a multi-dimensional space. We devise exactly matching algorithms to answer window queries and kNN queries, along with a cost model. A simulation model is implemented to study the performance of our proposed indexing structure and compare it with some existing methods. The result shows that the performance of exact match based on the proposed structure outperforms other methods significantly in terms of tuning time, for both the synthetical dataset and the real dataset. However, its access latency is a little bit longer than that of some existing R-tree based indexes.

In this paper, we assume a centralized server which has full knowledge of all the objects. In a global environment, the coverage of a server is limited. Thus, some qualified data objects may located outside the coverage. We are looking into this problem by considering some cooperation strategies amongst the broadcast servers. As an immediate next step of the current work, we are investigating other kinds of LDSQs, e.g., continuous nearest-neighbor query. In addition, we are considering approximation functions to efficiently obtain the search range and to reduce the index size.

6 Acknowledgment

The research was supported by Research Grant Council, Hong Kong SAR, China under grant number HKUST6179/03E and HKUST6225/02E.

References

[1] G. D. Abowd, M. Ebling, G. Hung, H. Lei, and H.-W. Gellersen. Context-aware computing. *IEEE Pervasive Computing*, 1(3):22–23, July-September 2002.

[2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'95)*, pages 199–210, San Jose, CA, USA, May 1995.

[3] P. K. Agarwal, M. Berg, J. Matousek, and O. Schwarzkopf. Constructing levels in arrangements and higher order voronoi diagrams. *SIAM Journal on Computing*, 27(3):654–667, June 1998.

[4] N. Beckmann and H.-P. Kriegel. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'90)*, pages 322–331, 1990.

[5] S. Berchtold, B. Ertl, D. A. Keim, H. P. Kriegel, and T. Seidl. Fast nearest neighbor search in high-dimensional space. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, pages 209–218, February 1998.

[6] J. Cai and D. J. Goodman. General packet radio service in GSM. *IEEE Communications Magazine*, 35(10):122–131, October 1997.

[7] K. L. Cheung and W.-C. Fu. Enhanced nearest neighbour search on the r-tree. *SIGMOD Record*, 27(3):16–21, 1998.

[8] Microsoft Corporation. What is the directband network? URL at http://www.microsoft.com/resources/spot/direct.mspx, 2003.

[9] Spatial Datasets. Website at http://dias.cti.gr/ ~ytheod/research/datasets/ spatial.html.

[10] A. Datta, A. Celik, J. K. Kim, D. VanderMeer, and V. Kumar. Adaptive broadcast protocols to support power conservation retrieval by mobile users. In *Proceedings of the 13th IEEE International Conference Data Engineering (ICDE'97)*, pages 124–133, Birmingham, UK, April 1997.

[11] A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar. Broadcast protocols to support efficient retrieval from databases by mobile users. *ACM Transactions on Database Systems (TODS)*, 24(1):1–79, March 1999.

[12] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proceedings of the 17th IEEE International Conference on Data Engineering (ICDE'01)*, April 2001.

[13] C. Gotsman and M. Lindenbaum. On the metric properties of discrete space-filling curves. *IEEE Transactions on Image Processing*, 5(5):794–797, May 1996.

[14] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'84)*, pages 47–54, 1984.

[15] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*, September 2000.

[16] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases (SSD'95)*, pages 83–95, 1995.

[17] Gí. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, 1999.

[18] Q. L. Hu, D. L. Lee, and W.-C. Lee. Optimal channel allocation for data dissemination in mobile computing environments. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, pages 480–487, Amsterdam, The Netherlands, May 1998.

[19] Q. L. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'2000)*, pages 157–166, San Diego, CA, USA, February 2000.

[20] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air - organization and access. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 9(3), May-June 1997.

[21] I. Kamel and C. Faloutsos. On packing r-trees. In *Proceedings of the 2th International Conference on Information and Knowledge Management (CIKM'93)*, pages 490–499, Washington, DC, November 1993.

[22] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proceedings of 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 500–509, Santiago de Chile, Chile, September 1994.

[23] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB'96)*, pages 215–226, 1996.

[24] W.-C. Lee and D. L. Lee. Signature caching techniques for information broadcast and filtering in mobile environments. *ACM/Baltzer Journal of Wireless Networks (WINET)*, 5(1):57–67, 1999.

[25] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez. Str: A simple and efficient algorithm for r-tree packing. In *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*, pages 497–506, Birmingham, UK, April 1997.

[26] M. Minsky and S. Papert. Perceptrons: An introduction to computational geometry. *The MIT Press*, 1969.

[27] D. Moore. Hilbert curve. URL at http://www.caam.rice.edu/ dougm/ twiddle/Hilbert.

[28] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'95)*, pages 71–79, May 1995.

[29] M. Satyanarayanan. A catalyst for mobile and ubiquitous computing. *IEEE Pervasive Computing*, 1(1):2–5, January-March 2002.

[30] H. Schwetman. *CSIM user's guide (version 18).* Mesquite Software, Inc, http://www.mesquite.com, 1998.

[31] T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, pages 154–165, July 1998.

[32] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R$^+$-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB'87)*, pages 507–518, 1987.

[33] M. Vanco, G. Brunnett, and T. Schreiber. A hashing strategy for efficient k -nearest neighbors computation. In *Proceedings of Computer Graphics International*, Canmore, Alberta, Canada, June 1999.

[34] J. Xu, B. Zheng, W.-C. Lee, and D. L. Lee. Energy efficient index for querying location-dependent data in mobile broadcast environments. In *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE'03)*, pages 239–250, Bangalore, India, March 2003.

[35] B. Zheng, W. C. Lee, and D. L. Lee. Spatial index on air. In *Proceedings of the first IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, pages 297–304, Dallas-Fort Worth, Texas, USA, March 2003.

[36] B. Zheng, J. Xu, W. C. Lee, and D. L. Lee. Energy-conserving air indexes for nearest neighbor search. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT'04)*, Heraklion - Crete, Greece, March 2004.