Search Continuous Nearest Neighbors On the Air*†

Baihua Zheng
School of Information Systems
Singapore Management University
bhzheng@smu.edu.sg

Wang-Chien Lee
Penn State University
University Park, PA 16802
wlee@cse.psu.edu

Dik Lun Lee
Hong Kong University
of Science and Technology
dlee@cs.ust.hk

Abstract

A continuous nearest neighbor *(CNN) search retrieves the nearest neighbors corresponding to every point in a given query line segment. It is important for location-based services such as vehicular navigation tools 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 large overhead on bandwidth. Algorithms have been proposed recently to support CNN search in the traditional client-server service model. In this paper, we conduct a pioneering study on CNN search in wireless data broadcast environments. We propose two air indexing techniques, namely,* R-tree air index *and* Hilbert Curve air index*, and develop algorithms based on these two techniques to search CNNs on the air. A simulation is conducted to compare the proposed air indexing techniques with a naive broadcast approach. The result shows that both of the proposed methods outperform the naive approach significantly. The Hilbert Curve air index is superior for uniform data distributions, while the R-tree air index is a better choice for skewed data distributions.*

1. Introduction

With the increasing popularity of mobile devices and rapid advance of wireless technology, *pervasive computing* has received tremendous attention in the past few years. Once the vision of pervasive computing is realized, people equipped with mobile devices will be able to access information from anywhere at anytime, even when they are moving. Due to the mobility of people and their devices (which are referred as *mobile clients* for the rest of the paper), the query submission point may change continuously which makes retrieval of location-dependent data a challenge. A *Continuous Nearest Neighbor* (CNN) search is an important class of queries that find a set of nearest neighbors corresponding to every point in a given query line segment. Examples of CNN search are everywhere in our daily life, e.g., "find the nearest gas stations along the route from my current location to Boston on Highway I-93.[1]" Thus, efficient solutions for continuous queries are much needed. In this paper, we focus on techniques that support CNN search in pervasive computing.

In contrast to a client-server service model (on point-to-point communication) to support pervasive computing, our focus in this paper is on the wireless broadcast model because of its strength in scalability. Wireless data broadcast is very attractive because broadcast data can be accessed by an arbitrary number of mobile clients simultaneously. The increased number of clients do not incur any extra cost at the broadcast server. Thus, it is very suitable for heavy-loaded systems, or localized data services such as tourist guide for a museum, local traffic information, and event broadcast in Olympic games, where users tend to seek the same kind of information. For many years, companies such as Hughes Network System have been using satellite-based broadcast to provide broadband services. The recent announcement of Microsft MSN Direct Service (http://www.msndirect.com), based on *smart personal objects technology* (SPOT) and a continuous broadcast network using FM radio subcarrier frequencies, has further ascertained the industrial interest on and feasibility of using wireless broadcast to providing pervasive data services. Supporting location related data and CNN queries in wireless data broadcast is important due to the broad application base. With broadcast of location information for gas stations, highway exits, hotels, restaurants, etc. on a wireless channel, mobile clients will be able to tune in to find the nearest gas stations or hotels, etc., while moving on the highway.

This paper presents the first study, to the best of the authors' knowledge, on supporting CNN search in wire-

* Baihua Zheng's work was supported in part by Wharton-SMU Research Center, Singapore Management University

† Wang-Chien Lee's work was supported in part by NSF grant IIS-0328881

1 While a route may not be a line segment, it can be decomposed into multiple line segments.

less data broadcast services. Two air indexing techniques, namely, *R-tree air index* and *Hilbert Curve air index*, and CNN search algorithms are developed to facilitate this unique and important query. We have conducted a simulation based performance evaluation on these two indexing techniques and a naive broadcast approach. The result shows that both of the proposed techniques outperform the naive approach significantly. The result also provides good insights to the problem of CNN search on air and point out needed efforts to advance this new research direction.

The rest of this paper is organized as follows. Section 2 provides a review of the related work. Section 3 describes an existing CNN search algorithm based on R-tree and the necessary revisions to enable R-tree on the air. Section 4 describes the Hilbert Curve air index and associated CNN search algorithm. Section 5 evaluates performance of the two proposed air indexes and compares them to a naive broadcast approach for CNN search. Finally, Section 6 concludes this paper with directions of the future work.

2. Related Work

CNN Search. The problem of CNN search was first identified by Sistla et al. in [5]. For a given line segment, every object o in the answer set *dominates* a part of the line segment, i.e., o is the nearest neighbor of any query point lying on that partial line segment. An illustrative example is given in Figure 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 $\overline{sp_1}$; that is, O_1 is the nearest neighbor of any point lying on $\overline{sp_1}$. Similarly, O_2 dominates $\overline{p_1p_2}$ and O_4 dominates $\overline{p_2e}$. p_1 and p_2 are called *split points* [8] since they are the points at which the nearest objects along the line segment change.

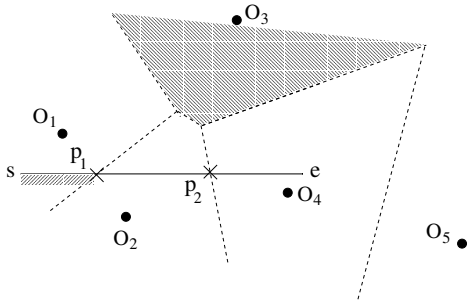


Figure 1. Example of CNN Search

There are some existing algorithms proposed to answer CNN searches. In [6], a sampling technique is employed to perform normal NN query at some pre-defined sampling points. A tight range to bound all the answers will be derived based on known answers obtained in first step to find the exact answers. However, its accuracy depends pretty much on the pre-defined sampling points on the query line. Tao et al. devised two search algorithms for CNN queries based on R-tree [7, 8]. The first algorithm is based on the concept of time-parameterized (TP) queries [7]. Treating a query line segment as the moving trajectory of a query point, the nearest object to the moving query point is valid only for a limited duration. Consequently, CNN queries can be transferred into TP queries. In that study, a new TP query was issued to retrieve the next nearest object once the valid time of the current query expired; that is, when a split point was reached. While the TP approach avoids the drawbacks of sampling, it is an incremental algorithm that needs to issue n NN queries in order to obtain the final answer set, where n is the number of objects in the final answer set. The second algorithm, proposed later, navigates R-tree based on certain heuristics [8]. The whole answer set is obtained within one single navigation of R-tree.

Index On Air. In addition to the typical performance concern of access efficiency, *power consumption* of mobile device is another crucial performance criteria for pervasive computing since all the device features and applications are directly or indirectly dependent on power availability. Thus, *access latency* and *tuning time* are used as the performance criteria for our study. The former is the period of time elapsed from the moment a query is issued to the moment when all the requested data are received. The latter equals to the period of time spent by a mobile device staying *active* in order to obtain the requested data, which reflects the power consumption of client.

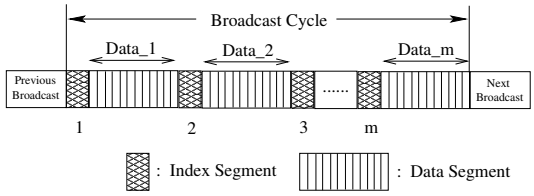


Figure 2. (1, m) Interleaving Technique on the Broadcast Channel

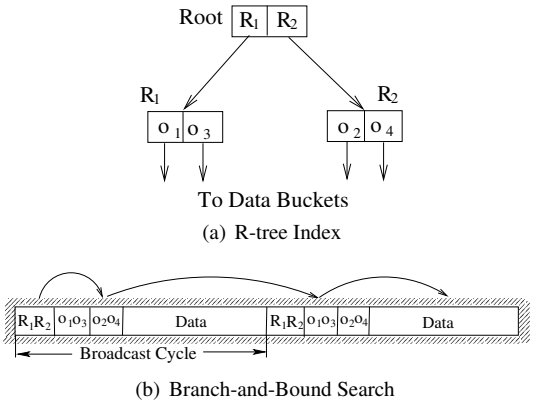
To facilitate power saving via wireless data broadcast, index information, called *air index*, is typically broadcast along with the data. Studies show that interleaving air index with data objects may help mobile devices to avoid receiving unwanted data and reduce power consumption of mobile devices. By looking up the air index, a device can predict the arrival time of its desired data so that it can slip into *doze mode* and switch back to *active mode* only when the data of desire arrives, thus substantially saves battery resource. Broadcast organizations that properly interleave index and data on the broadcast channel can significantly

improve (power) energy efficiency by trading off some access efficiency. Figure 2 demonstrates the well-known $(1, m)$ scheme, which broadcasts the whole index m times preceding every fraction $(\frac{1}{m})$ of the whole data objects during every broadcast cycle [3].

Most of the previous studies addressed only the *dissemination* and *scalability* aspects of the wireless data broadcast without taking into account the characteristics of its applications. In this paper, we address the *application* aspect by examining the issues of processing CNN search in wireless data broadcast environments.

3. CNN Search on R-Tree Air Index

R-tree, and it variants, have been widely used to support various spatial queries [2]. Thus far, all existing algorithms proposed for CNN search are based on R-tree. In this section, we first analyze the problem with broadcasting R-tree on the wireless channel and then describe a CNN search algorithm designed for disk-based R-tree. Finally, we adapt the algorithm to make it suitable for broadcast environments, which we call R-tree air index.



(a) R-tree Index

(b) Branch-and-Bound Search

Figure 3. Linear Access on Wireless Broadcast Channel

3.1. R-Tree Air Index

A search algorithm based on R-tree typically expands the search space around the query point using a branch-and-bound approach. Consequently, the navigation order of R-tree is dynamically determined based on the position of the query point. This usually requires backtracking before the target leaf node is found. Thus, R-tree is better supported by random access storages, such as memory and disk, but not the wireless channels.

Information is broadcast based on a pre-defined sequence and it is only available at the moment when it is broadcast. Consequently, backtracking may incur a significant access latency. Figure 3 depicts an example. Assuming that a search algorithm first visits root node, then the node R_2, and finally R_1, while the server broadcasts nodes in the order of root, R_1, and R_2. Consequently, if a client wants to visit 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. Thus, search algorithms need to be revised to fit the linear streaming property of wireless data broadcast.

3.2. CNN Algorithm for Disk-Based R-tree

A CNN search algorithm for disk-based R-tree was proposed in [8]. The search starts from the root of R-tree. For each branch of R-tree, which is represented as an MBR, it checks the following two heuristics: RT1) whether the minimal distance between the MBR and the query line segment is shorter than the maximal distance between a point on the query line segment and its nearest neighbor obtained thus far in the processing procedure, and RT2) the minimal distance between the MBR and some point in the query line segment is shorter than the distance between the point and its NN. R-tree branches which satisfy both of RT1 and RT2 will be visited in accordance with their minimal distance to the query line segment. This traversal order is expressed as an additional heuristic RT3 in [8]. This traversal process is recursively carried out until a leaf node is reached. The objects in the leaf node are checked to determine whether they should replace some of the NNs found so far. The process then backtracks to the upper level to continue the search. The heuristics used in the algorithm are explained in details below.

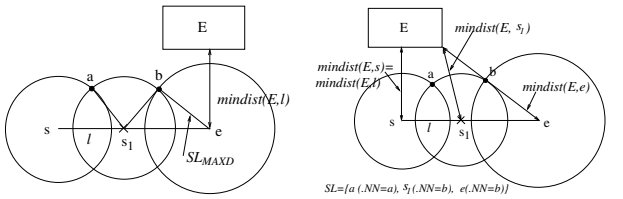
Heuristic RT1. Given a query segment l and a R-tree branch represented by an MBR E, the MBR E may contain qualified data objects *only if* $mindist(E, l) < SL_{MAXD}$, where $mindist(E, l)$ denotes the minimal distance between E and l, and SL_{MAXD} denotes the maximal distance between a point on l and its corresponding nearest neighbor.

As shown in Figure 4(a), the minimal distance between the query line segment l and a MBR E is larger than the current SL_{MAXD} (i.e., $dis(b, e)$ in the figure), which means no objects within E can be a nearest neighbor to any point on the query line segment. Hence, the R-tree branch associated with E can be pruned.

Heuristic RT2. Given a query segment l and a R-tree branch represented by an MBR E, E needs to be searched *if and only if* there exists a point s_i such that the dis-

tance between s_i and its nearest neighbor is larger than $mindist(s_i, E)$.[2]

An example is depicted in Figure 4(b). The MBR E will not be visited since $dis(s, a) < dis(s, E)$, $dis(s_1, b) < dis(s_1, E)$, and $dis(e, b) < dis(e, E)$. For the detailed proof of the heuristics, please refer to [8].



(a) Heuristic RT1 (b) Heuristic RT2

Figure 4. Heuristics of R-tree Index

Heuristic RT3. The R-tree branches that satisfy heuristics RT1 and RT2 are visited in ascending order of their minimal distances to the query line segment l.

RT3 dynamically determines the order of R-tree branches to be traversed, which allows the algorithm to obtain answers quickly and to further prune branches with no need to visit. However, this requires backtracking on R-tree. In wireless data broadcast, the nodes of R-tree must be accessed when they are broadcast on the air. Thus, the search algorithm needs to be revised to adapt to the air index.

3.3. CNN Algorithm for R-tree Air Index

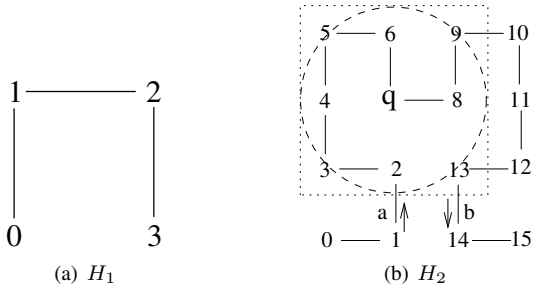
As explained in Section 3.1, the traversal order dynamically determined by RT3 will result in an unacceptable access latency. To eliminate this drawback, the branches of the R-tree air index must be visited according to their broadcast order. Heuristics RT1 and RT2 can still be applied to prune the unnecessary search branches. In summary, the CNN search algorithm revised for R-tree air index works as follows. It visits the tree branches sequentially, in the same order as the predefined broadcast sequence. The mobile clients will only listen to (i.e., traverse) the branches that satisfy RT1 and RT2 and stay in doze mode when the other branches are broadcast. In the visited leaf nodes, checking and updating is the same as in the original algorithm.

4. Search CNN on Hilbert Curve Air Index

To adapt to the linear streaming property of the wireless data broadcast channel, *Hilbert Curve (HC) index* was proposed in a previous work by the authors to process window queries and k nearest neighbor (k-NN) search on the air [10]. In this paper, we extend the Hilbert Curve index to answer CNN queries. In this section, we first summarize the HC index and then develop a new algorithm based on it to answer CNN queries via wireless data broadcast.

4.1. Hilbert Curve Index

Hilbert curve is a space-filling curve that visits every point in an n-dimensional grid space exactly once without crossing itself. Figure 5(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 the Hilbert curve of order $(i-1)$, which may be strategically rotated and/or reflected to fit the new curve. The Hilbert curves of orders 2 is depicted in Figure 5(b). As shown, a Hilbert curve maps points in an n-dimensional space to a 1-dimensional linear space. The numeral labels, called *index values*, represent the positions of points along the curve, by which data objects can be identified. In H_2 curve, the point of index value 2 denotes the point (1, 1) in the two-dimensional space (as shown in Figure 5(b)). Therefore, B^+-tree can be employed to index the data objects using their index values as the key. Thus, the leaf nodes of the HC index tree consist of 2-tuples, $<indexvalue, ptr>$, where ptr is the arrival time of the referenced data object.



(a) H_1 (b) H_2

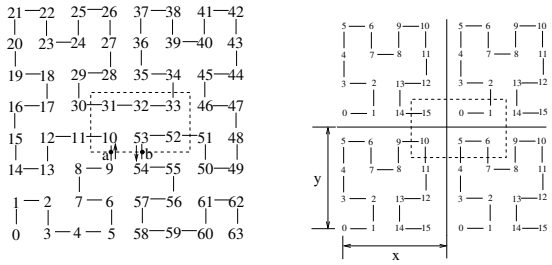
Figure 5. Hilbert Curves of order 1 and 2

Two steps, selection and filtering, are performed to process a window query. First a HC value boundary will be determined to include all the potential candidates, based on the given query window. All the points within the boundary need further checking to filter out those unqualified candidates. To process a k-NN search, we proposed a range estimation algorithm based on the locality property of the Hilbert curve. First, the k objects closest to the query point along the Hilbert curve are found and a minimal circle centered at the query point is constructed to contain all those k objects. This circle, which may contain more than k objects, is guaranteed to include the k

2 This heuristic can be checked efficiently by maintaining and computing based on a list of found split points instead of computing based every point on the query line segment.

nearest neighbors. Second, all the candidates within the circle will be checked and sorted according to their Euclidean distance to the query point. The top-k objects are the answers.

As illustrated in Figure 5(b), the dashed rectangle is the query window, which determines [a, b] to be the range of all the candidates. Consequently, the client only needs to check the objects between a and b inclusive on the curve, and eliminate those that are not lying within the window. It also shows a 4*NN* query issued at point q. By scanning the linear HC, objects 5, 6, 8, and 9 are detected to be 4NN objects of q on the Hilbert curve. Consequently, the dashed-line circle centered at q and bounding these 4 objects can be derived which contains at least 4 objects. By performing a window query based on this circle and ordering the objects based on Euclidean distance, the exact 4NN objects will be returned (i.e., objects 2, 4, 6, and 8 within the dashed circle).



(a) Before Partition (b) After Partition

Figure 6. Grid Partition

Performance of the above search algorithms depends on the locality of the data objects in the one-dimensional space. Noticing that the algorithms will have to check more points than necessary in an area where the nearby points in the original search space become widely separated in the linear Hilbert curve, a space partition algorithm is also proposed to reduce the extra searching cost in [10]. Figure 6 shows an example and please refer to [10] for detailed information. In the original space, all the points whose index values between 9 and 54 need checking with an assumption that the dashed-line rectangle is a query window. After applying the 2×2 partition, we only need to check those objects with index values between 0 and 1 for the sub-grid in the upper-right quadrant, objects whose value is 15 for the sub-grid in the upper-left quadrant, and so on.

4.2. CNN Algorithms for Hilbert Curve Air Index

Similar to the k-NN search algorithm based on HC index, the strategy for processing CNN search is to first determine a search range on the Hilbert curve and then find out the exact answer. The CNN search algorithm comprises three steps: 1) obtain the nearest neighbors to the two endpoints of the query line segment. Based on these two 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 detailed algorithm, we first develop five heuristics for processing CNN search. The first two heuristics are used in Step 1 to determine the approximated search range. The last three ones are used in Step 3 to obtain the final answer set from the candidate set of data objects. Table 1 defines some terminologies to facilitate the description.

Notation	Description
$dis(q, q')$	Euclidean distance between points q and q'
$proj(q, l)$	the projection of point q on the line l;
$bis(q, q')$	the perpendicular bisector of line segment connecting points q and q';
$NN(q)$	the nearest neighbor of the query point q
$CNN(\overline{se})$	the answer set containing all the 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

Table 1. Terminology Definition

Heuristic HC1. Given a query line segment $\overline{se}$, $NN(s)$=O_s, and $NN(e)$=O_e, $\{O_s, O_e\} \subseteq CNN(\overline{se})$

The above heuristics is intuitive. Since s and e are two points on the query line segment, their nearest neighbors are part of the final answer set.

Heuristic HC2. For a query segment $\overline{se}$, if $NN(s)$= $NN(e)$ =O_i, then *CNN*($\overline{se}$) = $\{O_i\}$.

The above heuristics can be shown with the *Voronoi Diagram*, which partitions a space into disjoint *Voronoi Cells (VCs)* based on locations of data objects in the space. For any query point located within a VC, its corresponding object is the nearest neighbor to that query point. As shown in Figure 1, the Voronoi Diagram partitions the space into 5 parts denoted by the dashed line, according to the positions of given objects. The shadowed polygon is the corresponding VC of object O_3, which means O_3 is the only nearest neighbor to any query point inside the shadowed polygon. Based on computational geometry, VCs are convex. Since $NN(s)$= $NN(e)$=O_i, both endpoints and hence the query line segment $\overline{se}$ lie inside the VC of object O_i. Therefore, object O_i is the nearest neighbor to any query point along the query line segment.

In Step 1 of the CNN search algorithm, the nearest neighbor(s) of the endpoints of the given line segment are ob-

Algorithm 1 Search Range Determination

Input: query line segment $\overline{se}$, $NN(s)$ and $NN(e)$;
Output: a search range;
Procedure:

1: find the intersection point, m, of $\overline{se}$ and $bis(NN(s), NN(e))$;
2: r = dis ($NN(s)$, m);
3: draw a line l passing $NN(s)$ and perpendicular to line $\overline{se}$;
4: find two points P_1 and P_2 on l such that dis(P_1, m) = dis(P_2, m) and dis(P_1, P_2) = $2r$;
5: draw a line l' passing $NN(e)$ and perpendicular to line $\overline{se}$;
6: find two points P_3 and P_4 on l' such that dis(P_3, m) = dis(P_4, m) and dis(P_3, P_4) = $2r$;
7: return the rectangle bounded by P_1, P_2, P_3 and P_4;

tained and included in the final answer set (based on Heuristic HC1). If the nearest neighbors to both endpoints are the same, the final answer set can be returned directly without further processing. Otherwise, a search range bounding all the candidate objects is determined based on Algorithm 1. Figure 7 shows a search range and illustrates a proof that the search range indeed bounds all the objects in the final answer set.

Claim: Given a query line segment, the search range determined by Algorithm 1 contains all the data objects in the final answer set.

Proof: Without loss of generality, a horizontal query line segment is assumed. Any non-horizontal line segment can be mapped to the x-axis based on coordinate transformation.

Given a point q' on $\overline{me}$ where r'= $dis(q', O_e)$, as shown in the Figure 7. If object O_e is not the nearest neighbor of point q', NN(q') should be located within the circle cir(q', r'). Parts of this circle may fall outside of the search range (as specified by the rectangle bounded by P_1, P_2, P_3, and P_4). If a part of the circle falls on the right side of the line $\overline{O_e O_{e'}}$, it must be inside cir(e, dis(e, O_e)). Since object O_e is the nearest neighbor of point e, there is no other objects within cir(e, dis(e, O_e)). As a result, there is no data objects located on the right of the search range to be included in the final answer set. Similarly, it can be shown that all the data objects in the final answer set are not located on the left side of the search range. Finally, data objects located beyond the top and bottom of the search range will not be included in the answer set because their shortest distances to the line segment will be greater than r, the longest distance from the line to either O_s or O_e. Therefore, the search range contains all the objects in the final answer set.

Similarly, given a point q' on $\overline{sm}$, we can show that the search range contains all the objects in the final answer set. Hence, our claim is proven.

□

Once the search range is determined, a window query is issued to obtain the candidate objects in the search range. Heuristics HC3-HC5 are developed to filter the candidate set for the real answer. In the following, we assume that all objects O_i inside the approximate search range are sorted in

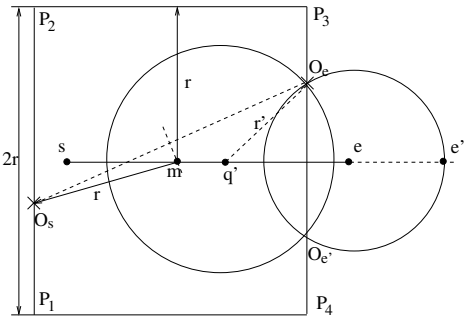
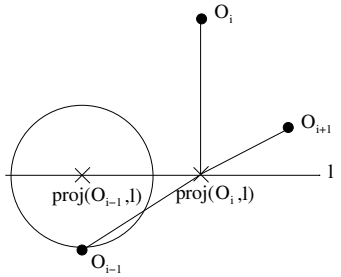
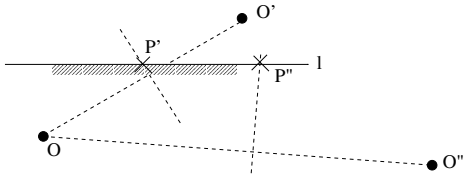


Figure 7. Search Range for k-NN Queries

ascending order of the x-coordinates, i.e., $O_i.x < O_{i+1}.x$. Here, $O_i.x$ refers to the x-coordinate of point O_i.



(a) Heuristic HC3 and HC4



(b) Heuristic HC5

Figure 8. Illustrative Examples of some Heuristics

Heuristic HC3. For an object O_i in the search range, if dis($proj(O_i, l)$, O_{i-1}) < dis($proj(O_i, l)$, O_i), and dis($proj(O_i, l)$, O_{i+1}) < dis($proj(O_i, l)$, O_i), $O_i \notin CNN(\overline{se})$.

The above heuristic is illustrated in Figure 8(a). The shortest distance between O_i and any point p on the query line segment is dis($proj(O_i, l)$, O_i). If this shortest distance is longer than both of dis($proj(O_i, l)$, O_{i-1}) and dis($proj(O_i, l)$, O_{i+1}), O_i is not the nearest neighbor for any part of the query line segment and thus does not belong to the final answer set.

Heuristic HC4. Given object O and query line segment l, if cir($proj(O, l)$, $dis(O, proj(O, l))$) does not contain any objects other than O, O is the nearest neighbor to the point $proj(O, l)$.

As shown in Figure 8(a), if there are no other objects within cir($proj(O_{i-1}, l)$, $dis(O_{i-1}, proj(O_{i-1}, l))$, it means that O_{i-1} is the nearest neighbor to the point $proj(O_{i-1}, l)$. Therefore, O_{i-1} is in the final answer set. This heuristic allows some obvious answers to be found at the very beginning of the filtering process.

Heuristic HC5. Given a line segment l, which is assumed to be dominated by O during the filtering process, if an object O' is the nearest neighbor to some point on l, the perpendicular bisector of O' and O intersects l at a split point P'.

Algorithm 2 Filtering

Input: query line segment $\overline{se}$, candidate answer set;
Output: final answer set;
Procedure:

```
1: Perform coordinate transformations to make the query line segment se lies on the x-axis;
2: Sort the objects in the candidate set in ascending order based on their x-coordinates;
3: Employ Heuristics HC3 and HC4 to eliminate out invalid objects and identify Nearest Neighbor, respectively. The rest of the candidate objects are labelled as unknown;
4: while the number of unknown objects > 0 do
5:    for each two successive valid answers O_i and O_{i+1} in the candidate set do
6:       find the intersection i of bis(O_i, O_{i+1}) and se;
7:       find the nearest neighbor o' of point i from the candidate set;
8:       if o' ∉ {O_i, O_{i+1}} then
9:          mark o' as NN;
10:   for each unknown object O_un in the candidate set do
11:      flag := 0;
12:      for each valid object O_NN in the answer set do
13:         if the intersection of bis(O_un, O_NN) and se is within the dominate segment associated with O_NN then
14:            flag=1; break;
15:      if flag == 0 then
16:         mark object O_un as invalid;
17: Return the final answer set consisting all of the NN objects;
```

If an object is the nearest neighbor to some point on a line segment, this line segment should pass through the VC of that object. As shown in Figure 1, the query line segment crosses the VCs of objects O_1, O_2, O_4, and as such should have one or two intersections with the VC of an object belonging to the answer set. For Voronoi Diagram, any edge of a VC is the perpendicular bisector of two objects. Therefore, if we find that the perpendicular bisectors of a given object and all known nearest neighbors do not intersect with the query line segment, the given object definitely is not in the answer set and thus can be thrown out of the candidate set.

As shown in Figure 8(b), assume that O is a known nearest neighbor to the shadowed query line segment, O' is an NN object to some point on this query line segment since the perpendicular bisector of O' and O intersects with it at point p'. However, O'' does not belong to the answer set since the intersection p'' is not inside the segment.

Based on the above heuristics, Step 3 of CNN search is explained in Algorithm 2. In the algorithm, an object in the candidate set is in one of the following states, namely, *NN*, *invalid*, and *unknown*. Just as these names suggested, an *NN* object is a nearest neighbor to the query line segment. An invalid object does not belong to the final answer set. An unknown object is one that needs further checking. An NN object in the final answer set dominates a segment of the query line, specified by two split points, produced by intersecting the query line segment with the perpendicular bisectors of this object and two other objects in the final answer set. Furthermore, the nearest neighbor(s) to the endpoints of the query line segment have their dominated line segments ended at their corresponding endpoints.

In summary, CNN search requires three scans of the HC index on air. The first two scans obtain the nearest neighbors of the two endpoints of the query line segment. Based on the detected NNs, a search range is determined as described in Algorithm 1. Thus, a window query can be issued to obtain the candidate set (which needs a further HC index scan). Finally, the filtering algorithm obtains the final answer set.

5. Performance Evaluation

This section evaluates the performance of the two proposed techniques, R-tree air index and Hilbert Curve air index for supporting CNN search in wireless data broadcast environments. There are two datasets used in the evaluation. In the first dataset (called UNIFORM), 10,000 points are uniformly generated in a square Euclidean space. The second dataset (called REAL) contains 1102 hospitals in south California, which is extracted from the point dataset available at [1].

For R-Tree, since the objects are available a priori, the *STR* packing scheme [4] is employed for its superior performance. In the following, STR R-tree denotes this implementation. As explained in Section 2, the search algorithm designed for disk indexing cannot be directly employed for air indexing. In this paper, we propose to broadcast the R-Tree in the depth-first order. Thus, the CNN search algorithm for R-tree air index scans the MBRs sequentially, while skipping R-tree branches based on heuristics developed in [8].

The system parameters for our evaluation are set as follows. In each packet, two bytes are allocated for the packet id. Two bytes are allocated for the time pointer and four bytes for each coordinate. The size of a data item is set to be 1 Kilobytes. The packet capacity is varied from 64 bytes to 1024 bytes in our experiments.[3] The results are obtained based on 1,000,000 randomly generated queries. The para-

3 For the wireless channel, the page capacity is normally assumed in the order of 100 bytes [3].

meter *SegLengthRatio* defines the ratio of the query line segment length to the total side length of the search space, with 0.1 as the default value. A 4×4 partition is applied to improve performance of HC air index, and the detailed concept of partition can be found in [10].

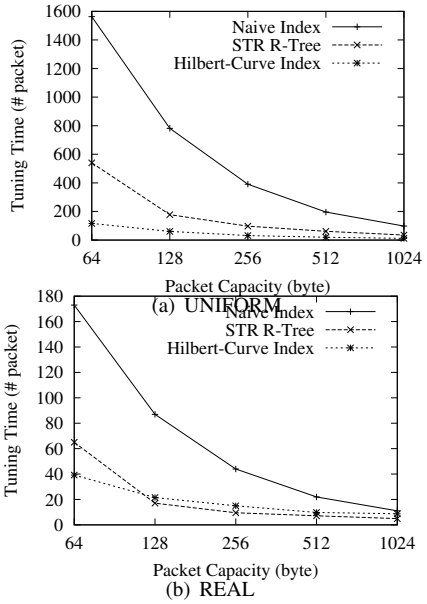


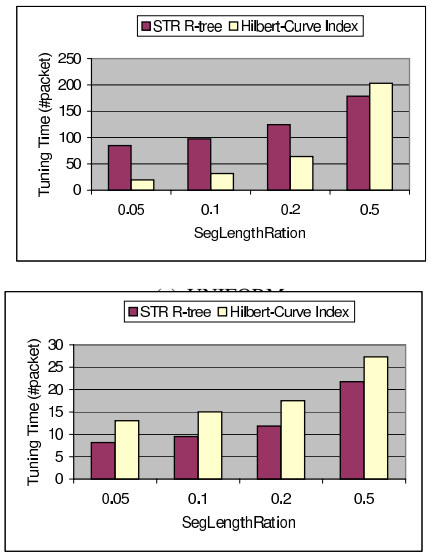
Figure 9. Tuning Time ($SegLengthRatio$ **=**0.1**)**

5.1. CNN Search Performance

In this section, we first compare the search performance based on R-tree air index and Hilbert Curve air index, with a fixed *SegLengthRatio*, by varying the packet size. Next, the same experiment is conducted by fixing the packet size but varying *SegLengthRatio* from 0.05 to 0.5.

Figure 9 depicts the tuning time of the two techniques in terms of the number of packet accesses, along with the performance of a naive approach which serves as the comparison baseline. For the naive approach, each object is represented by the coordinates and a pointer. The client has to retrieve all of the objects to process a CNN query. It is observed that the two proposed indexes improve the performance significantly. For the UNIFORM dataset, HC air index performs extremely well, with an average of 90.8% improvement, while R-tree air index has 70.1% average improvement, when compared to the naive approach. For the skew dataset, R-tree air index outperforms the naive method by 68.8%, while HC air index has 58.9% improvement.

It is noticed that R-tree air index is more efficient in responding to the REAL dataset which is skew, and HC index is more suitable for uniform dataset. The differences can be explained as follows. The data distribution has an important impact on the locality of the Hilbert curve, and hence affects the performance of HC air index. For the uniform dataset, the search range obtained by scanning the index is quite precise. For the real data, which has a skewed distribution, the search range may be larger than necessary and thus it causes excessive accesses.



(b) REAL

Figure 10. Tuning Time with Various $SegLengthRatio$

Figure 10 shows the result when packet size is fixed at 256 bytes while the length of the query line segment is valuable. Consistent with the previous experiment, the HC air index is suitable for uniform data distribution and R-tree air index is more suitable for skewed data distribution. The other observation is that R-tree air index has a more stable performance when the length of query line segment becomes longer.

Figure 11 depicts the expected access latency, employing (1, m) index organization scheme [3]. In this comparison, a naive search method without using any index is used as the base line. Its access time is set as 1. As shown, Hilbert Curve air index incurs a larger index size since it requires scanning the index twice for a k-NN search. Therefore, the index is broadcast twice consecutively in order to allow the clients to process CNN search within one broadcast cycle. Hence, this duplication results in a larger access latency.

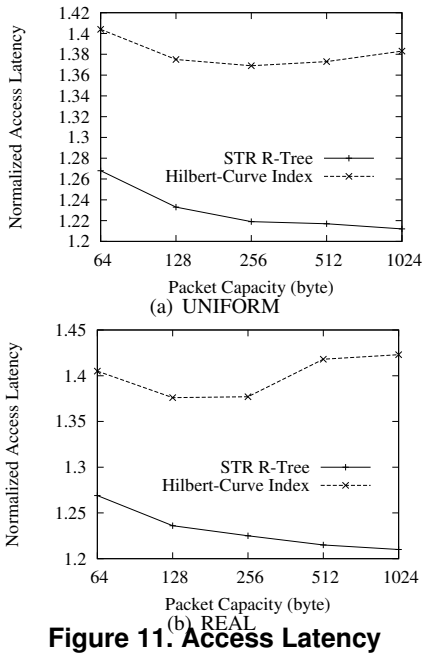


Figure 11. Access Latency

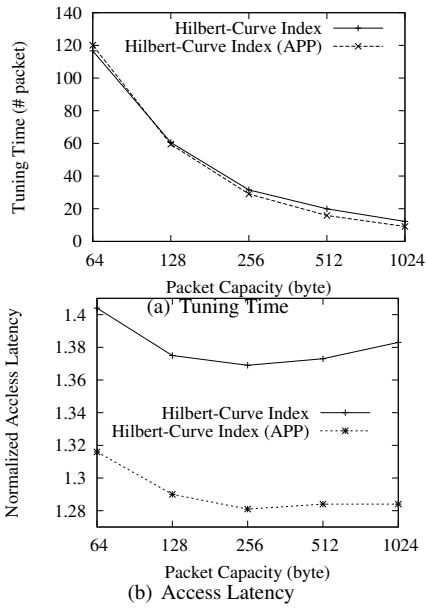


Figure 12. Improvement obtained from Approximation Function

5.2. Improvement obtained from Approximation Function

As we observe earlier, the Hilbert Curve air index is a more energy-efficient access method for CNN search under a uniform data distribution. An issue faced by the Hilbert Curve air index, however, is that it requires multiple scans of the index to return the final answer set. Due to the linear streaming property of the wireless data broadcast, a Hilbert Curve air index actually consists of consecutive broadcast of multiple B^+-tree, which extends the clients' average access latency. One way to reduce the index size is to employ an approximation function that determines a search range for k-NN search. This approximation function has been shown to be effective (though not 100% accurate) by both mathematical analysis and simulations (details can be found in [9]). By approximating the initial search range, the index is scanned only twice for finding the final answer set for CNN: one scan for obtaining the nearest neighbors of the two endpoints of the query line segment; the other scan for the window query to retrieve all the candidate objects. Therefore, the size of the Hilbert Curve air index can be significantly reduced. Figure 12 shows the simulation result for the UNIFORM dataset. Comparing to the original algorithm without any approximation, the access latency is greatly improved and is quite close to that of R-tree air index. The accuracy of the approximation, which is not shown here, is observed to be 100% accurate for all experiments we conducted based on the UNIFORM dataset.

6. Conclusion

With the popularity of the mobile devices and advance of the wireless networking, mobile applications have started to enter our daily life. Location, which plays an important role in many mobile applications, has received a lot of attentions from both the research and industry communities in recent years. In this paper, we investigate the support of an important class of location-based queries, namely, continuous nearest neighbor (CNN) search, in wireless data broadcast. To the best of our knowledge, this is the first attempt to support CNN search in a wireless broadcast system (e.g., DirectBand network of Microsoft and DirecWay of Hughes Network Systems).

In this paper, we propose two air indexing techniques, namely R-tree air index and Hilbert Curve air index for processing CNN search on air. We propose to broadcast R-tree in depth-first order and revise its CNN search algorithm to adapt to the wireless data broadcast model. Furthermore, we develop several heuristics and a new CNN search algorithm based on Hilbert Curve air index. Finally, a simulation is conducted to evaluate the performance of the proposed air indexing techniques for CNN. The simulation result shows that the Hilbert Curve air index achieves a superior performance on uniformly distributed data, especially after employing an approximation technique, while the R-tree air index provides an excellent performance for the skew data distribution.

While the air indexing techniques proposed in this study

still have room for improvement, we have opened a new research direction in the field of pervasive computing. Based on lessons learned, we are working on a more versatile indexing scheme that can adapt to both of the uniform and skew data distributions.

References

[1] S. Datasets. Website at http://dias.cti.gr/ ∼ytheod/research/datasets/ spatial.html.

[2] 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.

[3] 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.

[4] 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.

[5] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*, pages 422–432, Birmingham, UK, April 1997.

[6] Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD'01)*, pages 79–96, Los Angeles, CA, USA, July 2001.

[7] Y. Tao and D. Papadias. Time parameterized queries in spatio-temporal databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'02)*, pages 334–345, Madison, Wisconsin, USA, 2002.

[8] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*, Hong Kong, 2002.

[9] B. Zheng, W. C. Lee, and D. L. Lee. Search k nearest neighbors on air. In *Proceedings of the 4th International Conference on Mobile Data Management (MDM'03)*, pages 181–195, Melbourne, Australia, January 2003.

[10] 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.