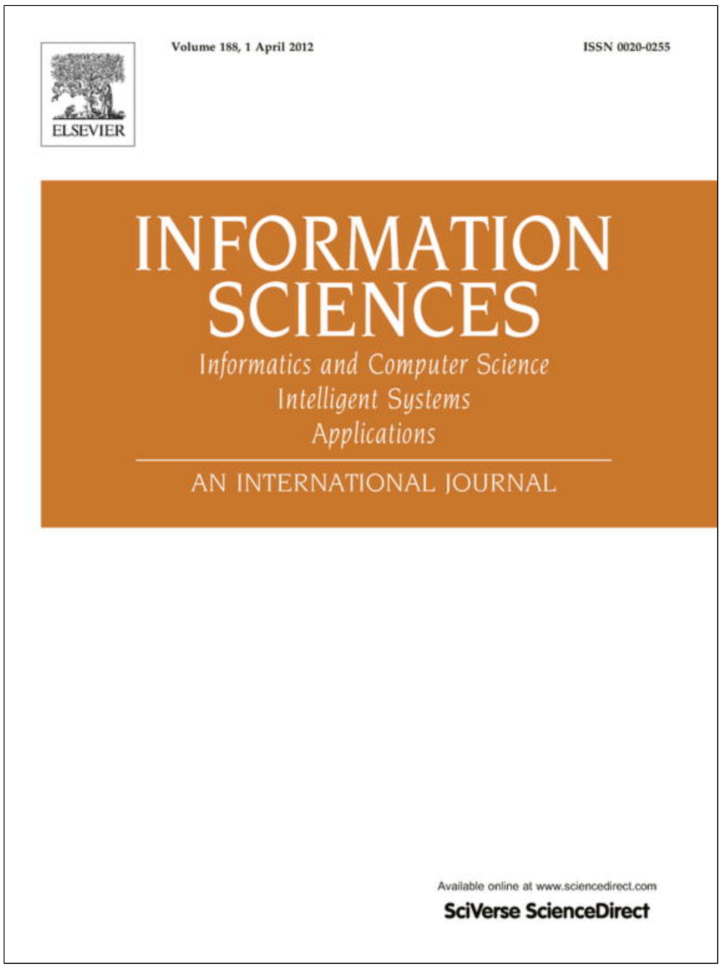

Volume 188, 1 April 2012
ISSN 0020-0255
ELSEVIER
INFORMATION SCIENCES
Informatics and Computer Science
Intelligent Systems
Applications
AN INTERNATIONAL JOURNAL
Available online at www.sciencedirect.com
SciVerse ScienceDirect

Information Sciences 188 (2012) 64–79

Contents lists available at SciVerse ScienceDirect

Information Sciences

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

Processing generalized k-nearest neighbor queries on a wireless broadcast stream ☆

HaRim Jung [a], Yon Dohn Chung [a,*], Ling Liu [b]

[a] Department of Computer Science and Engineering, College of Information and Communication, Korea University, Seoul 136-713, Republic of Korea
[b] College of Computing, Georgia Institute of Technology, Atlanta, GA 30280, USA

ARTICLE INFO

Article history:
Received 6 April 2010
Received in revised form 14 September 2011
Accepted 7 November 2011
Available online 18 November 2011

Keywords:
Mobile databases
Wireless broadcasting systems
Location-based services
Generalized k-nearest neighbor (GkNN) queries

ABSTRACT

In this paper, we investigate the problem of processing generalized k-nearest neighbor (GkNN) queries, which involve both spatial and non-spatial specifications for data objects, in a wireless broadcasting system. We present a method for processing GkNN queries on the broadcast stream. In particular, we propose a novel R-tree variant index structure, called the bit-vector R-tree (bR-tree), which stores additional bit-vector information to describe non-spatial attribute values of the data objects. In addition, each node in the bR-tree stores only one pointer to its children, which makes the bR-tree compact. We generate the broadcast stream by multiplexing the bR-tree and the data objects in the broadcasting channel. The corresponding search algorithm for the broadcast stream is also described. Through a series of comprehensive simulation experiments, we prove the efficiency of the proposed method with regard to energy consumption, latency, and memory requirement, which are the major performance concerns in a wireless broadcasting system. Furthermore, we test the practicality of the proposed method in a real prototype system.

© 2011 Elsevier Inc. All rights reserved.

1. Introduction

With technological advances in wireless networks and the development of navigation systems such as GPS, location-based services (LBSs) have emerged as potential applications in mobile computing environments [5,12,13,19,23,25,27,40,41]. Mobile users equipped with hand-held devices (e.g., smart phones, PDAs, and laptops) can now access a variety of valuable information from anywhere and at any time. Although most current LBSs heavily rely on a point-to-point communication method, they might suffer from drastic performance degradation due to the overwhelming server workload and communication contention when the population of users becomes extremely large.

Wireless broadcasting can be an attractive complementary communication method for provisioning LBSs [13,19,23,25,40,41]. In a wireless broadcasting system, the server periodically broadcasts data objects through a wireless broadcasting channel. When clients[1] receive queries from users, they independently tune into the channel and process the queries by continuously scanning the broadcast stream. In this way, the server pushes the query processing task entirely to the client side; thus, the server load is independent of the user population. In addition, the broadcasting channel can be simultaneously accessed by any number of clients; hence, communication contention among the clients is avoided.

Energy-efficiency is one of the most important considerations in a wireless broadcasting system owing to a client's limited battery power. Continuous scanning a broadcast stream causes a client to quickly consume its energy; hence, *air indexing*

☆ This work was supported by a Grant from the Korea Research Foundation, funded by the Korean Government (KRF-2008-313-D00856).
* Corresponding author.
E-mail addresses: harim3826@korea.ac.kr (H. Jung), ydchung@korea.ac.kr (Y.D. Chung), lingliu@cc.gatech.edu (L. Liu).
[1] Hereafter, for simplicity, we use the term 'client' to refer to a hand-held device carried by a mobile user.

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.11.007

strategies are widely adopted to address this problem [7–10,13,16,18,19,23,25,37,38,40–42]. The basic idea is to include the index information about the data objects in the broadcast stream so that a client is able to predict the appearance times of the required data objects in the broadcast stream. Consequently, a client can reduce its energy consumption by selectively scanning only the necessary portions of the broadcast stream. However, air indexing strategies might unnecessarily lengthen the broadcast stream owing to the additional index information, thereby resulting in greater user-perceived *latency*. The larger the index size, the greater is the latency; therefore, it is important to keep the index size as small as possible.

The main function of LBSs is to process *location-based queries* such as *range queries* and *k-nearest neighbor* (*k*NN) *queries*. A range query is a request for all the data objects located within a certain distance from a user's location. On the other hand, a *k*NN query is a request for the *k* closest data objects from a user's location. However, such conventional location based queries consider only the *spatial distance* between the data objects and users' location. Consequently, they may be insufficient for supporting complex real-life LBSs, which usually consider both the spatial and the non-spatial attributes of data objects in order to reflect users' personalized preferences.

In this paper, we introduce a generalized version of location-based queries, called the *generalized k-nearest neighbor* (G*k*NN) *query*. Assume a set of data objects $\mathcal{D}$, with each data object being associated with both spatial attributes (i.e., 2-dimensional coordinates) and a set of non-spatial attributes. Then, the G*k*NN query q, issued by a mobile user over $\mathcal{D}$, specifies the user's location and target values on the non-spatial attributes of interest. The result of q is a set of the k closest data objects in $\mathcal{D}$, where the degree of closeness between each data object and q is determined according to not only the spatial distance from the user's location but also the *non-spatial distance* from the user-specified target values. Example 1 illustrates a motivational example of the G*k*NN query.

Example 1. Consider a server that periodically broadcasts the dataset of local restaurants $\mathcal{R} = \{r_1, \ldots, r_8\}$ shown in Fig. 1, where the locations of the restaurants are shown with their average meal prices. Suppose that a client has received the G*k*NN query q from a user, say Smith, at location $q.p = (10,8)$, e.g., "Find a nearby restaurant ($k = 1$) whose average meal price is around \$10." If both the location and the average meal price of a restaurant are equally important to Smith, the client tunes into the broadcast channel to process q, and it retrieves the restaurant r_3, which is the best match for Smith's specifications. However, note that because a traditional *k*NN query supported by current LBSs takes only spatial attributes as the input, the client retrieves the spatially closest restaurant r_7 whose average meal price might be beyond Smith's budget.

In this paper, we address the problem of processing G*k*NN queries in a wireless broadcasting system. A straightforward approach is to construct a high-dimensional index structure and to include it in the broadcast stream so that clients can selectively scan the broadcast stream. Unfortunately, in a wireless broadcasting system, most high-dimensional index structures suffer from considerable performance degradation as the number of dimensions increases. This is mainly due to the fact that the basic unit of the broadcast stream typically has a small capacity, resulting in a smaller node fan-out. Consequently, the average height of the index structures increases, and excessive overlap between index nodes occurs. With this problem in mind, we propose a method that supports efficient processing of G*k*NN queries on the broadcast stream. The main contributions of our study are summarized as follows:

- We investigate, for the first time to the best of our knowledge, the problem of processing a generalized version of *k*NN queries, namely, G*k*NN queries, which involve both spatial and non-spatial specifications for data objects, in a wireless broadcasting system.
- We present a novel index structure called the *bit-vector R-tree* (*bR-tree*). The bR-tree is a two-dimensional index structure constructed on the basis of the spatial attributes of data objects; it stores additional bit-vector information to describe the non-spatial attributes. Furthermore, each node in the bR-tree stores only one pointer to its children in order to reduce the size of the bR-tree. This is achieved by placing the children of each node contiguously in the broadcast stream and by let-

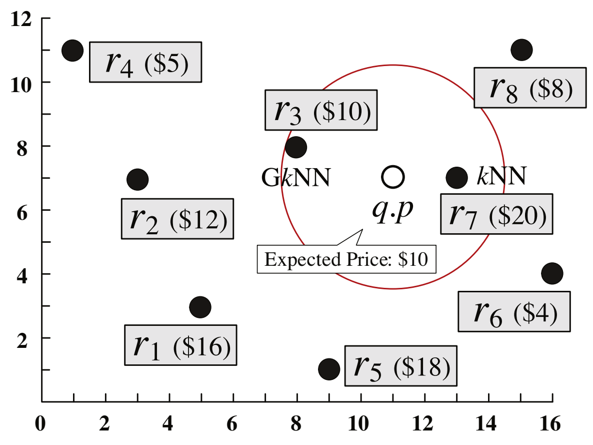


Fig. 1. Dataset of the local restaurants.

ting each node have the link to the first child. The remaining children can be accessed through *implicit pointers*. We generate the broadcast stream by multiplexing the bR-tree and the data objects in the broadcasting channel.
  – We choose to enhance the R-tree [14] because it is recognized as one of the most useful multi-dimensional index structures in academic research and commercial database management systems (DBMSs) such as Oracle [20,21].
- We propose a search algorithm for our index structure. To prune the search space efficiently, we introduce two distance metrics, *gmindist* and *gmaxdist*, defined between a G*k*NN query and a non-leaf node of the bR-tree.
- By conducting comprehensive simulation experiments, we verify the efficiency of the proposed method with regard to energy consumption, latency, and memory requirement. In addition, we test the practicality of the proposed method in a real prototype system for the wireless information delivery [10].

The remainder of this paper is organized as follows. In Section 2, related studies are reviewed, and in Section 3, the problem is formally defined. In Section 4, details of the proposed method are provided, and in Section 5, the performance evaluation results are presented. Finally, Section 6 concludes the paper.

2. Related work

2.1. Wireless broadcasting and air indexing

In mobile computing environments, wireless broadcasting has been widely adopted for delivering information to a large number of users [1,7–10,13,16,18,19,23,25,26,37,38,40–42]. In a wireless broadcasting system, the server periodically broadcasts data objects through the wireless broadcasting channel in a pre-scheduled sequential order. When each client receives a query from a user, it tunes into the channel and processes the query on the broadcast stream. Two performance metrics are commonly employed in a wireless broadcasting system: *access time* and *tuning time* [18]. The former is the duration from the moment a client receives a query from a user to the moment the query is satisfied. The latter is the duration for which the client remains in full operational mode, called *active mode*; it is proportional to the energy consumed by the client. In other words, access time reflects the user-perceived latency, whereas tuning time is a measure of the energy consumed by the client.

Air indexing strategies [7–10,13,16,18,19,23,25,37,38,40–42] are commonly adopted for reducing the tuning time at the expense of increasing access time. The basic idea is to use an index for the appearance time of each data object in the broadcast stream so that a client is able to stay in the energy-saving mode, referred to as *doze mode*, until the required data objects appear in the broadcast stream. The canonical data access protocol over the broadcasting channel is defined as follows:

- Initial probe: The client tunes into the broadcasting channel to determine when the index appears in the broadcast stream. Then, it switches into doze mode and waits for the index. This duration is called *probe wait*.
- Index search and data access: The client tunes into the channel again when the index appears in the broadcast stream. While traversing the index, the client determines when the data objects qualifying the given query appear in the broadcast stream, and it selectively accesses them by switching between active mode and doze mode. This duration is referred to as *bcast wait*.

Access time is the sum of probe wait and bcast wait, and air indexing strategies strive to reduce tuning time while minimizing the increase in access time. Imielinski et al. [18] introduced two tree-based air indexing strategies to reduce access time: $(1,m)$ indexing and *distributed indexing*. In $(1,m)$ indexing, the entire index is replicated m times and included in every $\frac{1}{m}$ fraction of the broadcast stream in order to reduce probe wait. However, repetition of the entire index significantly lengthens the broadcast stream, resulting in a large bcast wait. Distributed indexing overcomes this problem by partially replicating the index. It has been shown that distributed indexing significantly outperforms $(1,m)$ indexing in terms of access time, while maintaining a similar performance in terms of tuning time [18]. Therefore, we adopt distributed indexing for replicating the bR-tree. Note that access time overhead is also incurred because of the size of the index; hence, it is critical to keep the index size small.

The *bucket* is the basic unit of wireless broadcasting, and the broadcast stream consists of *index buckets* and *data buckets*. Index buckets hold index information, whereas data buckets hold data. We assume that each bucket has the same capacity, irrespective of its type (i.e., index bucket or data bucket). In order to make all buckets self-identifying, each bucket contains the following header information: bucket id, bucket type, upcoming appearance time of the next index bucket in the broadcast stream, and start time of the next broadcast stream. Note that the bucket id is determined by the offset of the corresponding bucket from the start time of the current broadcast stream. In the remainder of this paper, we use the number of buckets to measure access time and tuning time. The number of buckets can be translated into time values without loss of generality because all buckets are assumed to have the same capacity and the bandwidth of the broadcasting channel is fixed.

2.2. Location-based query processing

Following Guttman's research [14], the R-tree has been extensively used to access multi-dimensional data (e.g., spatial data), and many R-tree variant index structures have been proposed [2,11,17,22,24,29,32,33,35]. Fig. 2 shows the R-tree that

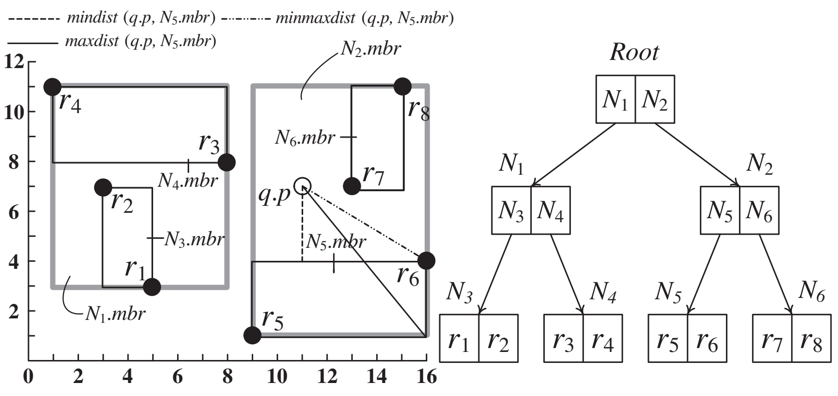


Fig. 2. An example of the R-tree that indexes the dataset in shown Fig. 1.

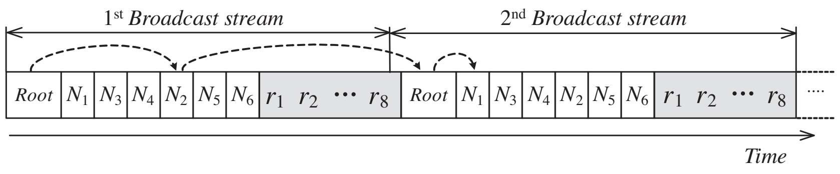


Fig. 3. Broadcast R-tree traversal.

indexes the dataset in Fig. 1, assuming a capacity of 2 entries per node. According to their spatial closeness, restaurants $r_1, \ldots, r_8$ are clustered into leaf nodes $N_3, \ldots, N_6$, each of which is represented as a *minimum bounding rectangle* (*mbr*). Then, the leaf nodes are recursively grouped into non-leaf nodes N_1 and N_2, which become the entries of the root. The R-tree naturally suits a range query. Starting from the root, the query is processed by visiting the nodes whose mbrs intersect with the given query region.

Several search algorithms have been proposed for processing a *k*NN query on the R-tree [6,15,31]. Most of them follow a branch-and-bound approach by utilizing the following distance metrics to prune the search space: *mindist*, *minmaxdist*, and *maxdist*. Given a query point *q.p* and a non-leaf node *N*, *mindist*(*q.p*,*N.mbr*) indicates the minimum possible distance between *q.p* and any data object inside *N.mbr*. On the other hand, *minmaxdist*(*q.p*,*N.mbr*) guarantees that there is at least one data object inside *N.mbr*, whose distance from *q.p* does not exceed this distance. Finally, *maxdist*(*q.p*,*N.mbr*) indicates the maximum possible distance between *q.p* and any data object inside *N.mbr*. An example of these distances is shown in Fig. 2.

The *best-first* algorithm [15,30] is the most efficient algorithm for processing *k*-NN queries; it dynamically determines the traversal order of the R-tree. This is achieved by maintaining a priority queue of candidate entries, sorted in the ascending order of their *mindist*. However, the performance of the best-first algorithm may deteriorate significantly in terms of access time in a wireless broadcasting system. The broadcast stream periodically flows in a pre-scheduled sequential order, and each R-tree node is only available when it appears in the broadcast stream. Whenever the dynamic traversal order of the best-first algorithm differs from the appearance order of the R-tree nodes in the broadcast stream, access time is significantly increased.

For example, suppose that the R-tree appears only once in each version of the broadcast stream (i.e., (1,1) indexing is applied), as shown in Fig. 3. When the best-first algorithm tries to visit N_1 after visiting N_2, it has to wait for the next broadcast stream because N_1 has already been broadcast. By taking the sequential property of the broadcast stream into account, the *appearance-first* algorithm [13,19,25,40] has been considered for improving the access time performance. The appearance-first algorithm sequentially visits the R-tree nodes according to their order of appearance in the broadcast stream, while filtering out the unqualified nodes according to the *mindist*-, *minmaxdist*-, and/or *maxdist*-based heuristics.

3. Problem definition and motivation

In this paper, we address the problem of processing G*k*NN queries in a wireless broadcasting system. Let $\mathcal{D} = \{d_1, d_2, \ldots, d_{|\mathcal{D}|}\}$ be a set of data objects to be broadcast, each of which is associated with spatial attributes $p = (x,y)$ and a set of non-spatial attributes $a = \{a_1, a_2, \ldots, a_n\}$. A data object $d \in \mathcal{D}$ is represented as $(d.p, d.a)$, where $d.p = (d.x, d.y)$ denotes the values of its spatial attributes and $d.a = \{d.a_1, d.a_2, \ldots, d.a_n\}$ denotes the values of its non-spatial attributes. A G*k*NN query *q*, issued by a mobile user over $\mathcal{D}$, is represented as $(q.p, q.v, k)$. Here, $q.p = (q.x, q.y)$ denotes the user's location, $q.v = \{q.v_1, q.v_2, \ldots, q.v_m\}$ denotes a set of the target values that the user specifies on a subset of non-spatial attributes $\{\acute{a}_1, \acute{a}_2, \ldots, \acute{a}_m\} \subseteq a$, and *k* is an integer value indicating the desired number of data objects.

The G*k*NN query *q* aims to retrieve the *k* closest data objects in $\mathcal{D}$, where the degree of closeness between each data object *d* and *q* is determined according to the *generalized distance*, which is the sum of two individual distances: spatial distance to the user's location and non-spatial distance to the user-specified target values. Specifically, given a G*k*NN query *q* and a data object *d*, the generalized distance *gdist* between *q* and *d* is defined as

$$gdist(q,d) = dist(q.p,d.p) + dist(q.v,d.a), \tag{1}$$

where $dist(q.p,d.p) = \sqrt{(q.x - d.x)^2 + (q.y - d.y)^2}$ is the spatial distance and $dist(q.v,d.a) = dist(q.v_1, d.\acute{a}_1) + dist(q.v_2, d.\acute{a}_2) + \cdots + dist(q.v_m, d.\acute{a}_m)$ is the non-spatial distance from *q* to *d*. We assume that all the non-spatial attributes are numerical (either continuous or ordered) and have the same scale; therefore, $dist(q.v,d.a)$ is simply calculated as $|q.v_1 - d.\acute{a}_1| + |q.v_2 - d.\acute{a}_2| + \cdots + |q.v_m - d.\acute{a}_m|$. Although we assign equal weights to $dist\ (q.p,d.p)$ and $dist(q.v,d.a)$ in our definition of $gdist(q,d)$, they might be associated with different weights. For example, $gdist(q,d) = \alpha \cdot dist(q.p,d.p) + (1 - \alpha) \cdot dist(q.v,d.a)$, $d) = \alpha \cdot dist(q.p,d.p) + (1 - \alpha) \cdot dist(q.v,d.a)$, where $\alpha \in [0,1]$, is the weighting parameter indicating the relative importance between $dist(q.p,d.p)$ and $dist(q.v,d.a)$.

Definition 1. G*k*NN query: A G*k*NN query *q* issued over $\mathcal{D}$ returns the subset $\text{G}k\text{NN}(q) \subseteq \mathcal{D}$, which contains *k* data objects from $\mathcal{D}$, and for which the condition

$$\forall d \in GkNN(q), \forall \acute{d} \in \mathcal{D} - GkNN(q) : gdist(q,d) < gdist(q,\acute{d}) \tag{2}$$

holds, where ties for the *k*th data object $d \in \text{G}k\text{NN}\ (q)$ are broken arbitrarily.

An intuitive approach for supporting G*k*NN queries in a wireless broadcasting system is to construct a high-dimensional index structure such as the high-dimensional R-tree, and to include it in the broadcast stream so that a client performs the search process by selectively scanning the broadcast stream. However, such a method suffers from considerable performance degradation because the broadcast R-Tree typically fails to handle a dataset with high dimensionality.

This is mainly due to the fact that a dimensionality increase results in a much smaller fan-out of the broadcast R-tree than that of the disk-resident R-tree because the capacity of a bucket is much smaller than that of a disk block. This causes height growth of the broadcast R-tree and excessive overlap among non-leaf nodes (i.e., mbrs) in the broadcast R-tree. The height growth increases the tuning time cost of the search process, which must descend to the leaf nodes to satisfy G*k*NN queries, and excessive overlap among non-leaf nodes results in several unnecessary search paths that do not lead to the final results of the G*k*NN queries.[2] Note that because the size of the broadcast R-tree becomes fairly large, the access time cost also increases. In addition, although the dataset is associated with a large number of non-spatial attributes, G*k*NN queries may specify target values on an arbitrary subset of these attributes. Unfortunately, previous studies have shown that a single high-dimensional R-tree built on the basis of all the spatial and non-spatial attributes is not effective in handling such queries because all spatial and non-spatial attribute values are coarsely approximated into the hyper-mbrs of non-leaf nodes [4,28,34,36,39].

In the next section, in order to remedy the problems stated above, we propose a method that supports efficient processing of G*k*NN queries in a wireless broadcasting system.

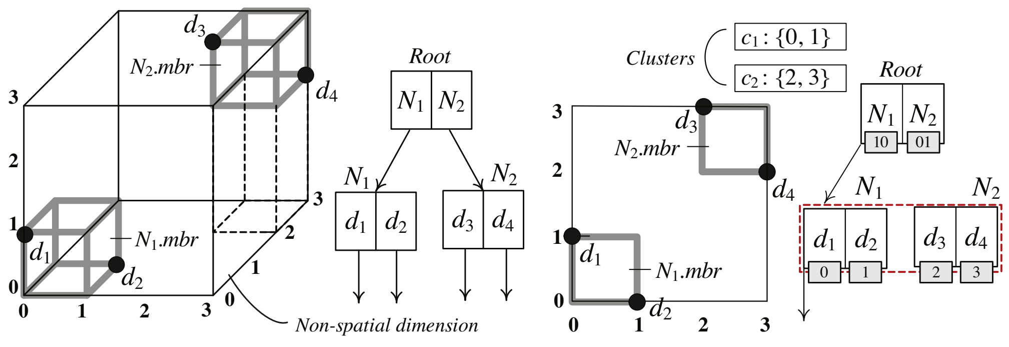
4. The proposed method

In this section, we propose a novel index structure, namely, the bR-tree, and the corresponding search algorithm for processing G*k*NN queries on the broadcast stream. In the following subsections, we make some assumptions to facilitate the problem statement. First, each data object $d \in \mathcal{D}$ is associated with only one non-spatial attribute *a*. Consequently, a G*k*NN query *q* is assumed to specify only one non-spatial target value *v* on *a*, and thus, $dist(q.v,d.a)$ is calculated as $|q.v - d.a|$. However, our method can be naturally extended for the case of multiple non-spatial attributes, as described in Section 5. In addition, we assume that $dist(q.p,d.p)$ and $dist(q.v,d.a)$ are normalized to be in the range $[0,1]$.

4.1. The Bit-vector R-tree ($bR - tree$)

The bR-tree is a two-dimensional R-tree constructed on the basis of spatial attributes, where additional bit-vector information, called the *node bit-vector*, is associated with each non-leaf node *N* to describe the non-spatial attribute values of the data objects covered by the subtree rooted at *N*. In addition, each non-leaf or leaf node in the bR-tree stores only one pointer to its children (bR-tree nodes or data objects). This is achieved by contiguously placing the children of each node in the broadcast stream and by letting each node contain the address (i.e., appearance time in the broadcast stream) of the first child. The other children can be accessed by adding an offset to that address. More formally, each leaf node of the bR-tree stores a single pointer to its children (i.e., data objects) and an array of child entries of the form $\langle d.p,d.a\rangle$. On the other hand, each non-leaf node of the bR-tree stores a single pointer to its children (i.e., bR-tree nodes) and an array of child entries of the form $\langle N.mbr,N.bv\rangle$, where *N.mbr* is defined as in the case of the conventional R-Tree, while *N.bv* is the node bit-vector of each child node *N*.

[2] The amount of overlap is known to be rather significant when the number of dimensions exceeds 4 [3].



(a) The R-tree (b) The bR-tree

Fig. 4. Difference between the R-tree and the bR-tree.

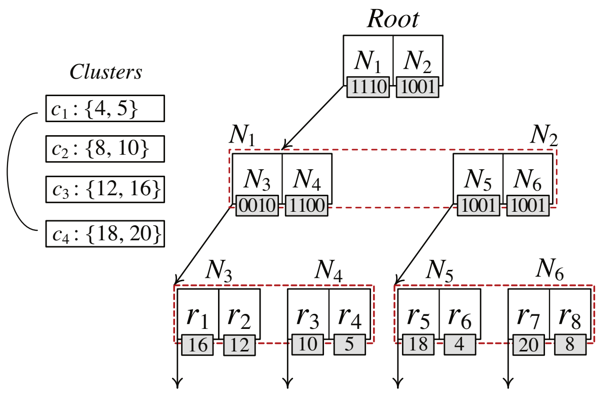


Fig. 5. An example of the bR-tree that indexes the dataset shown in Fig. 1.

Definition 2 (*Node bit-vector*). Suppose that a non-spatial attribute value of each data object $d \in \mathcal{D}$ is partitioned into $\mathcal{K}$ clusters.[3] We denote a set of data objects covered by the subtree rooted at N as Sub_N. Then, the node bit-vector $N.bv$, associated with a non-leaf node N, is a $\mathcal{K}$-bit vector with one bit for each cluster $c_i(1 \leqslant i \leqslant \mathcal{K})$, and each bit position $b_i(1 \leqslant i \leqslant \mathcal{K})$ in $N.bv$ is set as follows:

$$b_i = \begin{cases} 1, & \text{iff } \exists d \in Sub_N : d.a \in c_i(1 \leqslant i \leqslant \mathcal{K}), \\ 0, & \text{otherwise.} \end{cases}$$

Assuming a capacity of two entries per node, the difference between the bR-tree and the conventional R-tree is shown in Fig. 4, and an example of the bR-tree that indexes the dataset in Fig. 1 is shown in Fig. 5. In Fig. 4(b), the number of bits in a node bit-vector of the bR-tree is assumed to be 2 (i.e., $\mathcal{K} = 2$). On the other hand, in Fig. 5, the number of bits in a node bit-vector is assumed to be 4 (i.e., $\mathcal{K} = 4$). Note that we omit the planar representation of the bR-tree in Fig. 5 because it is exactly the same as that of the R-tree shown in Fig. 2.

In Fig. 5, the subtree rooted at N_4 covers the restaurants r_3 and r_4, whose average meal prices, \$10 and \$5, are in the clusters c_1 and c_2, respectively; thus, the node bit-vector $N_4.bv$ associated with N_4 is "1100". An important observation here is that each cluster can indicate a disjoint *cluster interval* whose boundary consists of the minimum and maximum prices in the cluster (e.g., c_1 and c_2 can indicate [4,5,8,10], respectively). Accordingly, $N_4.bv$ can indicate that the price of each restaurant $r \in Sub_{N_4}$ lies within one of the cluster intervals [4,5,8,10].

Observation 1. Given a non-leaf node N of the bR-tree, let $C(|C| \leqslant \mathcal{K})$ be the set of clusters whose corresponding bit positions in $N.bv$ are set to '1'. We denote the minimum and the maximum values in each cluster $c \in C$ as c^- and c^+, respectively. Then, $N.bv$ can indicate that $\forall d \in Sub_N, \exists c \in C : c^- \leqslant d.a \leqslant c^+$.

The use of a node bit-vector in the bR-tree is beneficial because each cluster interval can be represented by only a single bit, resulting in a larger fan-out per non-leaf node.[4] Because each bR-tree node needs to store just one pointer to refer to the

[3] We used the $\mathcal{K}$-*means* algorithm to generate clusters because of its simplicity; however, other sophisticated clustering algorithms can also be employed.
[4] In the case of the dataset being associated with multiple non-spatial attributes, multiple node bit-vectors are generated accordingly.

appearance times of its children in the broadcast stream, the bR-tree further improves performance in terms of tuning time and access time. Given a set of data objects $\mathcal{D}$, let f_l and f_{nl} be the fan-out of a leaf node and a non-leaf node, respectively, of the bR-tree or R-tree. The bR-tree and R-tree differ in terms of (i) the size of non-spatial information stored in each non-leaf node and (ii) the size of pointer information stored in each non-leaf or leaf node, resulting in different fan-outs f_l and f_{nl}. For example, when assuming default parameter settings in our experiments, the bR-tree has a fan-out of $f_l = 6$ and $f_{nl} = 6$, whereas the R-tree has a fan-out of $f_l = 5$ and $f_{nl} = 3$. The larger fan-out of the bR-tree decreases its height and resolves the problem of excessive overlap among non-leaf nodes.

Algorithm 1. Packing algorithm for bR-tree construction

```
Input a set of data objects $\mathcal{D}$ to be broadcast
Output the bR-tree
Procedure
1: Create an empty set $\mathcal{DB}$;
2: for each data object $d \in \mathcal{D}$ do
3:   Allocate a data bucket and fill $d$ into the data bucket;
4:   Insert the data bucket into $\mathcal{DB}$;
5: end for
6: Invoke PACK($\mathcal{DB}$);
End of Procedure
Function PACK($\mathcal{DB}$)
1:   Compute the fan-out $f$;
2:   if $|\mathcal{DB}| \geqslant f$ then
3:     Partition $\mathcal{DB}$ into $\lceil \frac{|\mathcal{DB}|}{f} \rceil$ groups using STR method;
4:     Create an empty set $\mathcal{IB}$;
5:     for each group $g$ do
6:       Allocate an index bucket;
7:       Fill the pointer to $g$ into the bucket;
8:         for each element $e$ in $g$ do
9:           Create the entry and fill the entry into the bucket;
10:        end for
11:      Insert the bucket into $\mathcal{IB}$;
12:    end for
13:  Invoke PACK($\mathcal{IB}$);
14: else // $|\mathcal{DB}| < f$
15:  Allocate a root index bucket;
16:  Fill the pointer to $\mathcal{DB}$ into the bucket;
17:  for each element in $\mathcal{DB}$
18:    Create the entry and fill the entry into the bucket;
19:  end for
20: end if
End of Function
```

The bR-tree is constructed solely on the basis of the spatial attributes of data objects using the packing algorithm because all the data objects to be broadcast are available a priori. Note that because the basic unit of wireless broadcasting is the bucket, as mentioned in SubSection 2.1, an index bucket corresponds to a bR-tree node. In addition, a data bucket is assumed to accommodate only one data object, and thus, each data bucket corresponds to a single data object. The packing algorithm constructs the bR-tree by creating one level at a time from the bottom to the root.

Algorithm 1 is the packing algorithm of bR-tree construction. Initially, a set of data buckets $\mathcal{DB}$ is created, where the data objects to be broadcast will be filled (Lines 1–5). Then, the algorithm invokes a function PACK, which takes $\mathcal{DB}$ as an input (Line 6). After computing the fan-out f,[5] PACK partitions $\mathcal{DB}$ into $\lceil \frac{|\mathcal{DB}|}{f} \rceil$ distinct groups, where the value of $\lceil \frac{|\mathcal{DB}|}{f} \rceil$ is the number of index buckets to be allocated (Lines 1–3 of PACK). This partitioning is accomplished by the *sort tile recursive* (*STR*) method [24]. The entries for elements in each group are created and filled into the same index bucket, where the single pointer to each group is also filled (Lines 5–10 of PACK). Then, PACK, which takes a set of index buckets $\mathcal{IB}$ as an input, is invoked (Line 11 of PACK). In this way, PACK recursively creates a new level of the bR-tree until the root is created, where the operations for handling the node bit-vector information are also added.

[5] The fan-out f can be computed by $\left\lfloor \frac{\text{bucket capacity} - \text{size ofheader information}}{\text{size of an entry}} \right\rfloor$.

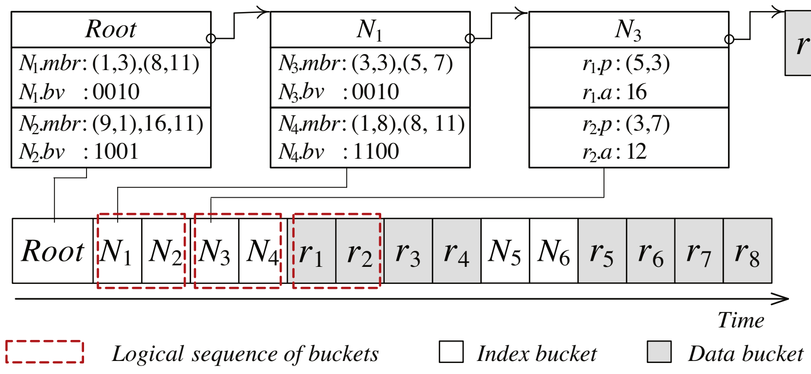


Fig. 6. Broadcast stream generated from the bR-tree.

Lemma 1 analyzes the time complexity of the function Pack. For the simplicity of analysis, we assume a fully packed bR-tree (i.e., all the bR-tree nodes have the maximum number of children), and thus, $|\mathcal{DB}| = f_l \times (f_{nl})^{h-1}$, where f_l, f_{nl}, and h denote the fan-out of a leaf node, fan-out of a non-leaf node, and height of the resulting bR-tree, respectively.

Lemma 1. *The time complexity of the function* Pack *is O (nlogn), where* $n = |\mathcal{DB}|$.

Proof. The function Pack consists of two steps. First, Pack partitions the input, i.e., $\mathcal{DB}$ or $\mathcal{IB}$, into $\frac{n}{f_l\cdot(f_{nl})^i}$ groups using the STR method, where $0 \leqslant i \leqslant h-1$. This requires the input of each invocation of Pack to be sorted once along the x axis and once along the y axis (see [24] for further details), and thus, it takes $\mathrm{O}(n\log n + \frac{n}{f_l\cdot(f_{nl})^0}\log\frac{n}{f_l\cdot(f_{nl})^0} + \frac{n}{f_l\cdot(f_{nl})^1}\log\frac{n}{f_l\cdot(f_{nl})^1} + \cdots + \frac{n}{f_l\cdot(f_{nl})^{h-1}}\log\frac{n}{f_l\cdot(f_{nl})^{h-1}}) = \mathrm{O}(n\log n)$. Second, each invocation of pack creates entries for elements in each group and fills them into the index bucket. This incurs $\mathrm{O}\left(f_l\cdot\frac{n}{f_l\cdot(f_{nl})^0} + f_{nl}\cdot\frac{n}{f_l\cdot(f_{nl})^1} + \cdots + f_{nl}\cdot\frac{n}{f_l\cdot(f_{nl})^{h-1}}\right) = \mathrm{O}(n)$. Therefore, the overall time complexity of Pack is O ($n$log$n$). □

After the bR-tree is constructed, it is sequentially arranged and interleaved with data buckets in the broadcast stream by traversing the bR-tree in a *depth-first* order. Fig. 6 shows the broadcast stream generated from the bR-tree in Fig. 5. It is important to note that when traversing the bR-tree, the children of each bR-tree node N are logically considered to be a single sequence of buckets and placed contiguously in the broadcast stream because N stores only one explicit pointer to its children. The pointer information stored in N is translated into the appearance time of the first bucket in the sequence (i.e., the first child) accordingly. The remaining buckets in the sequence (i.e., the remaining children) can be accessed through implicit pointers.

Definition 3 (*Implicit pointer*). Given a bR-tree node N, the implicit pointer ptr_i of N's ith child in the broadcast stream is calculated as

$$ptr_i = ptr_1 + b \times (i-1), \tag{3}$$

where ptr_1 and b are the appearance time of the first child and the time necessary to broadcast a single bucket, respectively.

With respect to index replication, unlike distributed indexing [18], we only need to replicate the root with some control information because the search process of most location-based queries (including GkNN queries) should start from the root. In addition, for optimality, the root is replicated as many times as the number of its children [7,18].

4.2. Search algorithm for the broadcast bR-tree

The search algorithm for the broadcast bR-tree sequentially visits the bR-tree nodes in the order of their appearance in the broadcast stream, while pruning the branches that are guaranteed to fail. Before presenting the detailed search algorithm, we introduce two distance metrics, *gmindist* and *gmaxdist*, which are defined between a GkNN query q and a non-leaf node N of the bR-tree.

Definition 4 (*gmindist*). Given a query q and a non-leaf node N of the bR-tree, let $c^-_{\min} = \min_{\forall c\in C}(|q.v - c^-|)$ and $c^+_{\min} = \min_{\forall c\in C}(|q.v - c^+|)$. Then, the *gmindist* defined between q and N is

$$gmindist(q,E) = mindist(q.p, N.mbr) + mindist(q.v, N.bv), \tag{4}$$

where

$$mindist(q.v, N.bv) = \begin{cases} 0, & \text{if } \exists c \in C : c^- \leqslant q.v \leqslant c^+; \\ \min\left(c^-_{\min}, c^+_{\min}\right), & \text{otherwise}. \end{cases}$$

Lemma 2. *Given a query q and a non-leaf node N of the bR-tree,* $\forall d \in Sub_N$, *mindist(q.p,N.mbr)* $\leqslant$ *dist(q.p,d.p).*

Proof. See Theorem 1 in [31]. □

Lemma 3. *Given a query q and a non-leaf node N of the bR-tree,* $\forall d \in Sub_N$, *mindist(q.v,N.bv)* $\leqslant$ *dist(q.v,d.a).*

Proof. We prove this lemma by contradiction. Assume that there exists some data object $\acute{d} \in Sub_N$ such that $dist(q.v, \acute{d}.a) < mindist(q.v, N.bv)$. We distinguish two cases:

1. If $\exists c \in C : c^- \leqslant q.v \leqslant c^+$, *mindist(q.v,N.bv)* = 0, which contradicts the above assumption.
2. If $\nexists c \in C : c^- \leqslant q.v \leqslant c^+, mindist(q.v, N.bv) = \min\left(c^-_{\min}, c^+_{\min}\right)$. Then, the following inequality holds:

$$|q.v - \acute{d}.a| < \min(c^-_{\min}, c^+_{\min}) \tag{5}$$

By Observation 1, for some $\acute{c}(\in C), \min(|q.v - \acute{c}^-|, |q.v - \acute{c}^+|) \leqslant |q.v - \acute{d}.a|$. On the other hand, we know from Definition 4 that $\min\left(c^-_{\min}, c^+_{\min}\right) \leqslant \min\left(|q.v - \acute{c}^-|, |q.v - \acute{c}^+|\right)$. This generates a contradiction to inequality (5). Therefore, $\acute{d}$ cannot exist. □

Corollary 1. *Given a query q and a non-leaf node N of the bR-tree,* $\forall d \in Sub_N$, *gmindist(q,N)* $\leqslant$ *gdist(q,d).*

Proof. Trivially proved by Lemmas 2 and 3. □

Definition 5 (*gmaxdist*). Given a query q and a non-leaf node N of the bR-tree, let $c^-_{\max} = \max_{\forall c \in C}(|q.v - c^-|)$ and $c^+_{\max} = \max_{\forall c \in C}(|q.v - c^+|)$. Then, the *gmaxdist* defined between q and N is

$$gmaxdist(q,N) = minmaxdist(q.p, N.mbr) + maxdist(q.v, N.bv), \tag{6}$$

where $maxdist(q.v, N.bv) = \max\left(c^-_{\max}, c^+_{\max}\right)$.

Lemma 4. *Given a query q and a non-leaf node N of the bR-tree,* $\exists d \in Sub_N$: *dist(q.p,d.p)* $\leqslant$ *minmaxdist(q.p,N.mbr).*

Proof. See Theorem 2 in [31]. □

Lemma 5. *Given a query q and a non-leaf entry N of the bR-tree,* $\forall d \in Sub_N$, *dist(q.v,d.a)* $\leqslant$ *maxdist(q.v,N.bv).*

Proof. Assume to the contrary that there exists some data object $\acute{d} \in Sub_N$ such that $maxdist(q.v, N.bv) < dist(q.v, \acute{d}.a)$. Then, the following inequality holds:

$$\max\left(c^-_{\max}, c^+_{\max}\right) < |q.v - \acute{d}.a| \tag{7}$$

By Observation 1, for some $\acute{c}(\in C), |q.v - \acute{d}.a| \leqslant \max(|q.v - \acute{c}^-|, |q.v - \acute{c}^+|)$. This produces a contradiction to inequality (7) because we know from Definition 5 that $\max(|q.v - \acute{c}^-|, |q.v - \acute{c}^+|) \leqslant \max\left(c^-_{\max}, c^+_{\max}\right)$. Hence, $\acute{d}$ cannot exist. □

Corollary 2. *Given a query q and a non-leaf node N of the bR-tree,* $\exists d \in Sub_N$: *gdist (q,d)* $\leqslant$ *gmaxdist (q,N).*

Proof. Trivially proved by Lemmas 4 and 5. □

Algorithm 2. Search algorithm for the broadcast bR-tree

```
Input a GkNN query $q$
Output the result of $q$
Procedure
1: Search_list = $\emptyset$;//sorted by increasing order of appearance
2: Distance_list = $\emptyset$;// sorted by increasing order of gmaxdist (or gdist), size = $k$
3: Perform initial probe;
4: Access Root and insert all the entries (stored in Root) into Search_list;
5: do{
6:   Remove the next entry $E$ from Search_list;
7:   if the gmindist (or gdist) value of the corresponding node (or data object) is greater than candidate_dist;
8:     continue;
9:   else
10:      if $E$ is a non-leaf node entry
11:        Access the corresponding node;
12:        for each child entry $E_{child}$
13:          if candidate_dist < gmindist(q,child)
14:            continue;
15:          else
16:            Insert $E_{child}$ into Search_list;
17:            Update Distance_list if necessary;
18:      else if $E$ is a leaf node entry
19:        Access the corresponding node;
20:        for each child entry $E_{child}$
21:          if candidate_score < gdist(q,child)
22:            continue;
23:          else
24:            Insert $E_{child}$ into Search_list;
25:            Update Distance_list if necessary;
26:      else // $E$ is a data object entry
27:        Access the corresponding data object $d$;
28:        Include $d$ into the candidate result;
29: }while (Search_list is empty)
30: Find out the final result among the data objects in the candidate result;
31: Return the final result;
End of Procedure
```

The aforementioned definitions, lemmas, and corollaries establish the foundation for pruning the search space. Specifically, given a G*k*NN query *q* and a non-leaf node *N*, if there currently exist at least *k* data objects whose *gdist* values are smaller than the value of $gmindist(q,N)$,*N* does not have to be considered because Sub_N cannot contain any better data object (by Corollary 1). On the other hand, the value of $gmaxdist(q,N)$ can enable us to conservatively estimate the *gdist* value of the candidate data object among Sub_N because the value of $gmaxdist(q,N)$ ensures that Sub_N contains at least one data object whose *gdist* value is smaller than or equal to it (by Corollary 2). The details of the search algorithm are provided in Algorithm 2, where the following two data structures are used:

- *Search_list*: *Search_list* stores entries for the bR-tree nodes (or the data objects) that should be accessed. The entries stored in *Search_list* are sorted according to the appearance order of their corresponding nodes (or data objects) in the broadcast stream.
- *Distance_list*: *Distance_list* stores *gmaxdist* (or *gdist*) values of the bR-tree nodes (or data objects), and the values stored in *Distance_list* are sorted in ascending order. In addition, *Distance_list* can store at most *k* values. Consequently, when a new value is inserted into *Distance_list*, the current *k*th value is removed from *Distance_list* in case it has already stored *k* values.

The search algorithm performs an initial probe, i.e., it tunes into the broadcast channel to find out when the next root of the bR-tree appears in the broadcast stream (Line 3). After the initial probe, the algorithm accesses the root and inserts all its entries into *Search_list* (Line 4). Then, the algorithm iteratively examines the entries stored in *Search_list* until it becomes empty. Let *candidate_dist* be the *k*th value stored in *Distance_list*. In case there are fewer than *k* values in *Distance_list*, *candidate_dist* = ∞. At each iteration, the algorithm removes the top entry from *Search_list* and checks if the *gmindist* or *gdist* value of the corresponding node or data object is greater than *candidate_dist* (Lines 6–7). If this is the case, the algorithm safely proceeds with the next iteration without accessing the corresponding node or data object. Otherwise, the algorithm performs the following:

- If the entry is a non-leaf node entry: The algorithm accesses the corresponding node N (Line 11). Then, for each child entry stored in N, the algorithm checks if $candidate\_dist < gmindist(q, child)$, where *child* denotes N's child (Lines 12–13). If so, the entry is skipped. Otherwise, it is inserted into *Search_list*, and the value of $gmaxdist(q, child)$ is inserted into *Distance_list* in case it is smaller than *candidate_dist* (Lines 16–17).
- If the entry is a leaf node entry: The algorithm accesses the corresponding node N (Line 19). Then, for each child entry stored in N, the algorithm checks if $candidate\_dist < gdist(q, child)$ (Lines 20–21). If this is the case, the entry is skipped. Otherwise, it is inserted into *Search_list*, and the value of $gdist(q, child)$ is inserted into *Distance_list* if it is smaller than *candidate_dist* (Lines 24–25).
- If the entry is a data object entry: The algorithm accesses the corresponding data object d and includes d in the candidate result (Lines 27–28).

Then, the algorithm proceeds with the next iteration. When *Search_list* becomes empty, the algorithm finds out the final result among the data objects in the candidate result (Lines 30–31). Now, we study the time complexity of the search algorithm for the broadcast bR-tree. Assuming a fully packed bR-tree, Lemma 6 analyzes the time complexity of traversing a subtree rooted at a bR-tree node N.

Lemma 6. *Given a bR-tree node N, the time complexity of traversing a subtree rooted at N is O (m), where m is the number of data objects covered by the subtree rooted at N, i.e.,* $m = |Sub_N|$.

Proof. The time complexity of traversing a subtree rooted at N is determined by two key operations of *Search_list* and *Distance_list*: insert and remove operations. We consider two cases:

1. If N is a non-leaf node at depth i of the bR-tree (assuming the root has depth 0): The number of descendant nodes under N is $(f_{nl})^1 + (f_{nl})^2 + \cdots + (f_{nl})^{(h-1)-i}$, where f_{nl} and h denote the fan-out of a non-leaf node and the height of the bR-tree, respectively, and the number of data objects covered by the subtree rooted at N is $\frac{\mathcal{D}}{(f_{nl})^i} = m$. Then, the time taken by the insert and remove operations of *Search_list* and *Distance_list* are at most $O((f_{nl})^1 + (f_{nl})^2 + \cdots + (f_{nl})^{(h-1)-i} + m) = O(m)$ and constant, respectively, because both lists are sorted.[6]
2. If N is a leaf node: There are no descendant nodes under N, and the number of data objects covered by the subtree rooted at N is $f_l = m$. Then, the time taken by the insert and remove operations of *Search_list* and *Distance_list* are at most $O(m)$ and constant, respectively.

Therefore, the overall time complexity of traversing a subtree rooted at N is $O(m)$. □

5. Performance evaluation

In this section, we compare the performance of the proposed method with that of a high-dimensional R-tree-based naïve method for processing G*k*NN queries on the broadcast stream. Throughout this section, we label the proposed method and naïve method as the bR-tree method and R-tree method, respectively.

5.1. Experimental setup

All the simulation experiments were conducted on a Pentium IV 3.2 GHz machine with 2 GB RAM. We used a system model similar to that described in [41]. The system model, which consists of a server, mobile clients, and a broadcast channel, was implemented using Java. We generated 5 sets of data objects (See Table 1), where the locations of the data objects are uniformly distributed in a square Euclidean space. In addition, the non-spatial attributes associated with the data objects in the dataset are scaled to [0, 1], and they follow a *Zipf* distribution with skew parameter $z = 0.8$.

In each experiment, we generated 1,000 queries and measured the average access time, tuning time, and memory requirement of the bR-tree method and R-tree method in terms of the number of buckets by varying one of the parameters listed in Table 1, where the default values of the parameters are stated in boldface. For measuring the memory requirement of both methods, we used the maximum number of entries stored in *Search_list*, as in [13,25]. Each query randomly specifies a set of target values on an arbitrary subset of the non-spatial attributes.

In constructing the broadcast stream, we employed the distributed indexing strategy. A data object was assumed to fit into one data bucket, irrespective of the bucket capacity. We set the size of the header information of a (data or index) bucket to 8 bytes. For a non-leaf node of the (high dimensional) R-tree, we allocated 2 bytes to the pointer per child, 32 bytes to the

[6] Without loss of generality, we assume that $(f_{nl})^1 + (f_{nl})^2 + \cdots + (f_{nl})^{(h-1)-i}$ is much smaller than m.

Table 1
Parameter Settings.

Parameter	Setting
# of data objects	10 K, 20 K, **30 K**, 40 K, 50 K
k	1, 8, 16, **32**, 64, 128, 256
α	0, 0.1, 0.3, **0.5**, 0.7, 0.9, 1
Bucket capacity	128, **256**, 512, 1024, 2048 bytes
# of non-spatial attributes	1, 2, **3**, 4, 5

spatial information (i.e., x- and y- coordinates for the lower-left and upper-right points of an mbr), and 16 bytes to the non-spatial information (i.e., values for the non-spatial interval) per non-spatial attribute. For a leaf node of the R-tree, we allocated 2 bytes to the pointer per child, 16 bytes to the spatial information, and 8 bytes to the non-spatial information per non-spatial attribute. On the other hand, for the configuration of the bR-tree, an important parameter that affects the overall performance is the size of a node bit-vector, namely, the value of $\mathcal{K}$. Hence, we generated 1,000 queries and measured the average access time, tuning time, and memory requirement of the bR-tree method by varying the size of the node bit-vector assigned to each non-spatial attribute. We set all the parameters listed in Table 1 to their default values.

Note that *gmindist* and *gmaxdist*, defined for the bR-tree, can be easily extended for considering multiple non-spatial attributes. For example, let us assume that a GkNN query q specifies target values v_1 and v_2 on non-spatial attributes a_1 and a_2. Then, the *gmindist* value of a non-leaf node N of the bR-tree with respect to q is calculated as $mindist(q.p,N.mbr) + min(q.p,N.mbr) + mindist(q.v_1,N.bv_1) + mindist(q.v_2,N.bv_2)$. On the other hand, the *gmindist* value of N with respect to q is calculated as $minmaxdist(q.p,N.mbr) + maxdist(q.v_1,N.bv_1) + maxdist(q.v_2, N.bv_2)$. As shown in Fig. 7(a), the access time gradually increases when the size of the node bit-vector increases, owing to the enlarged size of the bR-tree, which negatively affects the access time performance. Regarding the tuning time performance, the search algorithm for the broadcast bR-tree achieves the best performance when the size of the node bit-vector is set to 3 bytes, as shown in Fig. 7(b). Although a smaller size of the node bit-vector increases the fan-out of the bR-tree node, the search algorithm needs to traverse a greater number of unnecessary search paths owing to the coarser approximation of the actual non-spatial attribute values. On the other hand, a larger size of the node bit-vector guarantees a finer approximation of the non-spatial attribute values, but it decreases the fan-out of the bR-tree. It is also important to note that the memory requirement increases with the tuning time, as shown in Fig. 7(c), because *Search_list* stores the entries for the bR-tree nodes or data objects that should be scanned later.

Because the bR-tree method achieves a good tradeoff between the access time performance and tuning time performance (or memory requirement) when the size of the node bit-vector is set to 3 bytes, we employed this value in the remainder of our experiments. Therefore, for a non-leaf node of the bR-tree, we allocated 2 bytes to the single pointer, 32 bytes to the spatial information, and 3 bytes to the bit-vector information per non-spatial attribute. For a leaf node of the bR-tree, we allocated 2 bytes to the single pointer, 16 bytes to the spatial information, and 8 bytes to the non-spatial information per non-spatial attribute. Because determining the optimal size of a node bit-vector depends on multiple factors such as the distribution of non-spatial attribute values and query patterns of users of diverse interests, we plan to address this issue in a future study.

5.2. Experimental results

In the first experiment, we studied the effect of the number of data objects on the performance of the bR-tree method and R-tree method. As shown in Fig. 8, the access time, tuning time, and memory requirement of both methods increase with the number of data objects. This is due to the fact that as the number of data objects increases, the size of the bR-tree and R-tree

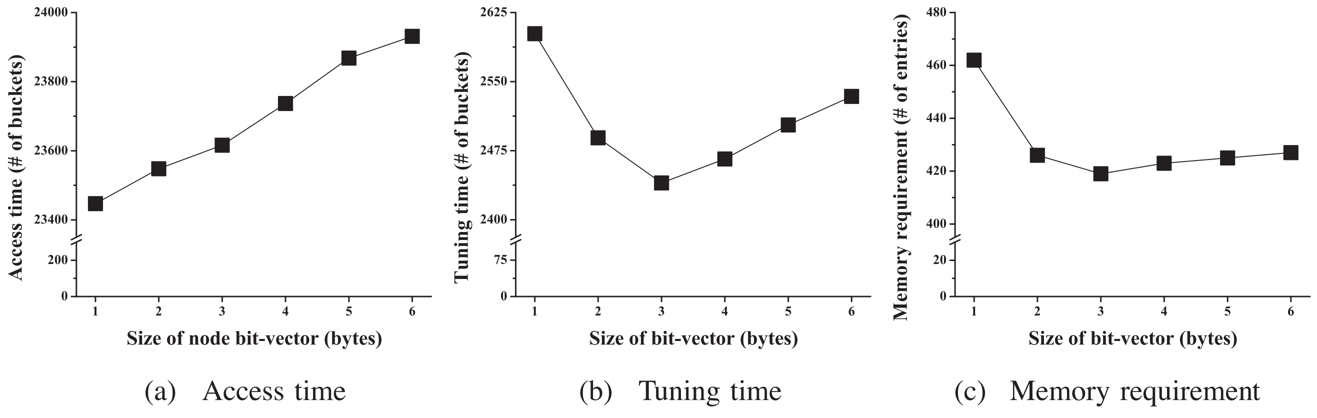


Fig. 7. Access time, tuning time, and memory requirement vs. node bit-vector size.

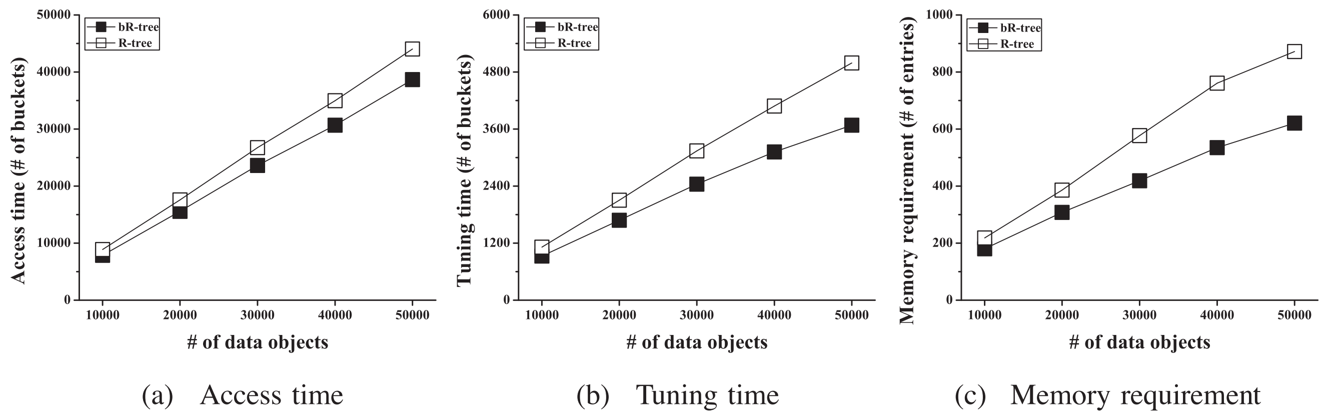


Fig. 8. Access time, tuning time, and memory requirement vs. # of data objects.

also increases, and more nodes of the bR-tree/R-tree have to be accessed. However, the bR-tree method outperforms the R-tree method in terms of access time, tuning time, and memory requirement owing to the larger fan-out of the bR-tree than that of the R-tree. The larger fan-out of the bR-tree decreases its height and resolves the problem of excessive overlap among non-leaf nodes. Therefore, the search algorithm for the bR-tree accesses a smaller number of bR-tree nodes. Thus, the bR-tree method achieves better access time and tuning time performance with less memory requirement than the R-tree method. In addition, because the size of the bR-tree is smaller than that of the R-tree, the bR-tree method further improves the access time performance. As compared to the R-tree method, on average, the bR-tree method requires 87.6% of the access time, 77.6% of the tuning time, and 75.4% of the memory space.

Fig. 9 shows the impact of the value of k on the performance of the bR-tree method and R-tree method. We observe that as the value of k increases, the performance of both methods deteriorates owing to the larger result size. However, the bR-tree method outperforms the R-tree method because of the same reason mentioned in the first experiment. On average, the bR-tree method require 87.7% of the access time, 73.5% of the tuning time, and 71.4% of the memory space, as compared to the R-tree method.

Fig. 10(a) shows the access time performance of the bR-tree method and R-tree method as a function of α, which indicates the relative importance between the spatial distance and non-spatial distance in a GkNN query. Because the access time performance is mainly affected by the (i) size of index structures, (ii) number of data objects to be broadcast, and (iii) result size (i.e., the value of k), the performance of both methods is practically unaffected by the value of α. However, the bR-tree method outperforms the R-tree method because the size of the bR-tree is smaller than that of the R-tree. On average, the bR-tree method incurs 89.1% of the access time of the R-tree method.

On the other hand, as shown in Fig. 10(b) and (c), the tuning time and memory requirement of both methods decrease when the value of α increases. We observe that for higher values of α ($\geqslant$0.5), the bR-tree method significantly outperforms the R-tree method because the R-tree is built on the basis of not only the spatial attributes but also all the non-spatial attributes, whereas the bR-tree is built on the basis of only the spatial attributes. As compared to the R-tree method, on average, the bR-tree method requires 61.1% of the tuning time and 62.7% of the memory space. When $\alpha \geqslant 0.5$, the bR-tree method requires only 44% of the tuning time and 47.22% of the memory space, as compared to the R-tree method.

Fig. 11 shows the performance of the bR-tree method and R-tree method with respect to the bucket capacity. The access time, tuning time, and memory requirement of both methods decrease as the bucket capacity increases because a larger

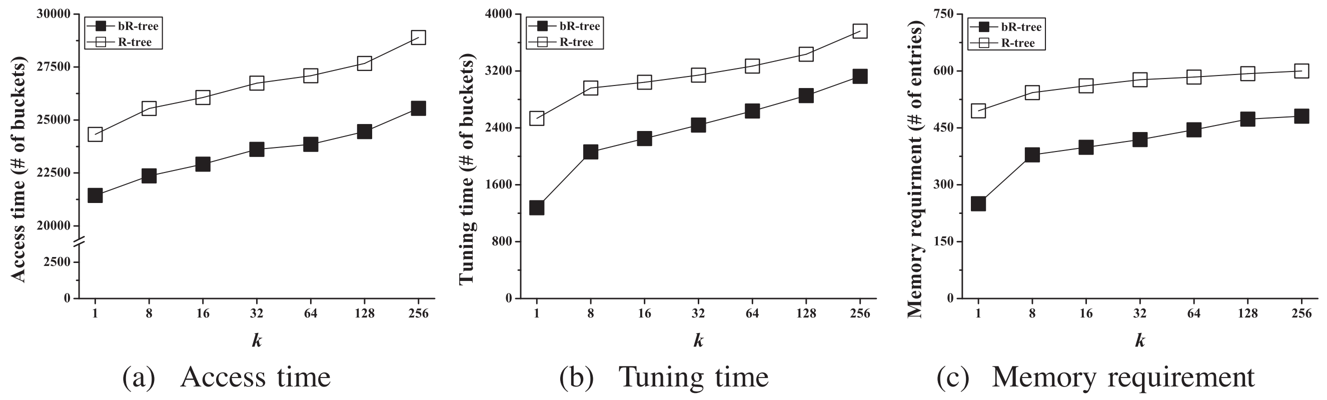
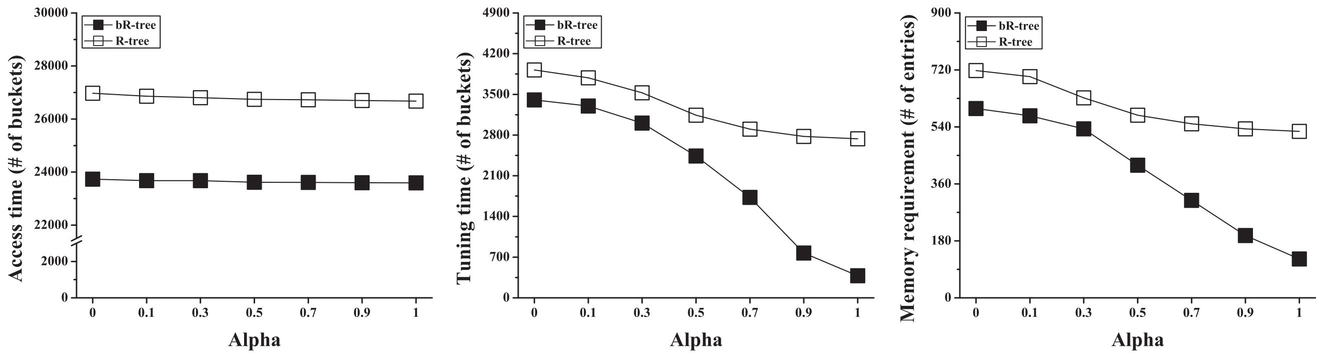
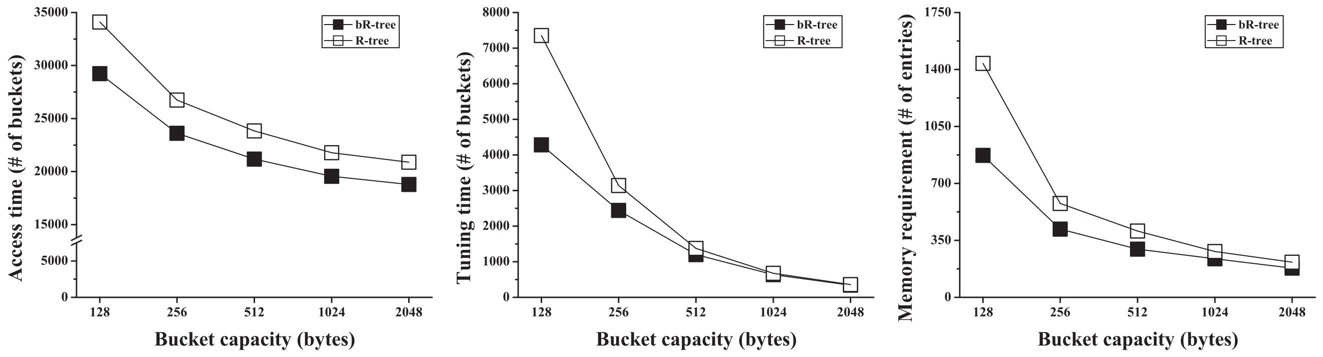


Fig. 9. Access time, tuning time, and memory requirement vs. k.



(a) Access time (b) Tuning time (c) Memory requirement

Fig. 10. Access time, tuning time, and memory requirement vs. α.

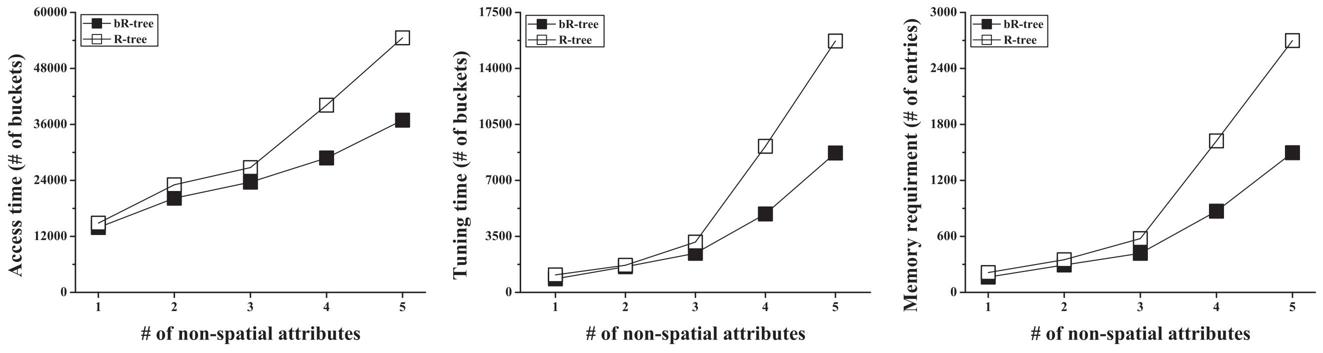


(a) Access time (b) Tuning time (c) Memory requirement

Fig. 11. Access time, tuning time, and memory requirement vs. bucket capacity.

capacity of the bucket increases the fan-out of the bR-tree/R-tree. However, as expected, the bR-tree method outperforms the R-tree method.

Finally, we varied the number of non-spatial attributes and evaluated the performance of the bR-tree method and R-tree method. As shown in Fig. 12(a), the access time of both methods increases with the number of non-spatial attributes because the increased number of non-spatial attributes increases the size of the bR-tree and R-tree. Similarly, as shown in Figs. 12(b) and (c), the tuning time and memory requirement of both methods increase because of the smaller fan-out of the bR-tree and R-tree. However, the bR-tree method outperforms the R-tree method because the fan-out of the bR-tree is much larger than that of the R-tree and the R-tree coarsely approximates the non-spatial attribute values (together with the values of the spatial attributes) into the hyper-mbrs of non-leaf nodes as the number of non-spatial attributes increases. As compared to the R-tree method, on average, the bR-tree method requires 81.8% of the access time, 71.8% of the tuning time, and 70.6% of the memory space.



(a) Access time (b) Tuning time (c) Memory requirement

Fig. 12. Access time, tuning time, and memory requirement vs. # of non-spatial attributes.

Table 2
Non-spatial attributes in the dataset.

Attribute	Domain	# of distinct values
a_1: the quality of food	$0 \sim 30$	5
a_2: the quality of service	$0 \sim 30$	12
a_3: the quality of decor	$0 \sim 30$	17
a_4: price	$0 \sim 520$	291

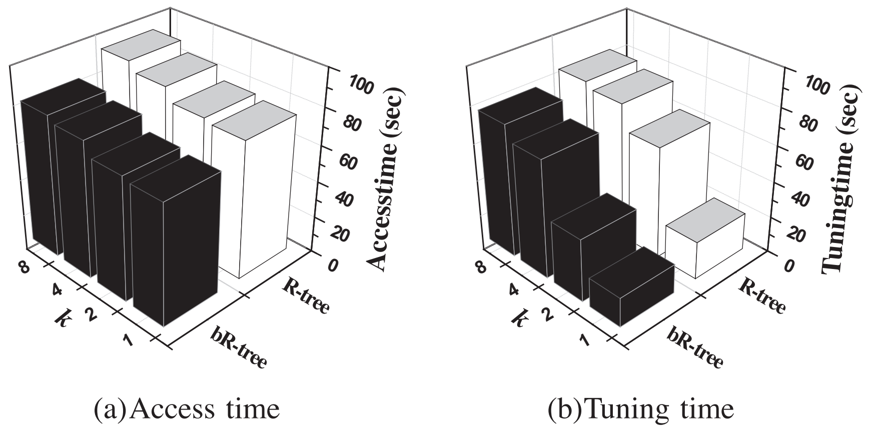


Fig. 13. Access and tuning times vs. *k*.

5.3. Empirical test on a real system

We empirically tested the practicality of the proposed method in a real system [10]. The system consists of a broadcasting server and a mobile client. The server continuously sends UDP packets via a single WiFi (IEEE 802.11.9) channel, where the size of a packet is 1024 bytes and the receiver address is not specified. The mobile client (Nokia N810 PDA) scans the channel and selectively receives packets from the channel. The doze mode is enabled by the *flight mode*, during which all communication devices are temporarily switched off. We generated a dataset of 291 imaginary restaurants, where the locations of the restaurants are represented by a real spatial dataset[7] containing details of cultural landmarks in California. In addition, each restaurant is associated with 4 non-spatial attributes, collected from the Zagat ratings.[8] Table 2 summarizes these 4 non-spatial attributes.

By using the dataset described above, we generated the broadcast stream, where the bR-tree/R-tree is interleaved using the distributed indexing strategy. Each restaurant object was assumed to occupy one packet. For the bR-tree method, we allocated 2 bytes to the set of node bit-vectors $E.bv_1 \sim E.bv_4$ assigned to the non-spatial attributes $a_1 \sim a_4$. To test the performance of the proposed methods, we generated 400 queries by setting α to 0.5 and varying the value of k from 1 to 8. Then, we measured the average access time and tuning time of the bR-tree method and R-tree method. Note that each query randomly specifies its location and 4 non-spatial target values over the set of restaurants, and both the spatial distance and the non-spatial distance to the query are scaled to be within the same range when calculating the generalized distance of each restaurant. As shown in Fig. 13, the bR-tree method outperforms the R-tree method in terms of the access time and tuning time.

6. Conclusions

In this paper, we addressed the problem of processing generalized k-nearest neighbor (GkNN) queries in a wireless broadcasting system. The primary goal of GkNN queries is to find the k best data objects, determined according to the mobile user's specifications on the spatial and non-spatial attributes of data objects. To process GkNN queries on the broadcast stream, we proposed the bR-tree, where each non-leaf node augments the bit-vector information generated to effectively describe the non-spatial attribute values of the data objects in its subtree. In addition, each bR-tree node stores only one pointer to its children, and hence compact. We also presented a search algorithm for the broadcast bR-tree. We carried out simulation experiments and showed that the bR-tree method outperforms the conventional R-tree-based naïve method, thereby validating the effectiveness of the bR-tree for processing GkNN queries on the broadcast stream.

[7] Available at http://www.rtreeportal.org.
[8] Available at http://www.zagat.com.

References

[1] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, Broadcast disks: data management for asymmetric communications environments, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 1995, pp. 199–210.
[2] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 1990, pp. 322–331.
[3] S. Berchtold, D.A. Keim, H.P. Kriegel, The X-tree: an index structure for high-dimensional data, in: Proceedings of International Conference on Very Large Data Bases, 1996, pp. 28–39.
[4] C. Bohm, S. Berchtold, D.A. Keim, Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases, ACM Computing Surveys 33 (3) (2001) 322–373.
[5] L. Chen, M. Lv, Q. Ye, G. Chen, J. Woodward, A personal route prediction system based on trajectory data mining, Information Sciences 181 (7) (2011) 1264–1284.
[6] K.L. Cheung, A. Fu, Enhanced nearest neighbour search on the R-tree, ACM SIGMOD Record 27 (3) (1998) 16–21.
[7] Y.D. Chung, M.H. Kim, An index replication scheme for wireless data broadcasting, Journal of Systems and Software 51 (3) (2000) 191–199.
[8] Y.D. Chung, An indexing scheme for energy-efficient processing of content-based retrieval queries on a wireless data stream, Information Sciences 177 (2) (2007) 525–542.
[9] Y.D. Chung, J.Y. Lee, An indexing method for wireless broadcast XML data, Information Sciences 177 (9) (2007) 1931–1953.
[10] Y.D. Chung, S. Yoo, M.H. Kim, Energy and latency efficient processing of full-text searches over a wireless broadcast stream, IEEE Transactions on Knowledge and Data Engineering 22 (2) (2010) 207–218.
[11] I. DeFelipe, V. Hristidis, N. Rishe, Keyword search on spatial databases, in: Proceedings of IEEE International Conference on Data Engineering, 2008, pp. 656–665.
[12] D.H. Francis, S.K. Madria, C. Sabharwal, A scalable constraint-based Q-hash indexing for moving objects, Information Sciences 178 (6) (2008) 1442–1460.
[13] B. Gedik, A. Singh, L. Liu, Energy efficient exact kNN search in wireless broadcast environments, in: Proceedings of Annual ACM International Workshop on Geographic Information Systems, 2004, pp. 137–146.
[14] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 1984, pp. 47–57.
[15] G.R. Hjaltason, H. Samet, Distance browsing in spatial databases, ACM Transactions on Database Systems 24 (2) (1999) 265–318.
[16] Q. Hu, W.-C. Lee, D.L. Lee, A hybrid index technique for power efficient data broadcast, Distributed and Parallel Databases 9 (2) (2001) 151–177.
[17] Y. Hua, B. Xiao, J. Wang, BR-Tree: a scalable prototype for supporting multiple queries of multidimensional data, IEEE Transactions on Computers 58 (12) (2009) 1585–1598.
[18] T. Imielinski, S. Viswanathan, B.R. Bardrinath, Data on air: organization and access, IEEE Transactions on Knowledge and Data Engineering 9 (3) (1997) 353–372.
[19] H. Jung, B.K. Cho, Y.D. Chung, L. Liu, On processing location based top-*k* queries in the wireless broadcasting system, in: Proceedings of ACM Symposium on Applied Computing, 2010, pp. 585–591.
[20] K.V.R. Kanth, S. Ravada, J. Sharma, J. Banerjee, Indexing medium-dimensionality data in Oracle, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 1999, pp. 521–522.
[21] K.V.R. Kanth, S. Ravada, D. Abugov, Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2002, pp. 546–557.
[22] I. Lazaridis, S. Mehrotra, Progressive approximate aggregate queries with a multi-resolution tree structure, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2001, pp. 401–412.
[23] D.L. Lee, W.-C. Lee, J. Xu, B. Zheng, Data management in location-dependent information services: challenges and issues, IEEE Pervasive Computing 1 (3) (2002) 65–72.
[24] S.T. Leutenegger, J.M. Edgington, M.A. Lopez, STR: a simple and efficient algorithm for R-tree packing, in: Proceedings of IEEE International Conference on Data Engineering, 1997, pp. 497–506.
[25] C.-M. Liu, S.-Y. Fu, Effective protocols for kNN search on broadcast multi-dimensional index trees, Information Systems 33 (1) (2008) 18–35.
[26] K. Liu, V.C.S. Lee, On-demand broadcast for multiple-item requests in a multiple-channel environment, Information Sciences 180 (22) (2010) 4336–4352.
[27] S.K. Madria, B. Bhargava, E. Pitoura, V. Kumar, Data organization issues for location-dependent queries in mobile computing, Lecture Notes in Computer Science 1884 (2000) 142–156.
[28] R. Orlandic, B. Yu, A retrieval technique for high-dimensional data and partially specified queries, Data and Knowledge Engineering 42 (1) (2002) 1–21.
[29] D. Papadias, P. Kalnis, J. Zhang, Y. Tao, Efficient OLAP operations in spatial data warehouses, Lecture Notes in Computer Science 2121 (2001) 443–459.
[30] D. Papadias, Y. Tao, G. Fu, B. Seeger, Progressive skyline computation in database systems, ACM Transactions on Database Systems 30 (1) (2005) 41–82.
[31] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 1995, pp. 71–79.
[32] S. Saltenis, C. Jensen, S. Leutenegger, M. Lopez, Indexing the positions of continuously moving objects, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2000, pp. 331–342.
[33] M. Sharifzadeh, C. Shahabi, VoR-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries, Proceedings of the VLDB Endowment 3 (1) (2010) 1231–1242.
[34] R.R. Sinha, M. Winslett, Multi-resolution bitmap indexes for scientific data, ACM Transactions on Database Systems 32 (3) (2007) 16.
[35] Y. Tao, D. Papadias, Performance analysis of R*-Tree with arbitrary node extents, IEEE Transactions on Knowledge and Data Engineering 16 (6) (2004) 653–668.
[36] Y. Tao, X. Xiao, J. Pei, Efficient skyline and top-k retrieval in subspaces, IEEE Transactions on Knowledge and Data Engineering 19 (8) (2007) 1072–1088.
[37] A.B. Waluyo, W. Rahayu, D. Taniar, B. Srinivasan, A novel structure and access mechanism for mobile data broadcast in digital ecosystems, IEEE Transactions on Industrial Electronics 58 (6) (2011) 2173–2182.
[38] J. Xu, W.-C. Lee, X. Tang, Exponential index: A parameterized distributed indexing scheme for data on air, in: Proceedings of International Conference on Mobile Systems, Applications, and Services, 2004, pp. 153–164.
[39] B. Yu, T. Bailey, Processing partially specified queries over high-dimensional databases, Data and Knowledge Engineering 62 (1) (2007) 177–197.
[40] B. Zheng, W.-C. Lee, D.L. Lee, Search *k* nearest neighbors on air, Lecture Notes in Computer Science 2574 (2003) 181–195.
[41] B. Zheng, W.-C. Lee, D.L. Lee, Spatial queries in wireless broadcast systems, Wireless Network 10 (6) (2004) 723–736.
[42] J. Zhong, W. Wu, Y. Shi, X. Gao, Energy-efficient tree-based indexing schemes for information retrieval in wireless data broadcast, Lecture Notes in Computer Science 6588 (2011) 335–351.