On Processing Location Based Top-k Queries in the Wireless Broadcasting System

HaRim Jung[†], ByungKu Cho[†], Yon Dohn Chung[†] and Ling Liu[‡]
[†]Department of Computer Science and Engineering, Korea University, Seoul, Korea
{harim3826, byungku, ydchung}@korea.ac.kr
[‡]College of Computing, Georgia Institute of Technology, USA
lingliu@cc.gatech.edu

ABSTRACT

In this paper, we explore the problem of processing a novel type of location based queries, named the location based top-k query, which involves both of spatial and non-spatial specifications for data objects in the wireless broadcasting system. We introduce two methods for processing location based top-k queries on the broadcast stream. In the first method, the search algorithm runs on the broadcast aggregate R-tree (aR-tree). However, the aR-tree may deteriorate the search performance, especially in terms of the tuning time. With this problem in mind, we propose a novel index structure, called the bit-vector R-tree (bR-tree), which stores additional bit-vector information to facilitate processing of location based top-k queries. The search algorithm on the broadcast bR-tree is also described. Our simulation experiments demonstrate that the bR-tree method clearly outperforms the aR-tree method in terms of the tuning time, while maintaining similar or better performance in terms of the access time.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Application—*spatial databases and GIS*

General Terms

Algorithms, Design, Performance

Keywords

Wireless broadcasting system, Location based service, Location based top-k query

1. INTRODUCTION

With the wide spread of wireless networks and the emergence of positioning technologies like GPS, Location Based Services (LBSs) have emerged as one of the most promising applications in mobile computing environments. Mobile users, equipped with hand-held devices, can now access a variety of valuable information from anywhere and at any time. In mobile computing environments, *wireless broadcasting* is considered to be an effective way for provisioning LBSs since it accommodates an unlimited number of mobile users simultaneously at a constant cost [2-5, 16].

The main functionality of LBSs is to process *location based queries* like range queries and k-Nearest Neighbor (k-NN) queries. However, such conventional location based queries only consider the spatial proximity of data objects to mobile users' locations. Consequently, they may be insufficient for real-life applications, which usually take both spatial and non-spatial attributes of data objects into consideration to reflect the users' preferences. This paper introduces a novel class of location based queries: a *location based top-k query.* Assume a set of data objects $\mathcal{D}$, with each data object being associated with both spatial attributes and a set of non-spatial attributes. Then, the location based top-k 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 the ranked set of k data objects with the lowest *proximity scores* in $\mathcal{D}$. Without loss of generality, we assume that a lower score is more preferable than a higher score. The score of each data object is determined according to the combination of the *spatial distance* to the user's location and the *non-spatial distance* to the user-specified target values. In the following, we illustrate the motivational example to our work.

Example 1. *Consider the server that periodically broadcasts the dataset of local restaurants $\mathcal{R} = \{r_1, r_2, \cdots, r_8\}$ shown in Figure 1, where the locations of the restaurants are depicted with their average prices. Suppose that a mobile user at location q.p = (10, 8) asks for one (k = 1) nearby restaurant and his/her expected price is \$10. If both the location and price of a restaurant are equally important to the user, a mobile client*[1] *tunes into the broadcast channel and retrieves r_7 that best matches the user's specifications. Note that although r_3 is the nearest restaurant to the user's location, its price (= \$20) is much more expensive than the user-expected price, and thus it cannot be included in the result.*

In this paper, we propose the methods for processing location based top-k queries in the wireless broadcasting system. In particular, we make the following contributions:

[1]Without ambiguity, we use the terms hand-held device and mobile client interchangeably.

- We explore, for the first time to the best of our knowledge, the problem of processing location based top-k queries that involve both of spatial and non-spatial specifications for data objects in the wireless broadcasting system.
- We first propose a naïve method, where the search algorithm runs on the broadcast *aggregate R-tree* (*aR-tree*) [6, 9]. Then, we present a novel index structure called the *bit-vector R-tree* (*bR-tree*), which stores additional bit-vector information to improve the search performance. We also describe the search algorithm on the broadcast bR-tree.
- We show through simulation experiments that the bR-tree method outperforms the aR-tree method in terms of both the *access time* and the *tuning time*, which are commonly used as the main performance measures in the wireless broadcasting system.

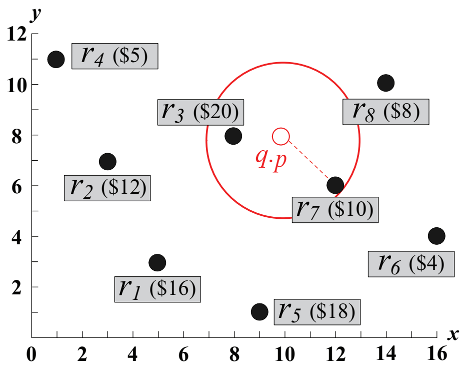


Figure 1: Dataset of the local restaurants

The remainder of this paper is organized as follows. In Section 2, background and related work are presented. In Section 3, details of the proposed methods for processing location based top-k queries are described. In Section 4, the performance evaluation results are presented. Finally, in Section 5, we conclude the paper.

2. BACKGROUND AND RELATED WORK

2.1 Wireless Data Broadcasting

In the wireless broadcasting system, the server periodically broadcasts data objects through the public wireless channel in a predetermined sequential order. When each mobile client receives a query from its user, it tunes into the channel and processes the query on the broadcast stream. In this way, the server pushes the query processing task entirely to the mobile client side, and thus the server load is independent of mobile user population.

Two performance measures are commonly used in the wireless broadcasting system: the access time and the tuning time [14]. The former is the duration elapsed from the moment a mobile client receives the query from its user to the moment the query is satisfied. The latter is the duration during which the mobile client remains in the full operational mode, called the *active mode*, which is proportional to the amount of the energy consumed by the mobile client.

Air index interleaving [2-5, 14-16] is commonly used for reducing the tuning time at the expense of the increased access time. The basic idea is to interleave an index information with data objects on the broadcast stream. By first examining the index information, the mobile client can get the arrival times of the relevant data objects for the given query. As a result, the mobile client selectively scans only the relevant portions of the broadcast stream, by switching between the active mode and the power conserving mode, referred to as the *doze mode*, during its access time. The (1, m) interleaving [14] is the most representative method, where the entire index information is replicated m times and inserted into every $\frac{1}{m}$ fraction of the broadcast stream.

The basic unit of wireless data broadcasting is called the *bucket*, and the broadcast stream consists of *index buckets* and *data buckets*. Index buckets hold the index information, while data buckets hold the data. We assume that all buckets have the same capacity irrespective of their types. In this paper, we use the number of buckets to measure the access time and the tuning time.

2.2 Location Based Query Processing

Since Guttman's work [1], the R-tree has been extensively used as the access method for multi-dimensional data (e.g., spatial data), and many R-tree variant index structures have been proposed [6, 8-9, 11]. Figure 2 shows the R-tree that indexes the dataset in Figure 1. The R-tree naturally suits for processing of a range query. Starting from the root, the query is processed by only considering the entries whose minimum bounding rectangles (*mbr*s) intersect a given query region. Regarding processing of a k-NN query on the R-tree, several algorithms have been proposed [7, 10, 12]. Most of them follow a branch-and-bound approach, by utilizing the following distance metrics to prune the search space: *mindist* and *minmaxdist* [12]. Given a query point $q.p$ and a non-leaf entry E, *mindist*($q.p$, $E.mbr$) indicates the minimum possible distance between $q.p$ and any data objects covered by the subtree of E. On the other hand, *minmaxdist*($q.p$, $E.mbr$) guarantees that the subtree of E covers at least one data object within this distance from $q.p$.

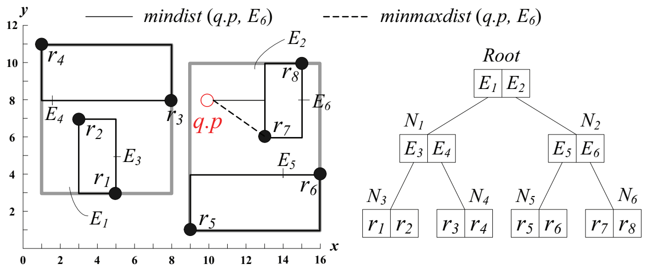


Figure 2: The R-tree index structure

The *best-first* algorithm proposed in [7] is considered to be the most efficient solution for processing k-NN queries, which dynamically determines the traversal order of the R-tree. However, the performance of the best-first algorithm may deteriorate significantly in terms of the access time in the wireless broadcasting system due to the *sequential property* of the broadcast stream. The broadcast stream periodically flows in a predetermined sequential order, and each R-tree node is only available when it appears on the broad-cast stream. Whenever the dynamic traversal order of the best-first algorithm differs from the appearance order of the R-tree nodes on the broadcast stream, the access time is significantly extended. For example, suppose that the R-tree is broadcast once in each version of the broadcast stream as shown in Figure 3. When the best first algorithm tries to visit N_1 after visiting N_2, it has to wait for the next broadcast stream, since N_1 has already been broadcast.

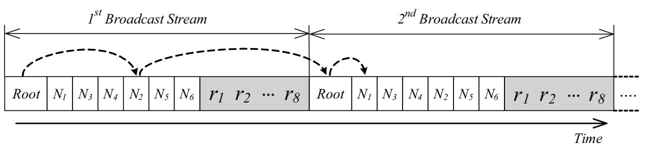


Figure 3: Broadcast R-tree traversal

By taking the sequential property of the broadcast stream into account, the *appearance-first* algorithm has been considered in [2, 4] for improving the access time performance. The appearance-first algorithm sequentially visits the R-tree nodes in the order of their appearance on the broadcast stream, while filtering out the unqualified nodes according to the *mindist*- and *minmaxdist*-based heuristics.

3. THE PROPOSED METHODS

3.1 Problem Definition

Let $\mathcal{D} = \{d_1, d_2, \cdots, d_{|\mathcal{D}|}\}$ be a set of data objects, each of which is associated with spatial attributes $p = (x,\ y)$ and a set of non-spatial attributes $a = \{a_1, a_2,\ \cdots, a_n\}$. 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 $d.a = \{d.a_1, d.a_2,\ \cdots, d.a_n\}$ denotes the values of its non-spatial attributes. A location based top-k 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, $q.v = \{q.v_1, q.v_2, \cdots, q.v_n\}$ denotes a set of the target values that the user specifies on a, and k is an integer value indicating the desired number of the top-k data objects. The result of q is a ranked set of the k data objects with the lowest proximity scores in $\mathcal{D}$. The score of each data object d with respect to q is determined according to the following expression:

$$score(q, d) = dist(q.p, d.p) + dist(q.v, d.a),$$

where $dist(q.p, d.p) = \sqrt{(q.x - d.x)^2 + (q.y - d.y)^2}$ is the spatial distance and $dist(q.v, d.a) = \sum_{i=1}^{n} |q.v_i - d.a_i|$ is the non-spatial distance from q to d.[2]

In the rest of this section, we make some assumptions to facilitate our statement of the problem. Specifically, we assume that each data object $d \in \mathcal{D}$ is associated with only one non-spatial attribute a. In consequence, a query q is assumed to specify only one non-spatial target value v on a, and thus $dist(q.v, d.a) = |q.v - d.a|$. However, our methods can be naturally extended without any modification for the case of multiple non-spatial attributes, as will be shown in Section 4. In addition, we assume that $dist(q.p, d.p)$ and $dist(q.v, d.a)$ are scaled to be within $[0, 1]$.

3.2 The aR-tree method

3.2.1 The Aggregate R-tree (*aR-tree*)

The conventional R-tree used in spatial databases deals with only spatial attributes, and thus is not appropriate for processing location based top-k queries. Therefore, the first naïve method makes use of the aR-tree [6, 9]. Let Sub_E be a set of data objects covered by the subtree of a non-leaf entry E. Then, the aR-tree used in this method is the R-tree variant index structure that additionally satisfies the following:

- Each leaf entry of the aR-tree is associated with the non-spatial attribute value of the data object pointed by it;
- Each non-leaf entry E of the aR-tree augments the non-spatial interval $E.iv = [iv^-, iv^+]$, where iv^- and iv^+ are the minimum and the maximum values, respectively, among the non-spatial attribute values of all the data objects in Sub_E.

Figure 4 shows an example of the aR-tree that indexes the dataset in Figure 1, where we can observe an important property of the aR-tree.

PROPERTY 1. *Given a non-leaf entry E of the aR-tree, $E.iv$ ensures that $\forall d \in Sub_E$, $iv^- \leq d.a \leq iv^+$.*

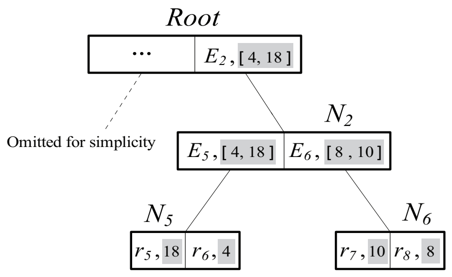


Figure 4: Example of the aR-tree

3.2.2 Search Algorithm on the Broadcast aR-tree

The search algorithm on the broadcast aR-tree sequentially visits the aR-tree nodes in the order of their appearance on the broadcast stream, while pruning the branches that are guaranteed to fail. Before presenting the detailed search algorithm, the following metrics, defined between a query q and a non-leaf entry E of the aR-tree, are introduced: *minscore* and *maxscore*.

DEFINITION 1. *Given a query q and a non-leaf entry E of the aR-tree, the following expression calculates the minscore of E with respect to q:*

$$minscore(q, E) = mindist(q.p, E.mbr) + mindist(q.v, E.iv),$$

where $mindist(q.v, E.iv) =$

$$\begin{cases} 0, \text{ if } & iv^- \leq q.v \leq iv^+; \\ \min(|q.v - iv^-|, |q.v - iv^+|), & \text{otherwise.} \end{cases}$$

LEMMA 1. *Given a query q and a non-leaf entry E of the aR-tree, $\forall d \in Sub_E$, minscore $(q, E) \leq score(q, d)$.*

PROOF. Since we know from [12] that $\forall d \in Sub_E$, *mindist* $(q.p,\ E.mbr) \leq dist(q.p,\ d.p)$, we only need to prove that $\forall d \in Sub_E$, $mindist(q.v, E.iv) \leq dist(q.v, d.a)$. For this purpose, we proceed with two cases:

[2]Although $dist(q.p, d.p)$ and $dist(q.v, d.a)$ can be associated with different weights from each other, we assign equal weight to them in this paper.

1. If $iv^- \leq q.v \leq iv^+$, $mindist(q.v, E.iv) \leq dist(q.v, d.a)$ for any data object $d \in Sub_E$ since $mindist(q.v, E.iv) = 0$.

2. Otherwise, by Property 1 and Definition 1, we have that $\forall d \in Sub_E$, $\min(|q.v - iv^-|, |q.v - iv^+|) \leq |q.v - d.a|$. Therefore, $mindist(q.v, E.iv) \leq dist(q.v, d.a)$ for any data object $d \in Sub_E$. $\square$

Definition 2. *Given a query q and a non-leaf entry E of the aR-tree, the following expression calculates the maxscore of E with respect to q:*

$$maxscore(q, E) = minmaxdist(q.p, E.mbr) + maxdist(q.v, E.iv),$$

where $maxdist(q.v, E.iv) = \max(|q.v - iv^-|, |q.v - iv^+|)$.

Lemma 2. *Given a query q and a non-leaf entry E of the aR-tree,* $\exists d \in Sub_E : score(q,d) \leq maxscore(q, E)$.

Proof. Similar to the proof of Lemma 1, since we know from [12] that $\exists d \in Sub_E : dist(q.p, d.p) \leq minmaxdist(q.p, E.mbr)$, it suffices to prove that $\forall d \in Sub_E$, $dist(q.v, d.a) \leq maxdist(q.v, E.iv)$. By Property 1 and Definition 2, we have that $\forall d \in Sub_E$, $|q.v - d.a| \leq \max(|q.v - iv^-|, |q.v - iv^+|)$. Hence, $dist(q.v, d.a) \leq maxdist(q.v, E.iv)$ for any data object $d \in Sub_E$. $\square$

The above definitions and lemmas establish the foundation for pruning the search space. Specifically, given a query q and each non-leaf entry E, if there exist at least k data objects whose scores are lower than $minscore(q, E)$, E does not have to be considered since its subtree cannot cover any better data object (by Lemma 1). On the other hand, $maxscore(q, E)$ can help conservatively estimate the scores of the current top-k data objects during the search process since $maxscore(q, E)$ ensures that the subtree of E covers at least one data object whose score is lower than or equal to it (by Lemma 2). The details of the search algorithm are given in Algorithm 1, where two data structures below are used:

- *Search_list*: *Search_list* stores non-leaf entries whose subtrees should be traversed. The entries stored in *Search_list* are sorted according to the appearance order of their subtrees on the broadcast stream.

- *Candidate_list*: *Candidate_list* stores either non-leaf or leaf entries whose subtrees cover the current top-k data objects during the search process. The entries stored in *Candidate_list* are sorted according to the ascending order of their *maxscore* (or actual score) values. *Candidate_list* can store at most k entries.[3]

The search algorithm performs an initial probe, i.e., tunes into the broadcast channel to find out when the next root of the aR-tree appears on the broadcast stream (line 1). After the initial probe, the algorithm accesses the root and inserts all its entries into *Search_list* (line 2). Then, the algorithm iteratively examines the entries stored in *Search_list* until *Search_list* becomes empty. Let *candidate_score* be the *maxscore* (or score) value of the k^{th} entry stored in *Candidate_list*. In case there are fewer than k entries in *Candidate_list*, *candidate_score* $= \infty$. At each iteration, the algorithm first removes the top entry E from *Search_list* and checks if $candidate\_score < minscore(q, E)$. If this is the case, the algorithm safely skips E and proceeds with the next iteration (lines 5-7). Otherwise, the algorithm accesses the node N pointed by E, removes E from *Candidate_list* if it is there (lines 8-10), and thereafter performs the following:

- If child entries obtained from N are non-leaf: For each child entry, denoted as E_{child}, the algorithm checks if $candidate\_score < minscore(q, E_{child})$. If so, E_{child} is skipped (lines 13-14). Otherwise, it is inserted into *Search_list* and *Candidate_list* (lines 15-16). Let us note that E_{child} is inserted into *Candidate_list* only if its *maxscore* value is less than *candidate_score*.

- If child entries obtained from N are leaf: For each child entry E_{child}, the algorithm checks if $candidate\_score < score(q, d)$. If so, E_{child} is skipped (lines 19-20). Otherwise, it is only inserted into *Candidate_list* (lines 21-22).

Then, the algorithm proceeds with the next iteration. When *Search_list* becomes empty, the algorithm retrieves the data objects, pointed by the entries in *Candidate_list*, as the final result (lines 24-25).

Algorithm 1 Search algorithm on the broadcast aR-tree

```
Input a location based top-$k$ query $q$, $k$
Output the result of $q$
Procedure
1:  Perform initial probe;
2:  $Search\_list = \{Root\}$;  //sorted by appearance order
3:  $Candidate\_list = \emptyset$;  //sorted by $maxscore$ (or score), size=$k$
4:  do{
5:      Remove the next entry $E$ from $Search\_list$;
6:      if $candidate\_score < minscore(q, E)$
7:          continue;
8:      else
9:          Remove $E$ from $Candidate\_list$;
10:         Access the node $N$ pointed by $E$;
11:         if child entries are non-leaf
12:             for each child entry $E_{child}$
13:                 if $candidate\_score < minscore(q, E_{child})$
14:                     continue;
15:                 else
16:                     Insert $E_{child}$ into $Search\_list$ & $Candidate\_list$;
17:         else  //child entries are leaf
18:             for each child entry $E_{child}$
19:                 if $candidate\_score < score(q, d)$
20:                     continue;
21:                 else
22:                     Insert $E_{child}$ into $Candidate\_list$;
23: }while ($Search\_list$ is empty)
24: Retrieve the data objects pointed by entries $\in Candidate\_list$;
25: Return the result;
End of Procedure
```

3.2.3 Discussion

The search algorithm on the broadcast aR-tree skips all the non-leaf entries whose *minscore* values are greater than the current *candidate_score*. However, each non-leaf entry E of the aR-tree augments typically large non-spatial interval $E.iv = [iv^-, iv^+]$; this negatively affects the pruning capability of the search algorithm because $q.v$ of a query q is likely to lie within $E.iv$ (i.e., $iv^- \leq q.v \leq iv^+$), which makes $minscore(q, E)$ inaccurately underestimated. Please note that if $iv^- \leq q.v \leq iv^+$, $mindist(q.v, E.iv) = 0$.

In addition, the size of E is fairly large since E augments additional two floating-point numbers to represent

[3]Consequently, when a new entry is inserted into *Candidate_list*, the current k^{th} element is removed from *Candidate_list* in case *Candidate_list* has already stored k entries.

$E.iv$. In the case of a dataset with multiple non-spatial attributes, the size of E would be significantly enlarged. Let $f_{max}=\lfloor\frac{\textit{the bucket capacity}}{\textit{the size of } E}\rfloor$be the maximum fan-out of a non-leaf node in the aR-tree.[4] As the size of E increases, the value of f_{max} decreases. Consequently, the overall search cost in terms of the tuning time increases due to the height growth of the aR-tree. Since the search algorithm sequentially passes through the broadcast aR-tree to get the final result, the access time performance deteriorates as well when the size of E increases.

3.3 The bR-tree Method

3.3.1 The Bit-vector R-tree (bR-tree)

Motivated by the observations mentioned above, we present the bR-tree for processing location based top-k queries in a more efficient manner on the broadcast stream. The bR-tree is an R-tree variant index structure, where each non-leaf entry E augments a bit-vector information to describe the non-spatial attribute values of all the data objects in Sub_E. Specifically, let N be the total number of distinct non-spatial attribute values of the dataset $\mathcal{D}$, where $N \leq |\mathcal{D}|$. Then, we classify N non-spatial attribute values into $\mathcal{K}$ $(\leq N)$ clusters and generate a *node bit-vector* for each non-leaf entry E of the bR-tree. We use the $\mathcal{K}$*-means* clustering algorithm for simplicity, but other sophisticated clustering algorithms can be employed.

DEFINITION 3. ***Node bit-vector***: *Suppose that a bit is assigned for each cluster* c_i $(1 \leq i \leq \mathcal{K})$*. Then, the node bit-vector* $E.bv$*, associated with a non-leaf entry* E*, is the* $\mathcal{K}$*-bit binary vector such that for each bit position* b_i $(1 \leq i \leq \mathcal{K})$ *in* $E.bv$*,* b_i*="1", if and only if the following condition holds* : $\exists d \in Sub_E : d.a \in c_i$.

As a consequence of augmenting a node bit-vector to each non-leaf entry, the non-leaf node of the bR-tree contains entries of the form (*node_ptr*, *mbr*, *bv*), where *node_ptr* and *mbr* have their usual meanings. The use of a node bit-vector is beneficial since it leads to the small size of a non-leaf entry. As with the aR-tree, each leaf node of the bR-tree contains leaf entries, each of which is associated with the non-spatial attribute value of the data object pointed by it. Figure 5 shows the bR-tree, assuming the number of bits in a node bit-vector is 4 (i.e., $\mathcal{K}$=4). For example, the subtree of E_5 covers the restaurants r_5 and r_6, whose prices 18 and 4 are in the clusters c_4 and c_1 respectively; hence, the node bit-vector augmented in E_5 is "1001".

An important observation is that each cluster c in the figure can indicate the disjoint *cluster interval* whose boundary consists of the minimum and the maximum prices in c (e.g., $c_1 \Rightarrow [4, 5]$ and $c_4 \Rightarrow [18, 20]$). Accordingly, $E_5.bv$ can indicate that the price of each restaurant $r \in Sub_{E_5}$ lies within one of the cluster intervals [4, 5] and [18, 20].

PROPERTY 2. *Given a non-leaf entry* E *of the bR-tree, let* C $(|C| \leq \mathcal{K})$ *be the set of clusters whose corresponding bit positions in* $E.bv$ *are set to "1". We denote the minimum and the maximum values in each cluster* $c \in C$ *as* c^- *and* c^+ *respectively. Then,* $E.bv$ *can indicate that* $\forall d \in Sub_E$, $\exists c \in C : c^- \leq d.a \leq c^+$.

[4]Note that, because the basic unit of wireless data broadcasting is the bucket, as mentioned in Section 2.1, an index bucket corresponds to a node in the aR-tree.

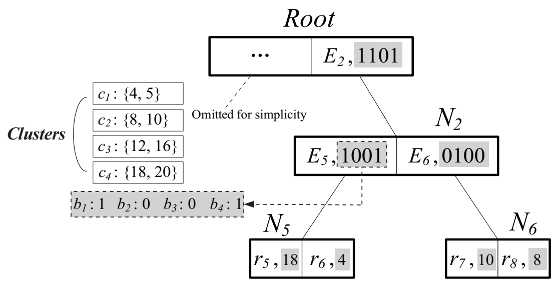


Figure 5: Example of the bR-tree

3.3.2 Search Algorithm on the Broadcast bR-tree

The search algorithm on the broadcast bR-tree also sequentially visits the bR-tree nodes in the order of their appearance on the broadcast stream, while pruning the impossible branches by using the modified *minscore* and *maxscore*.

DEFINITION 4. *Given a query* q *and a non-leaf entry* E *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 minscore defined between* q *and* E *is:*

$$minscore(q, E) = mindist(q.p, E.mbr) + mindist(q.v, E.bv),$$

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

LEMMA 3. *Given a query* q *and a non-leaf entry* E *of the bR-tree,* $\forall d \in Sub_E$*,* $mindist(q.v, E.bv) \leq dist(q.v, d.a)$.

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

1. If $\exists c \in C : c^- \leq q.v \leq c^+$, $mindist(q.v, E.bv) = 0$, which contradicts the above assumption.
2. Otherwise, $mindist(q.v, E.bv) = \min(c^-_{\min}, c^+_{\min})$. Then, the following inequality holds:
$$|q.v - \acute{d}.a| < \min(c^-_{\min}, c^+_{\min}) \qquad (1)$$
By Property 2, we have for some $\acute{c}$ $(\in C)$ that $\min(|q.v - \acute{c}^-|, |q.v - \acute{c}^+|) \leq |q.v - \acute{d}.a|$. On the other hand, we know from Definition 4 that $\min(c^-_{\min}, c^+_{\min}) \leq \min(|q.v - \acute{c}^-|, |q.v - \acute{c}^+|)$. This generates a contradiction to inequality (1). Therefore, $\acute{d}$ cannot exist. □

COROLLARY 1. *Given a query* q *and a non-leaf entry* E *of the bR-tree,* $\forall d \in Sub_E$*, minscore* $(q, E) \leq score(q, d)$.

PROOF. Trivially proved by Lemma 1 and Lemma 3. □

DEFINITION 5. *Given a query* q *and a non-leaf entry* E *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 maxscore defined between* q *and* E *is:*

$$maxscore(q, E) = minmaxdist(q.p, E.mbr) + maxdist(q.v, E.bv),$$

$$\textit{where } maxdist(q.v, E.iv) = \max(c^-_{\max}, c^+_{\max}).$$

LEMMA 4. *Given a query* q *and a non-leaf entry* E *of the bR-tree,* $\forall d \in Sub_E$*,* $dist(q.v, d.a) \leq maxdist(q.v, E.bv)$.

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

$$\max(c^{-}_{\max}, c^{+}_{\max}) < |q.v - \acute{d}.a| \quad (2)$$

By Property 2, we have for some $\acute{c}$ $(\in C)$ that $|q.v - \acute{d}.a| \leq \max(|q.v - \acute{c}^{-}|, |q.v - \acute{c}^{+}|)$. This produces a contradiction to inequality (2) since we know from Definition 5 that $\max(|q.v - \acute{c}^{-}|, |q.v - \acute{c}^{+}|) \leq \max(c^{-}_{\max}, c^{+}_{\max})$. Hence, $\acute{d}$ cannot exist. □

COROLLARY 2. *Given a query q and a non-leaf entry E of the bR-tree, $\exists d \in subtree(E) : score(q, d) \leq maxscore(q, E)$.*

PROOF. Trivially proved by Lemma 2 and Lemma 4. □

The search algorithm on the broadcast bR-tree is analogous to that on the broadcast aR-tree, hence we omit the details. However, it is important to note that the node bit-vector, associated with each non-leaf entry of the bR-tree, can indicate a number of small and discretized non-spatial intervals. This makes the *minscore*, defined for the bR-tree, more accurate compared with the *minscore* defined for the aR-tree. Consequently, the search algorithm on the broadcast bR-tree achieves better pruning capability than that on the broadcast aR-tree.

4. PERFORMANCE EVALUATION

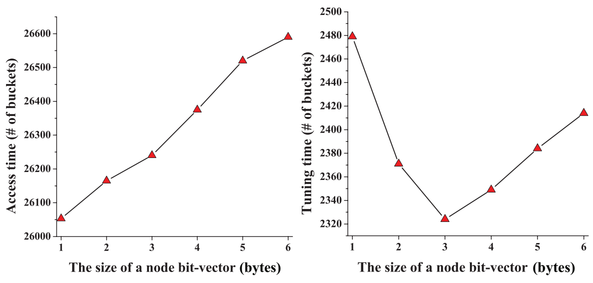
4.1 Experiment Setup

The system model, which consists of a server, mobile clients and a broadcast channel, was implemented in the Java language. We generated 5 sets of data objects (See Table 1), where the locations of data objects are uniformly distributed in a square Euclidian space. In addition, the data objects in the datasets are associated with 3 non-spatial attributes (a_1, a_2 and a_3), which are scaled to [0, 1] and follow a Zipf distribution with parameter 0.8. In each experiment, we generated 1,000 queries and measured the average access and tuning times of the proposed methods in terms of the number of buckets by varying one of the parameters illustrated in Table 1, where the default values of the parameters are typeset in boldface.

Table 1: Parameter Settings

Parameter	Setting
# of data objects	10K, 20K, **30K**, 40K, 50K
k	1, 8, 16, **32**, 64, 128, 256
Bucket capacity	128, **256**, 512, 1024, 2048 bytes

In the construction of the aR-tree and the bR-tree, we employed the *sort tile recursive* (*STR*) *packing* method [13], where the operations for handling non-spatial attributes of the data objects are added. In addition, we employed the (1, m) interleaving method. Here, we used the optimal m to minimize the access time [14]. The size of a data object was assumed to fit into one data bucket irrespective of bucket capacity. For the configuration of each non-leaf entry E of the aR-tree, we allocated 2 bytes to the pointer, 16 bytes to the spatial information ($E.mbr$), and 8 bytes to the non-spatial information ($E.iv_i$) per each non-spatial attribute a_i $(1 \leq i \leq 3)$. On the other hand, for the non-leaf entry E of the bR-tree, an important parameter affecting the access and the tuning time 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 and tuning times of the bR-tree method by varying the size of the node bit-vector $E.bv_i$ assigned to each non-spatial attribute a_i $(1 \leq i \leq 3)$. We set all the parameters in Table 1 to their default values. Note that the *minscore* and the *maxscore*, defined for the bR-tree, can be easily extended for considering 3 non-spatial attributes. For example, $minscore(q, E) = mindist(q.p, E.mbr) + \sum_{i=1}^{3} mindist(q.v_i, E.bv_i)$, and $maxscore(q,E) = minmaxdist(q.p,E.mbr) + \sum_{i=1}^{3} maxdist(q.v_i, E.bv_i)$.[5]



(a) Access time (b) Tuning time

Figure 6: Access and tuning times vs. node bit-vector size

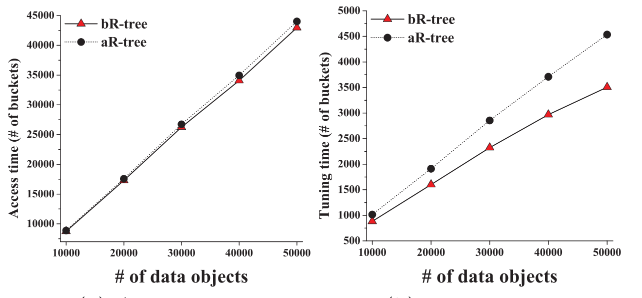
As depicted in Figure 6(a), the access time gradually increases when the size of $E.bv_i$ increases, due to the enlarged size of the bR-tree, which negatively affects the access time performance. Regarding the tuning time performance, the search algorithm on the broadcast bR-tree achieves the best performance when the size of $E.bv_i$ is set to 3 bytes, as plotted in Figure 6(b). Although the small size of $E.bv_i$ reduces the height of the bR-tree, the algorithm needs to access more unnecessary nodes due to the lack of the non-spatial information. On the other hand, the large size of $E.bv_i$ provides the abundance of the non-spatial information to the algorithm but leads to the height growth of the bR-tree. Since the bR-tree method achieves a good tradeoff between the access time performance and the tuning time performance when the size of $E.bv_i$ is set to 3 bytes, we chose to use this value in the remainder of our simulation experiments. As with the aR-tree, we allocated 2 bytes and 16 bytes to the pointer and the spatial information, respectively, of each non-leaf entry of the bR-tree.

4.2 Experimental Results

In the first experiment, we studied the effect of the number of data objects on the performance of the aR-tree and the bR-tree methods. As illustrated in Figure 7, the access and tuning times of the both methods increase as the number of data objects increases. This is due to the fact that as the number of data objects becomes lager, the size of the aR-tree and the bR-tree increases and more nodes of the aR-tree/bR-tree have to be accessed. However, the bR-tree method outperforms the aR-tree method in terms of the access and tuning times, due to the smaller size (height) of the bR-tree than that of the aR-tree. In addition, since the bR-tree method achieves better pruning capability compared with the aR-tree method, it accesses a smaller number

[5]Similarly, the *minscore* and the *maxscore* defined for the aR-tree can be easily extended for considering 3 non-spatial attributes.

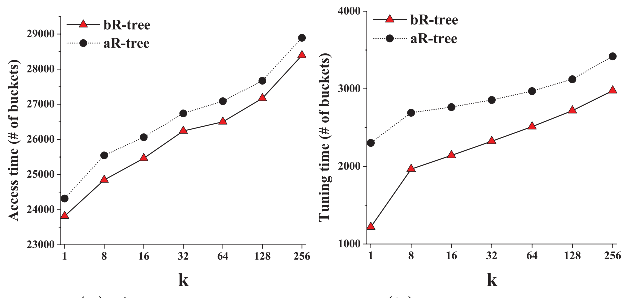
of nodes than the aR-tree method. In comparison with the aR-tree method, the bR-tree method requires 97.6% of the access time and consumes 82.2% of the tuning time on the average.



(a) Access time (b) Tuning time

Figure 7: Access and tuning times vs. # of data objects

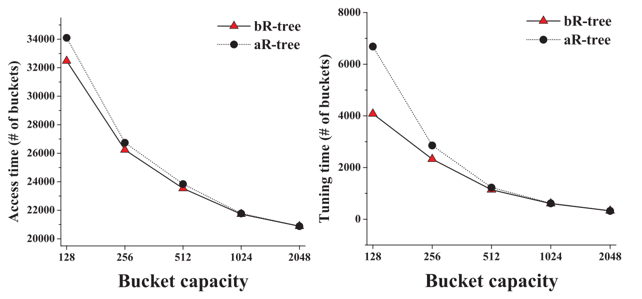
Figure 8 demonstrates the impact of the value of k on the access and tuning times of the proposed methods. Again, the bR-tree method performs better than the aR-tree method due to the same reason mentioned in the first experiment. On the average, the bR-tree method incurs 97.4% of the access time of the aR-tree method, and it consumes 77.4% of the tuning time of the aR-tree method.



(a) Access time (b) Tuning time

Figure 8: Access and tuning times vs. k

Finally, we varied the bucket capacity and studied the performance of the proposed methods. As plotted in Figure 9, the access and tuning times of the both methods decrease as the bucket capacity increases since the larger capacity of bucket reduces more the size (height) of the aR-tree/bR-tree. However, as expected, the bR-tree method performs better than the aR-tree method in all the cases, in terms of the access and tuning times.



(a) Access time (b) Tuning time

Figure 9: Access and tuning times vs. bucket capacity

5. CONCLUSIONS

In this paper, we addressed the problem of processing location based queries in the wireless broadcasting system. In particular, we presented a location based top-k query whose primary goal is to find the best k data objects, determined according to the mobile user's specifications on the spatial and the non-spatial attributes of data objects. To process location based top-k queries on the broadcast stream, we first proposed a naïve method, namely the aR-tree method. Then, we presented the bR-tree, where each non-leaf entry augments the bit-vector information generated to effectively describe the non-spatial attribute values of the data objects in its subtree. We also discussed the search algorithm on the broadcast bR-tree. We carried out simulation experiments and demonstrated that the bR-tree method outperforms the aR-tree method, validating the effectiveness of the bR-tree for processing location based top-k queries on the broadcast stream.

6. ACKNOWLEDGMENTS

This work was supported by the Korea Research Foundation Grant funded by the Korea Government (KRF-2008-313-D00856).

7. REFERENCES

[1] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching,"*In SIGMOD*, 1984.
[2] B. Gedik, A. Singh, and L. Liu, "Energy Efficient Exact kNN Search in Wireless Broadcast Environments," *In ACM-GIS*, 2004.
[3] B. Zheng, W.-C. Lee and D. Lee, "Spatial Queries in Wireless Broadcast Systems," *Wireless Network*, 10(6), 2004.
[4] B. Zheng, W.-C. Lee, and D. Lee, "Search k Nearest Neighbors on Air," *In MDM*, 2003.
[5] D. Lee, W.-C. Lee, J. Xu, and B. Zheng, "Data Management in Location-Dependent Information Services: Challenges and Issues," *Pervasive Computing*, 1(3), 2002.
[6] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, "Efficient OLAP Operations in Spatial Data Warehouses," *In SSTD*, 2001.
[7] G.R. Hjaltason, H. Samet, "Distance Browsing in Spatial Databases," *ACM Trans. Database Systems*, 24(2), 1999.
[8] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," *In ICDE*, 2008.
[9] I. Lazaridis and S. Mehrotra, "Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure," *In SIGMOD*, 2001.
[10] K.L. Cheung, A. Fu, "Enhanced Nearest Neighbour Search on the R-tree," *SIGMOD Record*, 27(3), 1998.
[11] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," *In SIGMOD*, 1990.
[12] N. Roussopoulos, S. Kelley, and F.Vincent, "Nearest Neighbor Queries," *In SIGMOD*, 1995.
[13] S.T. Leutenegger, J.M. Edgington and M.A. Lopez, "STR: A Simple and Efficient Algorithm for R-tree Packing," *In ICDE*, 1997.
[14] T.Imielinski, S.Viswanathan and B.R.Bardrinath, "Data on Air: Organization and Access," *IEEE Trans. Knowledge and Data Engineering*, 9(3), 1997.
[15] 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.
[16] W.-C. Lee and B. Zheng, "DSI: A fully distributed spatial index for location-based wireless broadcast services," *In ICDCS*, 2005.