Study on Location-Dependent Queries in Broadcasting Environment

Lien-Fa Lin*

Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, R.O.C.
E-mail: lienfa@cc.kyu.edu.tw

Abstract

Location-based services (LBSs) provide information based on location information specified in a query. Queries that support LBSs are called location-dependent queries (LDQ). In this paper, the issues involved with organizing location dependent data and answering LDQ queries on the air are investigated. An efficient data organization, called Jump Rdnn-tree, and the corresponding search algorithms are proposed. The performance of the proposed Jump Rdnn-tree and other traditional indexes (enhanced for wireless broadcasting) is evaluated using both uniform and skew data. The results show that Jump Rdnn-tree substantially outperforms the traditional indexes.

Keyword: location-dependent services, data broadcast, energy-conserving, mobile computing.

1. Introduction

Owing to the popularity of personal digital devices and advances in wireless communication technologies, location-based services (LBSs) have received a lot of attention from both of the industrial and academic communities.

NN and KNN queries return a specific number of data objects in the order of distance between a given query point and the data object. Researches have been made on NN and KNN in the traditional wired, disk-based client-server environment are abundant and successful, representing important types of query in LBSs. However, little studies have been made on wireless broadcasting environment except for NN [8], and linear property of wireless broadcasting media and power conservation requirement of mobile devices make the problem particularly interesting and challenging.

LBSs have several types of query, including NN, KNN, Window Query, and RNN. We can find some literature about broadcasting environment using on air index technology, and these methods can be divided into object-based index and solution-based index. R-tree is currently the most popular object-based index method. This index structure is capable of processing all types of LDQ, but it requires back-tracking technology to accelerate query processing, if it is applied to NN, KNN or RNN query. However, back-tracking technology does not fit in broadcasting environment where only linear access is available. Solution-based index does not have the problem of back-tracking, and it is used to process specific types of queries, such as 1NN, 2NN, etc. If LDQ processes different types of queries simultaneously, then solution-based index must build separate indexes for query types. Supporting many types of query requires broadcasting different types of solution-based index simultaneously, and it is not a very efficient method.

The limited battery capacity of mobile devices makes power become a critical issue in wireless environment. Therefore, there is much literature dedicated to general query processing on mobile devices with effective power management [1,2,3,4]. From these studies we have deduced some principles for designing a good on air index. We use these principles to design an on air index method that allow simultaneous support multiple types of LDQ queries, including NN, KNN, and RNN.

The rest of paper is organized as follows. Section 2 is an overview of related work. In Section 3, we propose several principles for designing an effective on air index. The details of Jump Rdnn-tree index structure are introduced in Section 4. Finally, we summarize the paper and describe our future work.

2. Related work

* Lien-Fa Lin is also a lecturer at the Department of Information Communication Kao Yuan University, No.1821, JhongShan Rd, Lujhu Township, Kaohsiung Country 821, Taiwan, R.O.C.
In this section, we shall introduce LDQ query, and research topics that relate to on air index and LDQ query in broadcasting environment in the following subsections.

### 2.1. Location-Dependent Query

In this paper we concern the three different types of LDQ queries, namely NN, KNN, and RNN, which are explained separately below. KNN returns a number of objects \((k)\) that have the shortest distances in relation to a given query \((q)\); if \(k=1\), then it is a NN query. KNN is widely applied to pattern recognition, image processing, computer aided design, and multimedia indexing. RNN query returns a collection of nearest neighbor objects \((q)\) from a given collection of objects \((S)\). Applications of RNN query is quite comprehensive, including decision making support system, market strategy, database archive, bio information, etc. Much literature [5,7] has mentioned the RNN query application.

A straightforward solution to computing reverse nearest neighbor (RNN) queries is to check for each point whether it has a given query point as its nearest neighbor. However, this approach is unacceptable when the number of points is large. Therefore, the general approach is using RNN-tree, a special type of R-tree to handle the RNN problem. Yang and Lin [7] have improved the method proposed by Korn and Muthukrishnan [5] and proposed Rdnn-tree structure, which may simultaneously process problems of NN and RNN. The difference between Rdnn-tree and R-tree is that Rdnn-tree stores NN information of every related object (such as distance of nearest neighbor) and therefore allows more efficient processing of RNN problems.

### 2.2. On Air Index

Because linear access is not considered in the design of traditional index structure, the algorithm that is currently adopted in disk-based spatial index can not satisfy the need of effective power conservation. Shown in Figure 1 is R-tree index; its broadcasting sequence is root, \(R_1\), and \(R_2\). The visit sequence for searching for NN with a given query point of \(q_2\) is :Root is first visited because the distance between \(q_2\) and \(R_2\) is shorter than that to \(R_1\). Therefore, \(R_1\) is skipped and \(R_2\) is visited first. However, the shortest object to \(q_2\) is \(O_3\) of \(R_1\) in MBR, and therefore \(R_1\) must be first visited. However, at this time \(R_1\) has just been broadcasted and it can only be accessed in the next broadcasting cycle.

With the feature of linear access in broadcasting environment, if the broadcasting sequence differs from the sequence of query, then long access latency will occur. Therefore, branch-and-bound query method in broadcasting environment is not a very effective method in term of access latency. An alternative, is direct access to MBRs sequentially. However, this method will cause unnecessary traversal of MBRs, and index search performance will not be optimized. For example, the search of NN for \(q_1\), the real NN is \(O_4\) of MBR \(R_2\), and accessing to \(R_1\) is obviously a waste of resource. Therefore, a new index method must be designed for wireless broadcasting environment to effectively adopt the feature of linear access in broadcasting environment and satisfy the need of power conservation for mobile devices.

![Example of R-tree Indexing](image)

**Figure 1. Example of R-tree Indexing**

### 3. Effective Broadcast Index Design

Access latency of accessing to data and tuning time that a mobile device requires in active mode are the two benchmarks for broadcast index efficiency measurements. Broadcast index is mixed with broadcast data and sent out together, and MU receives data in the following three steps [6]: (1) Initial probe: during any point in time of broadcasting, a user tunes into a broadcast channel and wait for the index data to be broadcasted. This period of time is called initial probe waiting. (2) Index search: When index data arrives, a user receives the index data, selectively accesses some index data according to his/her needs, and finds the location of the requested data. (3) Data retrieval: When the requested data arrives, a user downloads and accesses to the data. The time required for these three steps shall influence broadcast index efficiency. Therefore, a design of effective broadcast index must reduce the time required for these three steps.

**Reduction of initial probe time:** Initial probe waiting is the time that a user waits for index data. By duplicating multiple indexes in the entire broadcast cycle, the possibility of index appearing may increase, and the initial probe waiting time can be reduced. Imielinski et al. [3] used interleaving method to
duplicate m copies of index data in order to reduce initial probe waiting time.

**Reduction of index data size:** Index searching time is related to the size of index data; the smaller the index data size is, the shorter the search time will be. Consequently, the entire broadcast cycle will be shorter, and the average access latency will be smaller. For example, Imielinski et al. [3] only duplicated k layers of index tree to reduce the size of index data. Hu et al. [2] used the signature capture technique to reduce index data size.

**Effective data placement:** Chen et al. [1] has proved that different broadcasting sequence of different data would affect average access latency of data retrieval, and proposed ORD algorithm to reduce average access latency of data retrieval. Jianting and Le Gruebwald [4] proposed to reduce access latency by arrangement of the sequence of broadcast data according to retrieval frequency. Currently broadcast index studies focus on one single step to enhance efficiency without considering improving the efficiencies of the three steps. This Study has designed a new broadcast index to handle RNN query in broadcasting with considerations for the three steps.

### 4. A New Index Structure for LDQ Query

The design of a good broadcast index as mentioned in Section 3 includes three steps: reducing initial probe time, reducing size of index data, and effective placement of broadcast data. We shall explain how we improve Rdnm-tree with these three steps.

**Reduction of initial probe time:** The traditional approach is to increase the possibility of the appearance of index by duplicating index. However, this approach will cause longer broadcast cycle and longer average data access latency. Our approach is to build a Jump Rdnm-tree with our index structure for broadcast data. Data and index will be mixed together and broadcasted based on every sub-tree. After the index of such sub-tree has been broadcasted, the data under the sub-tree will be broadcasted in order to reduce the distance between data and index instead of broadcasting all data after the index broadcasting is completed. Taking Figure 2 as example, the broadcasting sequence is B, B1, b1, a, b, c, d, b2, e, f, g, B2, b3, h, i, b4, j, k, and l. The cyclic index structure is also adopted. The location of each index node to be visited next is calculated with Depth First Search (DFS) traversal according to depth priority. This approach is also called jump pointer, as shown in Figure 2. Because every index node has a jump pointer, index trees are interlined through jump pointers. Therefore, index tree search may begin from any index node instead of root node. In the same time, index tree search follows pointer sequence, and there will be no back-tracking problem.

**Reduction of index data size:** As mentioned earlier, we adopt cyclic index structure, replication of index data for the entire broadcast cycle is not required and only one copy of index data is needed. Therefore, the size of index data is very small. Also, if the largest sub-tree at every layer of the structure of a traditional index tree has f fan-out that represents the index tree needs f pointers are needed to record the address of every sub-tree. Because of linear access feature of wireless broadcasting environment, MU can only access data in active mode, or skip the current data in doze mode and wait for the next data access in active mode. That is, only two behaviors are available: Next, which proceeds to next action, and Jump, which skips data retrieving. If the criteria for Jump are not met, then Next takes place and Next behavior does not have to be recorded. Therefore, any search for non-leaf node of a tree requires only keeping one jump pointer, and other f-1 pointers for sub-trees can be eliminated, leading to reduction of index data size.

**Effective data placement:** Query point is produced based on the entire search space. The possibility of the occurrence in every area shall affect searching efficiency. If the corresponding index data for the area where the possibility of the occurrence of query point is higher are broadcasted earlier, then the index search efficiency will be better and if the location of the query point is in uniform distribution. Therefore, the broadcast sequence of a sub-tree is determined according to the area of MBR where the sub-tree is, because the larger the area of MBR is, the higher the possibility of the query occurrence will be.

After Rdnm-tree is improved by the above-mentioned approach, the problem of back-tracking is eliminated, and Rdnm index tree with cyclic structure fits with linear access feature of broadcast better. The detailed algorithm of RNN queries of cyclic broadcast is illustrated in Algorithm 1: RNN-Search-On-Air. D(q, ptid) represents the distance between query point q and objects ptid, wile D(q, rect) represents the distance
between query point \( q \) and rectangle \( \text{rect} \). The detailed algorithms of KNN and RNN queries of cyclic broadcast are illustrated in Algorithm 1, KNN-Search-On-Air and Algorithm 2, RNN-Search-On-Air. \( D(q, \text{ptid}) \) represents the distance between query point \( q \) and objects \( \text{ptid} \), while \( D(q, \text{rect}) \) represents the distance between query point \( q \) and rectangle \( \text{rect} \).

**KNN-Search-On-Air (Node \( n \), Point \( q \))**

**Step 1:** \( K \) hotspots selected as the candidates of nearest neighbor are placed in K-Buffer to calculate the distance between \( q \) and every point, and then sorted from small to large. Top-Of-K-Buffer is selected as pruning distance.

**Step 2:** The value of Non-Found is \( K \), and the packet of the first internal node is the Start-Packet.

**Step 3:** If \( \text{Node} = \text{Start-Packet} \) or Non-Found = 0, then end.

**Step 4:**

Case 1: If \( n \) is the coordinate of data object, check every coordinate \((x, y)\) in \( n \).

If \( D((x, y), q) < \) pruning-distance, then \( D((x, y), q) \) will replace the point of pruning distance and the sequence of K-Buffer will be replaced. Top-Of-K-Buffer will be the new pruning-distances.

If \( D((x, y), q) < D_{nn}/2 \), then the value of Non-Found = 1; if Non-Found = 0, then end.

Case 2: If \( n \) is internal node, then the distances between child node and \( q \) will be sorted from small to large in order to determine the distances between every child node and \( q \).

If \( D(q, \text{MBR}) \) is smaller than pruning-distance, then call KNN-Search-On-Air (JumpPtr, \( q \)).

**Algorithm 1. KNN Search-On-Air**

**RNN-Search-On-Air (Node \( n \), Point \( q \))**

Case 1: If \( n \) is leaf node then

For all (data-item, dnn) in \( n \)

If \( D(q, \text{ptid}) < \) dnn, then data-item is the RNN for \( q \).

Case 2: If \( n \) is non-leaf node, then

For all branch \( B = (\text{ptr}, \text{MBR}, \text{Maxdnn}) \) in \( n \)

If \( D(q, \text{rect}) < \) MaxDnn, then call RNN-Search-On-Air (B.ptr, \( q \)).

**Algorithm 2. RNN Search on Air**

**5. Conclusion and Future Work**

LBSs are an important and practical application in the domain of mobile computing. In this paper we have discussed how to effectively organize and deploy data in wireless broadcasting environment and answered questions about many types of LDQ. The results show that our approach is better than currently adopted Rdnn-Tree Air indexing that is improved based on R-tree to accommodate linear access feature of broadcasting for LDQ query. In the future, we shall further extend to more advanced and more dynamic location-dependent queries.

**6. References**


