Search K Nearest Neighbors on Air*

Baihua Zheng[1], Wang-Chien Lee[2], and Dik Lun Lee[1]

[1] Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{baihua,dlee}@cs.ust.hk
[2] The Penn State University, University Park, PA 16802
wlee@cse.psu.edu

Abstract. While the K-Nearest-Neighbor (KNN) problem is well studied in the traditional wired, disk-based client-server environment, it has not been tackled in a wireless broadcast environment. In this paper, the problem of organizing location dependent data and answering KNN queries on air are investigated. The linear property of wireless broadcast media and power conserving requirement of mobile devices make this problem particularly interesting and challenging. An efficient data organization, called *sorted list*, and the corresponding search algorithm are proposed and compared with the well-known spatial index, *R-Tree*. In addition, we develop an approximate search scope to guide the search at the very beginning of the search process and a learning algorithm to adapt the search scope during the search to improve energy and access efficiency. Simulation based performance evaluation is conducted to compare sorted list and R-Tree. The results show that the utilization of search scope and learning algorithm improves search efficiency of both index mechanisms significantly. While R-Tree is more power efficient when a large number of nearest neighbors is requested, the sorted list has better access efficiency and less power consumption when the number of nearest neighbors is small.

Keywords: KNN, location-dependent search, wireless broadcast, index structure, mobile computing.

1 Introduction

The advance of wireless technologies and mobile devices allows users to obtain information anywhere and anytime. Location-dependent information services (LDISs) are services that return results based on some location information. Examples of LDISs include returning local traffic reports and the nearest restaurants with respect to the user's current location.

A K-Nearest-Neighbor (KNN) query returns a specific number of data objects, sorted by their distances from a given position. The KNN problem, well

* Research supported by Research Grants Council, Hong Kong SAR, China under grant number HKUST6079/01E and AoE/E-01/99.

studied in the traditional disk-based, client-server computing environment, represents a very important class of queries in LDISs [1, 4, 7]. With the anticipated dramatic increase of the mobile user population, the scalability of LDISs will be a major challenge. A wireless broadcast system capable of answering KNN queries is considered a promising solution because it can serve a virtually unlimited number of users within its coverage.

Although studies on wireless broadcast alone and KNN problems in the disk-based environment are well documented in the literature, this paper, to the best of our knowledge, is the first attempt to address the KNN problem in a wireless broadcast environment. The linear property of wireless broadcast media and power conserving requirement of mobile devices make the problem particularly interesting and challenging. Considering the specific features of wireless broadcast environments, a new data orgranization scheme and a new index structure are proposed, along with several related algorithms. The main contributions of this paper are three-folded: 1) proposed a simple but efficient index structure to support linear transmission of location dependent data and processing of KNN queries; 2) developed a search radius estimation function to provide approximated search scope for KNN query processing; 3) developed a learning algorithm to dynamically adapt the search scope based on objects' distribution and the ratio of requested k.

The rest of this paper is organized as follows. Section 2 provides a brief overview of related work. A search radius estimation function is introduced in Section 3 to provide an initial search scope, along with a learning algorithm devised to dynamically adapt the scope according to the real situation. Section 4 explains the detailed index structure proposed in this paper. Performance evaluation results are shown in Section 5. Finally, Section 6 concludes this paper and discusses the future work.

2 Background and Related Work

Related work on KNN search and wireless data broadcast are reviewed, respectively, in the next two sections. Then, we discuss the problem and revision of R-Tree for broadcasting in a wireless channel.

2.1 K-Nearest Neighbor Search

K-Nearest Neighbor (KNN) search returns a specified number of data objects, sorted by their distances from a given query point. KNN has been addressed mostly in the context of spatial databases, though its applications can also be found in pattern recognition, image processing, CAD, and multimedia indexing [2, 8].

With a large candidates set, answering KNN via scanning through the whole set becomes extremely expensive. Index structures and related search algorithms have been proposed to provide efficient processing of KNN queries. The main idea is to use heuristics to detect and filter unqualified paths, thus reducing the search cost. Existing algorithms for KNN queries can be divided into two categories based on how the candidates set is scanned, namely, single-step search and multi-step search.

Single-Step Search With the support of index structures, algorithms in this category search for KNN by scanning the candidates only once. There are several methods documented in the literature. Branch-and-bound algorithms, e.g., R-Tree, use heuristic distances to choose the next node for visiting and pruning. Various algorithms differ in the search order and the heuristics used to prune branches [4, 10]. Incremental algorithms report the objects one by one to allow the algorithm to operate in a pipelined fashion. They are especially suitable for complex queries involving proximity [3].

Multi-Step Search Methods in this category scan the candidates multiple times until the proper answers are obtained. Korn, et. al. proposed an adapted algorithm [8]. First, a set of k primary candidates was selected based on stored statistics to obtain the upper bound d_{max} which can guarantee that there are at least k sites within the distance d_{max} from the query point q. Next, a range query was executed on the site set to retrieve the final candidates. An extended version of this algorithm was proposed in [11], in which d_{max} was adapted every time a candidate object was checked.

2.2 Wireless Data Broadcast

Generally speaking, there are two approaches for mobile access of location-dependent data. For a **on-demand access** mode, the server locates the appropriate data and then returns the answer according to the query submitted by a client, in which data is transferred in a point-to-point connection. For a **broadcast** mode, server broadcasts information in the wireless channels periodically and the client is responsible for filtering its desirable data.

Compared to on-demand access, a major advantage of broadcast is that it allows simultaneous access by an arbitrary number of mobile users. Disseminating information via broadcast is a very efficient and scalable method for a large client population. LDISs must anticipate demands from a large number of mobile users. It is envisaged that many LDISs such as region-wide tourism information will utilize broadcast for the dissemination of information to the rapidly increasing population of mobile users. The focus of this paper is to investigate the feasibility of answering KNN queries on a wireless data broadcast environment.

Power conservation is a critical issue for mobile devices. In the wireless broadcast environment, power consumption can be reduced by interleaving auxiliary index with data [6]. By looking up the index, mobile devices are able to anticipate the arrivals of the desired data and stay in the doze mode until the data of interest arrives. To interleave the index and the data on the broadcast channel, we employ the $(1, m)$ interleaving technique [6]. That is, the whole index is replicated m times and broadcast before every $\frac{1}{m}$ fraction of the broadcast cycle. *Index search time* is frequently used to evaluate the effectiveness, in terms of power consumption, of an index and data organization method for broadcast channel. Thus, we use it along with *access time* as metrics for performance evaluation.

In wireless communication, the data stream is normally delivered in *packets* (or *frames*), to facilitate error-detecting, error-correction, and synchronization. Thus, both index search time and access latency are measured in terms of *number of packet accesses* [5, 6]. The drawback of air indexing is that broadcast cycles are lengthened due to the additional indexing information. Hence, the index size should be kept as small as possible.

2.3 Spatial Index on Air?

Indexes for KNN search have been well studied, however, sequential access feature of broadcast systems introduces new challenges. Original indexes are designed for random access storage (e.g., disks), therefore they may not work well in a wireless broadcast environment. Figure 1 and Figure 2 illustrate an example. Assume that a query looking for the nearest neighbor of point q_2. The query can be answered by scanning a R-Tree (see Figure 1) in the following order: first visits root node, then R_2 (which contains object o_2 and o_4), and finally R_1 (which contains object o_1 and o_3). This works well for random accessed disk index but renders a problem for air index: when a mobile device tries to retrieve R_1 after visiting R_2, it has to wait until the next broadcast cycle when R_1 is broadcast again (denoted by the second arc in Figure 2).

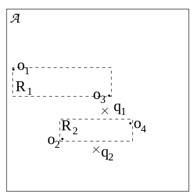


Fig. 1. Example MBR Structures

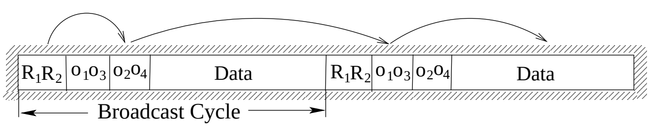


Fig. 2. Sequential Access In Wireless Broadcasts

As observed in the above example, single-step linear scan is a major requirement for any efficient algorithm in wireless broadcast environments. The R-Tree is broadcast in a depth-first order. To perform KNN search on broadcasted R-Tree, a mobile device maintains a buffer for the top-k objects. It sequentially traverses the R-Tree from the root and compares its maximal distance to objects in the current buffer, M, with its minimal distance to the next branch to visit, m. If $m > M$, then the incoming nodes bounded with this unqualified branch can be pruned (by turning the device into doze mode and wake up only when the subsequent branch arrives). Otherwise, the device stays awake and recursively traverses the branches. Since it has to pass through the whole index in order to access the searched objects, the index size has a great impact on access time.

3 Approximate Search Scope

Since requested parameter k, compared to the number of whole candidates set, is really small, some guidence exists to remove the impossible objects. An intuitive guidance is the distance of the kth nearest neighbor to the query point q that defines the radius of the necessary search space, named $radius_{needed}$. Since $radius_{needed}$ is dependent on both the position of the query and the parameter k, it is impossible to provide an exact value. In this work, a search radius estimation function is devised to approximate the search boundary within which the top k objects to a given query point are expected to be obtained. Based on that boundary, some unqualified candidates are detected at the very beginning of the search process. Although it is not guaranteed to be 100% accurate, the accuracy is in the limit approach 100%.

The objective of a search guidance is to facilitate intelligent filtering by trying to fix a search scope as early as possible so that objects outside the search scope will be disqualified. By assuming a uniform object distribution, a search radius approximation function is given in Eq. (1). Due to the space limitation, its detailed derivation is omitted and the complete proof can be found in [12]. Here, n is the number of objects, k is the requested number and c is the control constant factor to increase flexibility considering different paramters k and objects distributions.

$$r = c \times \ln(n) \times \sqrt{\frac{k}{(\pi \times n)}} \tag{1}$$

R-Tree can be modified with the help of our search radius estimation function to further reduce unnecessary traversal. No matter where the query is, only the MBRs within the expected search radius are accessed. Liu et al. proposed a similar approach in which one or more window queries were used to retrieve the KNN objects of a query point [9]. Two estimation methods were provided, using statistical knowledge based on density and bucket. However, their objective is to satisfy the KNN problem by applying 3 or 4 windows queries. This kind of multiple-scan introduces long tuning time and also a large access latency. Our algorithm tries to provide a relatively more accurate estimation in order to satisfy the request.

3.1 Learning Algorithm

In Equation (1), a constant c is introduced to increase the flexibility of the radius approximation. For a large value of k and n, the constant should be smaller than the small of k and n by intuition. Besides, the search radius is approximated based on the assumption that the objects are uniform distributed. Consequently, it is not surprising that the expected search radius is not accurate when the distribution is skewed. Parameter c in Equation (1) services for this purpose, and a learning algorithm is introduced to adapt it according to real situations dynamically.

Originally, the server assigns a static value for constant c and all the clients use the same default configuration. In broadcast environments, it is difficult for the server to obtain the feedback information about the accuracy of the approximated search radius. Consequently, the clients should adapt the setting of c based on their situations. The detailed description of the algorithm is provided in Algorithm 1.

Algorithm 1 Learning Algorithm of Constant Assignment

Input: requested k, SAT_k, objects number n, $CONS_k$;
Procedure:

1: **if** n >= k **then**
2: $\quad SAT_k.num++$; $SAT_k.value+=\frac{n}{k}$;
3: $\quad$ **if** $SAT_k.num ==$ Num **then**
4: $\quad\quad CONS_k$=$(1-\alpha)\times CONS_k + \alpha \times CONS_k \times SAT_k.num/SAT_k.value$;
5: $\quad\quad SAT_k.num = 0$; $SAT_k.value = 0$;
6: $\quad$ **end if**
7: **else**
8: $\quad cur\_accu = \frac{n}{k}$;
9: $\quad$ **if** $n == 0$ **then**
10: $\quad\quad cur\_accu = 0.01$;
11: $\quad$ **end if**
12: $\quad CONS_k = (1-\alpha)\times CONS_k + \alpha \times (CONS_k/cur\_accu)$;
13: $\quad SAT_k.num = 0$; $SAT_k.value = 0$;
14: **end if**

The basic idea is to adapt the search scope in accordance with the degree of contentment. For any specific k, if the number of objects currently returned is smaller than the requested one, the search scope should be increased accordingly. Here, we use an idea similar to the analogous aging function to obtain the access probability which is shown in line 12, with cur_accu denoting the current accuracy. If the search scope always contains adequate number of objects for some specific k, the search scope could be reduced. In our algorithm, a client can use an accumulator to keep the frequency that its requests are satisfied, which is reset to zero once its accuracy is not 100%. Also, the *satisfied degree*, which is defined as the ratio of the number of objects in the candidate set to the requested number k, is maintained. Once the client obtains Num times of accurate answers continuously, it means the current setting is stable enough and can be reduced correspondingly. Here, Num is predefined. Lines $2-5$ show the detailed action employed to reduce the sufficient search radius. Parameter α is the constant factor to weight the importance of the current setting of c.

4 A New Air Index for KNN Search

Based on the guidance of search radius, a simple index structure is devised. In the following, we explain the details.

4.1 Sorted List

In wireless environments, the dimension is usually low, i.e., two or three in accordance with the real world situation. Hence, sorted in each dimension, the objects can be represented by two or three sorted lists, each corresponding to one dimension. The following description is based on a two-dimensional space which is easy to be extended to a three dimensional space.

Represented by two lists, the objects in the first one are sorted in x-dimension. Each object is represented by its x-coordinate and the pointer pointing to its position in the second sorted list. In the second list, each object's y-coordinate and the related pointer pointing to the real data are recorded. In other words, the coordinates and pointer information together provide sufficient information to access the objects.

Given a query point q and the corresponding radius r, the objects to be examined should satisfy the condition: $x \in [q.x-r, q.x+r]$ and $y \in [q.y-r, q.y+r]$, where $(q.x,\ q.y)$ denotes the coordinates of the query point. By listening to the channel, a client can detect two sets of possible candidates that satisfy the above conditions and their intersection provides the candidate answer set. Then the top k objects are returned. In case there are only k' ($k' < k$) objects in the candidate set, this query is not satisfied and the ratio of k' to k is defined as the accuracy of the corresponding search radius. Although the accuracy of this algorithm is not guaranteed to be always 100%, no false answer is returned and the returned k' objects are guaranteed to be the top-most k' nearest neighbors.

4.2 Packing the Sorted List

In wireless environments, information are transmitted to the clients in the unit of *packet*.[3] Therefore, all the data has to be packed into packets.

Considering the broadcast of a sorted list, there are two kinds of information. In the lower level, the packets contain the objects position information and the related pointers to the data packets containing the real data of the objects. In the upper level, the packets contain the index information of the lower level packets for detecting the packets needed for query processing. Given a query

[3] Similar to the concept of *page* in traditional databases.

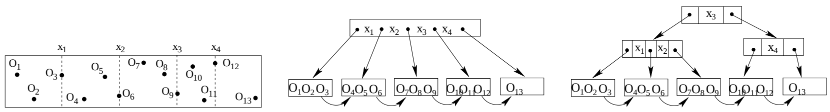
point q and a search radius r, $[q.x - r, q.x + r]$ defines the x-dimensional scope within which the objects should be checked. With the help of the upper level index, the packets containing the objects whose x-coordinates are within this scope can be obtained. Similar action is done for the y-dimensional sorted list. After downloading the upper level packets, the top k objects can be detected by examining the objects coordinates. Then the lower level packets provide the pointers to the packets containing the real data information of those objects.

(a) Original Sorted List

(b) Fixed Partition

Fig. 3. Fixed-Partition Sorted List for the Running Example

To pack the sorted list for broadcast, two different schemes, namely *fixed-partition sorted list* and *full-occupancy sorted list* are proposed, respectively. The first one partitions the sorted list by a fixed distance, while the other one tries to maximize usage of the packet capacity. In [12], a detailed theoretical analysis is provided to estimate the index search time of these paging schemes. It is omitted here due to space limitation.



(a) Original Sorted List

(b) No-Index

(c) Tree-Liked

Fig. 4. Full-Occupancy Sorted List for the Running Example

Fixed-Partition. As illustrated in Figure 3, the whole sorted list is divided by a fixed distance, dis. The fan-out of a packet is three for the example. There are several distance chosen metrics, such as the capacity of the packet. In this paper, a simple algorithm is used to choose the suitable fixed partition distance. Based on the capacity of a packet, the number of packets, denoted as num, required for packing the objects can be computed. The basic distance, denoted as b_dis, can be obtained by partitioning the whole list into num parts, i.e., $b\_dis = width(height)/num$. Based on b_dis, a possible range of dis (e.g., $[b\_dis/4, 4b\_dis]$ for our simulation) can be decided. With the help of the theoretical analysis of the index search time, greedy search can be employed to check the possible distance within the range. Thus, the distance that produces the index having the best search performance is obtained.

The advantage of this partition is to simplify the upper-level index. A client can locate the packets by hashing the search scope, i.e., $\lfloor (q.x - r)/dis \rfloor$ and $\lceil (q.x + r)/dis \rceil$, into the lower-level packets. However, due to the fixed partition distance and the packet's capacity, it is not unusual to have many packets with a low utilization rate. As shown in Figure 3, some packets only contain one object.

Full-Occupancy. This partition tries to maximize the utilization of the lower-level packets, enabling the packet to contain the maximal objects information, i.e., the packets in lower level are full-occupied. For the upper-level index, there are two alternatives. One is to store the whole information about the lower-level packets and no index is provided. The clients have to read all the upper-level index to locate the lower-level packets needed for filtering. The other one is to build a tree-liked index. Given the packet capacity, the number of lower-level packets can be obtained. Therefore, the search time in the upper-level index can also be approximated. The one providing more efficient search time is employed. Figure 4 shows an example, assuming each packet contains three objects. The advantage of full-occupancy sorted list is the high utilization of lower-level packets.

4.3 Discussion

For every deletion and insertion of objects, the corresponding index has to be updated. Since sorted list is used to represent the objects, the updated objects should be inserted or deleted from the lists. Considering *full-occupancy* sorted list, the lower-level packing is processed according to the packet capacity. For the *fixed-partition* sorted list, the fixed distance used to partition the objects in the lists needs not be changed when the number of updated objects is small compared to the total number of objects. However, its performance is affected and the re-construction of the index should be done later. The time for carrying out the re-construction of index information should be determined based on update cost and requirements of the applications.

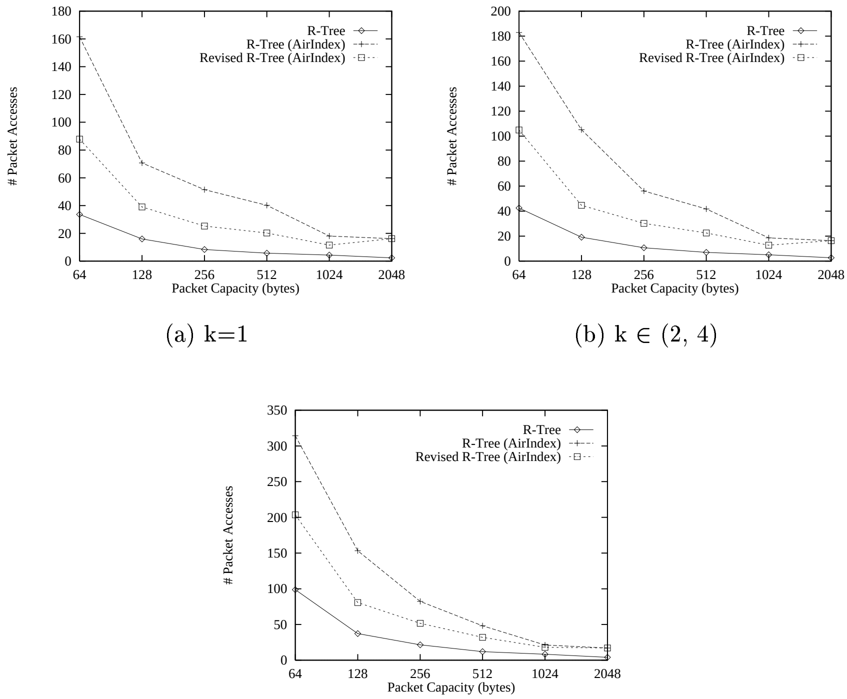
5 Performance Evaluation

Performance evaluation has been conducted to compare sorted list with the air version of R-Tree. Two datasets are used in the evaluation. In the first dataset (UNIFORM), we uniformly generate 5,000 points in a square Euclidean space. In the second dataset (SKEW), a skewed object distribution is generated as follows. First, the square space is equally divided into 5×5 subspaces. Then, we generate 10,000 objects, with the probabilities of falling in the subspaces follow a Zipf distribution and the skewness parameter is set to 1.2.

Based on our discussion in Section 2, R-Tree is revised for air indexing. Due to the linear property of wireless broadcast, the nodes of R-Tree are sequentially accessed while unqualified branches are pruned using the distance heuristics. In the following, *R-Tree (AirIndex)* denotes the revised scheme to cater for air indexing, which also combines the guidance provided by the search radius. While

R-Tree denotes the original algorithm devised for disk index, without any modification.

The system parameters are set as follows. In each packet, two bytes are allocated for the packet id. Two bytes are used for a pointer and four bytes are for a co-ordinate. The packet capacity is varied from 64 bytes to 2048 bytes. Queries are produced randomly in the search space, and the results are obtained by the final statistics of about 1,000,000 queries. For the parameter of k, three different settings are provided, $k = 1$, $k \in [2, 4]$ and $k \in [21, 30]$. Due to the space limitation, only partial results are depicted. Observation from the rest of results will be summarized in text.



(a) k=1

(b) k ∈ (2, 4)

(c) k ∈ (21, 30)

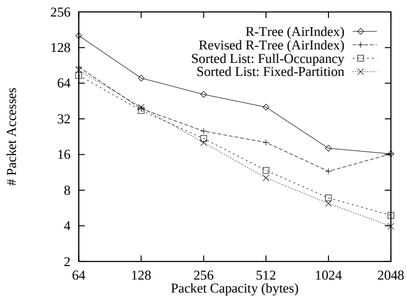
Fig. 5. Improvement Brought by Search Boundary of UNIFORM Dataset

5.1 Improvement brought by the Search Scopes

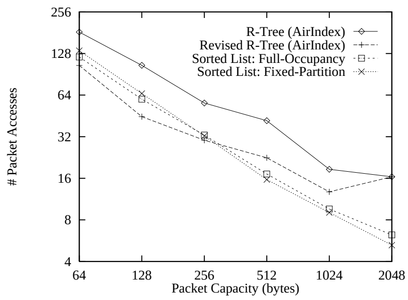
Search scope is shown to be useful in pruning the search space and thus improving performance. Some experiments are done to study the accuracy of the search scope estimation and the accuracy is close to 100% as the value of c increases. Parameter c is set to 0.5 by default in the later simulations. In the following, we show that it is also useful for branch-and-bound algorithms. A revised R-Tree which adopts the search scope is compared to the original R-Tree.

As shown in Figure 5, the number of packets accessed in air-index is extended distinctively, compared to the performance in disk-index environments. The reason is the sequential access property of air index. When applying the approximated search scope, the performance is improved significantly. Considering the UNIFORM data, the improvement achieved by the revised algorithm over the original algorithm is 37.9%, 37.3% and 28.0% for the three different situations, respectively. For the SKEW dataset, the improvement is about 19.3%, 19.8% and 19.2%, respectively. Of course, the revised algorithm sometimes cannot satisfy a KNN request, while the original algorithm always returns the exact answers. However, for the simulations shown, the accuracy is 100% in all cases since the requested k is relatively small compared to the total number of objects.

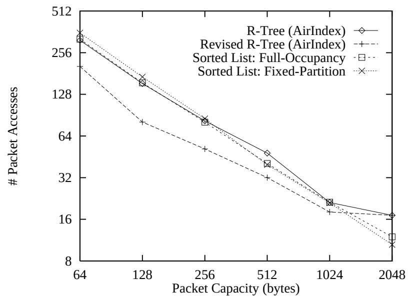
In summary, the performance of R-Tree is decreased by the specific properties of air indexing. Combining the search scope guidance, the tuning time is significantly improved.



(a) k=1



(b) k ∈ (2, 4)



(c) k ∈ (21, 30)

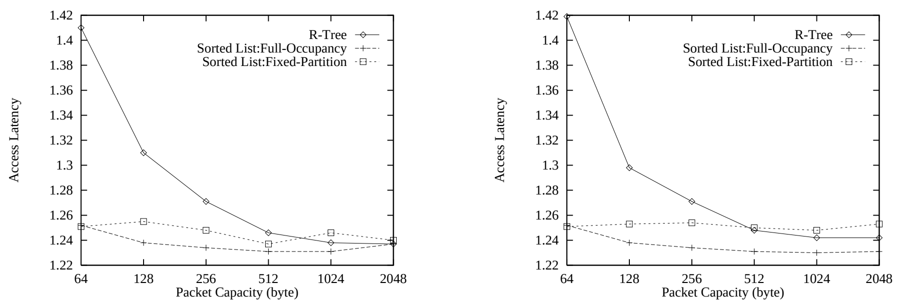
Fig. 6. Index Search Time for UNIFORM Dataset

5.2 Performance of Sorted List Index

In this section, we compare the performance of sorted list and R-Tree air indexes, in terms of average number of packet accesses, and access latency. Since we are mainly interested in supporting KNN search in broadcast, the performance of the original branch-and-bound algorithm in traditional disk index environments is omitted. In the following simulations, the accuracy of the indexing schemes is 100% accurate, according to the simulation results.

As discussed before, in wireless broadcast, improving the index search time saves power and connection cost. Figure 6 shows the index search time for the various indexes of the UNIFORM dataset. From the result, we found that the sorted list indexes improve performance when the value of k is relatively small compared to the population of objects. When k is set to one in the UNIFORM dataset, the *Fixed Sorted List* outperforms *Revised R-Tree* for about 33.7% in average. When k is increased to between two and four, the performance improvement only occurs for the large packet capacity. When the packet capacity is smaller than 256, the performance of the *Fixed Sorted List* is worse than the revised R-Tree for about 27.5%. When the packet capacity becomes larger, the *Fixed Sorted List* can provide a better performance, about 42.5% better than the one based on R-Tree. Considering the SKEW dataset, the improvement is more significant. It is 43.0% when k is one and 22.1% when k is between two and four. However, when k becomes larger, the revised algorithm based on R-Tree works best, about 29.0% better than the *Fixed Sorted List* one.

Considering two different packing schemes of sorted lists, their performance is decided by the object distribution. For UNIFORM distribution, *fixed-partition* can provide better service in most situations. While the distribution is clustered, its performance is reduced. For the SKEW dataset, *full-occupancy* scheme outperforms *fixed-partition* scheme by about 4.5%, 9.1% and 11.0% for the three different settings of k.



(a) UNIFORM (b) SKEW

Fig. 7. Access Latency for Two Datasets

By observing the simulation results, the performance of sorted list indexes is sensitive to the ratio of k over the total number of objects. When the ratio is really small, *Sorted List* can provide better performance than the existing ones. When the ratio becomes larger, the approximated search radius is also larger. Therefore, more objects have to be checked and the search radius somehow loses its power of providing accurate guidance. Since it only considers the distance in one dimension, the larger the search scope is, the more the false objects checked. Consequently, the performance becomes worse. However, considering real applications in mobile computing environments, the clients usually have interest in a small k. Therefore, the sorted list index structure is expected to be the first choice in many situations, in terms of efficient tuning time, along with a much smaller variance.

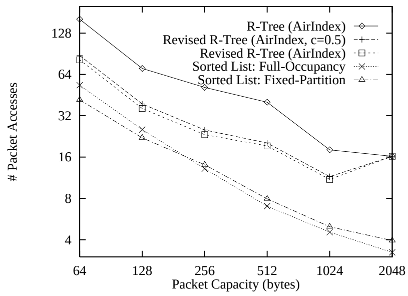
The access latency is affected by the index size. The larger the index size, the longer the access latency. Figure 7 shows the access latency for the index methods. The optimal value of m depends on the index size and is calculated for each index structure separately based on the technique presented in [6]. In the figures, the latency is normalized by the expected access latency without any index (i.e., half of the time needed to broadcast the database). It is obvious that R-Tree always performs the worst due to its large index size overhead. Thus, it is safe to conclude that the index overhead for the sorted list indexes is maintained at an acceptable level, and only introduces a limited variation to the access latency.

Comparing the two packing schemes for the sorted list index, *fixed-partition* incurs more size overhead for nearly all the settings due to the poor utilization ratio of packets. Therefore its expected access latency is also larger.

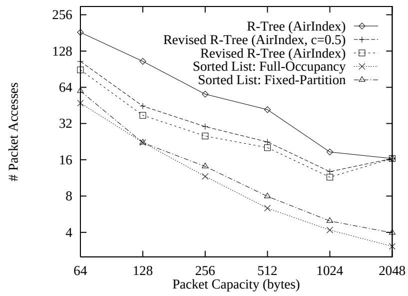
5.3 Performance of Learning Algorithm

The impact of *learning algorithm* on the performance is examined in this section. In our simulation, the parameter α is set to 0.10, 0.15 and 0.25 for $k = 1$, $k \in [2, 4]$ and $k \in [21, 30]$, respectively. The reason for different settings of α is motivated by the intuition from different values of k. We do not consider the relationship between the value of α, k, and n, which deserves further studies. Figure 8 shows the simulation result. The related accuracy result of the simulation results has not been shown, while it is almost 100% for the first two settings of k and about 99% for the last one.

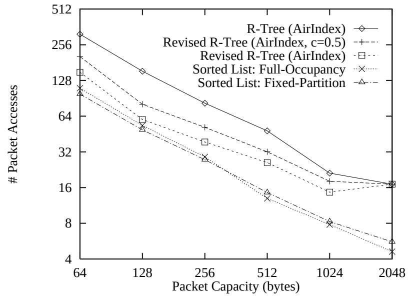
From the simulation results, fixing the value of c is not suitable for different values of k. Thus, dynamic assignment of c is more reasonable and improved the performance significantly. There are several observations obtained. First, the improvement is reduced for the SKEW dataset, compared to the UNIFORM one, since the statistical knowledge, derived based on the uniform distribution assumption, can not provide accurate guidance. Second, the improvement becomes more distinct as k becomes larger. The reason is that originally we assign the same value to c, while in real case the larger k is, the smaller c value is needed.



(a) k=1



(b) k $\in$ (2,4)



(c) k $\in$ (21,30)

Fig. 8. Performance Comparison of Learning Algorithm for UNIFORM Dataset

6 Conclusion

KNN search is a very important and practical application in mobile computing. Although it has been well studied in the traditional disk-based, client-server computing environments, more research is needed for the wireless broadcast platform. In this paper, we address the issues involved with organizing location dependent data and answering KNN queries on air.

To reduce the search space, a guidance is provided based on the approximation of the necessary search scope for any given k. With the guidance, existing algorithms can be revised to satisfy the specific properties of air indexing. A simple index structure, *sorted list*, is proposed to take advantage of the search guidance. Two packing algorithms for the index are also proposed. Besides, we use a learning algorithm to provide dynamic adaption of the search radius, according to the real situation.

Performance evaluation shows that the search scope can provide valuable guidance for processing KNN queries. Applying the search scope to R-Tree also reduces its search time and variance. For sorted list, it improves the performance in terms of access latency, page accesses and also the variance significantly, especially when the ratio of k to the total number of objects is relatively small. When the ratio becomes larger, the strength of sorted lists is reduced.

As for future work, we plan to continue the study of KNN on air problem since currently the search scope can only provide suitable guidance but not guarantee a return of exact k objects. We also plan to develop new solutions that take into account various object distributions to provide efficient answers for KNN queries.

References

1. S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 397–410, 1999.
2. H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proceedings of the 17th IEEE International Conference on Data Engineering (ICDE'01)*, April 2001.
3. G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases (SSD'95)*, pages 83–95, 1995.
4. Gí. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, 1999.
5. Q. L. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'2000)*, pages 157–166, San Diego, CA, USA, February 2000.
6. T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air - organization and access. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 9(3), May-June 1997.
7. N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Tucson, AZ, May 1997.
8. F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB'96)*, pages 215–226, 1996.
9. D-Z. Liu, E. Lim, and W. Ng. Efficient k nearest neighbor queries on remote spatial databases using range estimation. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management(SSDBM'02)*, Edinburgh, Scotland, 2002.
10. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (Sigmod'95)*, pages 71–79, May 1995.
11. T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (Sigmod'98)*, pages 154–165, July 1998.
12. B. Zheng, W. C. Lee, and D. L. Lee. K-nearest neighbor queries in wireless broadcasting environments. Technical report, Dept. of Computer Science, Hong Kong Univ. of Science and Technology, July. 2002.