Exploring the *k*NN Search on Broadcast Multi-dimensional Index Trees*

Shu-Yu Fu and Chuan-Ming Liu
Computer Science and Information Engineering
National Taipei University of Technology
Taipei 106, TAIWAN
Tel: +886-2-27712171 ext 4251
{bobby, cmliu}@mcse.csie.ntut.edu.tw

ABSTRACT

Data broadcasting provides an effective way to disseminate information in the wireless mobile environment using a broadcast channel. How to provide the service of the k nearest neighbors (kNN) search using data broadcasting is studied in this paper. Given a data set D and a query point p, the kNN search finds k data points in D closest to p. By assuming that the data is indexed by an R-tree, we propose an efficient protocol for kNN search on the broadcast R-tree in terms of the tuning time which is the amount of time spent listening to the broadcast, latency which is time elapsed between issuing and termination of the query, and memory usage on the clients. We last validate the proposed protocol by extensive experiments.

1: INTRODUCTIONS

Emerging technologies on the mobile communications and positioning systems enable a wireless environment where mobile clients can ubiquitously access the information-centric services, such as the electronic news service, traffic information, stock-price information, etc. In such an environment, data broadcasting provides an effective way to disseminate information to mobile clients in the wireless environments where the server disseminates information via the broadcast channel and each mobile client can independently retrieve the relevant data of individual interest [1][6][7][8][9][10][11][13].

Besides the asymmetric bandwidth, the energy is a scarce resource for mobile clients; therefore, raises an important issue when designing the mechanisms to provide the services. Two cost measures therefore are usually considered in a data broadcasting environment. The *latency* (i.e., the time elapsed between issuing and termination of the query) indicates the Quality of Service (QoS) provided by the system and the *tuning time* (i.e., the amount of time spent on listening to the channel) represents the power consumption of mobile clients. These two measures are the same when only the data are broadcasted. On the other hand, broadcasting data with an index provides an efficient approach to disseminate information in terms of energy consumption [6][7][8][11][12][13] and distinguished these two measures. The index allows a mobile client to tune into a broadcast only when data of interest and relevance are available; therefore, minimizes the power consumption. In our work, we further consider the amount memory used when a client executes a query since the size of memory is also usually limited.

In this paper, we use R-trees [5] and its variations [2][3] as index trees and discuss the *k* nearest neighbors (*k*NN) search on a broadcast R-tree. The *k*NN search finds the *k* objects closest to a query point *p*. Having the *k*NN search service, a mobile client can have the queries like ”please give me 10 nearest hotels” or ”please find 5 gas stations nearby”. In order to minimize the latency, tuning time, and memory used at the client simultaneously, we investigate how a server schedules the broadcast for an index tree and what the query processing is at the client side. Scheduling the index tree for broadcast involves determining the order by which the index nodes are sent out and adding additional data entries to the index nodes.

The organization for the rest of this paper is as follows. After giving the preliminaries in Section 2, we present and discuss the broadcast schedules in Section 3. The corresponding algorithm for executing *k*NN search on a broadcast index tree is proposed in Section 4. The experiment work is discussed in Section 5. Section 6 concludes this paper.

2: PRELIMINARIES

The research about R-trees for multi-dimensional data has been extensively explored in recent decades. The index node of an R-tree uses the *minimum bounding rectangle* (MBR) as its index which surrounds the MBRs of its children and contains the information of the children, including the MBRs of children. The leaf nodes only contain the MBRs of data objects. Figure 1 shows an R-tree and the corresponding MBRs of the node in the R-tree.

* Work supported by the National Science Council under the grant numbers NSC-94-2213-E-027-043

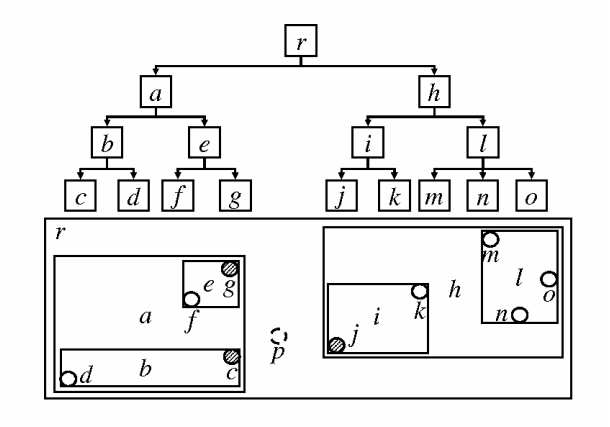


Figure 1. A *k*NN (shades) at the query point *p* (dashed circle) with *k*=3 on a 16-node R-tree.

Suppose a query point p is given. We now give three types of distance for node v in an R-tree which are usually used in the process of the kNN search on an R-tree.

- *mindist*(v) is the minimum distance from p to v's MBR; and
- *minmaxdist*(v) is the minimum distance of the maximum distances from p to each face of v's MBR.
- *maxdist*(v) is the maximum distance from p to v's MBR.

The conventional kNN search algorithm on R-trees proposed in [14] uses *mindist* and *minmaxdist* to prune the nodes which are impossible to be in the kNN of p. The authors defined $k$thdist to be the *minmaxdist* of the current kth nearest neighbor during the processing and initially ∞. A node v can be skipped if $k$thdist < *mindist*(v).

In wireless broadcast environments, the conventional kNN search algorithm mentioned above can be adopted to a broadcast R-tree but leads to a larger latency due to the serializability of a broadcasted tree [4]. Based on the conventional kNN search algorithm, the authors in [4] provided an energy-efficient kNN searching approach on broadcast R-trees using all the three types of distance. In order to reduce the tuning time, the approach used a conservative way to predict the $k$thdist and can prune the index nodes in the earlier stages. Hence, the tuning time is reduced.

For simplicity, we use one node, including the index and data (or leaf) node, as a packet in the broadcast. For any index node v, the packet basically contains v's identifier and information of v's children. The information for each child v' of v includes the address of v' in the broadcast and the index for v'. The un-shaded area in Figure 2 shows the basic content of a broadcasted index node r in Figure 1. The address allows a client to tune in when the relevant node appears in the broadcast and is crucial to reducing the tuning time [6][7][8][11][12][13]. The leaf node in the broadcast contains only the node's identifier and the data content.

r	*a*	MBR	4
	h	MBR	5

Figure 2. A broadcast index node; the un-shaded part representing the basic content; the shaded part indicating the additional entry used in this paper.

Our proposed kNN search algorithm uses *mindist*, *maxdist*, and an additional entry in the index node to prune the nodes which are definitely irrelevant to the kNN. We say a client explores node v when the client tunes into the broadcast to receive and process all the entries stored with v. Our algorithm maintains two lists: *C-List* and *R-List*. *C-List* stores the candidate nodes to be explored later and *R-List* keeps the nodes which are in the current result of kNN during the search processing. All the nodes in *C-List* and *R-List* are ordered by *maxdist*, respectively. The objective is to minimize the tuning time and latency as well as the memory usage for a particular kNN search at mobile clients.

3: DATA BROADCAST SCHEDULES

There are two aspects which should be considered when designing a data broadcasting protocol in a wireless environment. One is the broadcast schedule at the server and the other is the corresponding query process at the mobile clients. In this section, we discuss the broadcast schedules at the server. The schedules based on the breadth-first traversal can achieve a better tuning time [4] but result in a large memory usage at the mobile clients. Furthermore, most of the existing algorithms for different types of queries on broadcasted R-trees use the broadcast schedules based on the depth-first traversal [1][4][6][7][8][9][10][11][13]. We thus consider the broadcast schedules based on the depth-first traversal. Our work tailors toward one broadcast R-tree for different kinds of queries.

We consider two broadcast schedules based on the depth-first traversal and these two broadcasts differs in the ordering of the children of each node in the R-tree. The first broadcast schedule, pDFS, organizes the broadcast simply by the depth-first order. Such a broadcast schedule for R-trees has been used and studied in many papers [4][6][7][8][9][10][11][13]. However, such a broadcast schedule does not consider any factor that might effect the performance in terms of the tuning time, latency and memory usage. We thus consider a variation of the pDFS, called wDFS. The wDFS broadcast schedule will rearrange the R-tree by the subtree sizes in a non-increasing order and then place the nodes in the broadcast according to the depth-first order. Figure 3 shows the broadcasts of the

R-tree in Figure 1 generated by the *p*DFS and *w*DFS schedules, respectively.

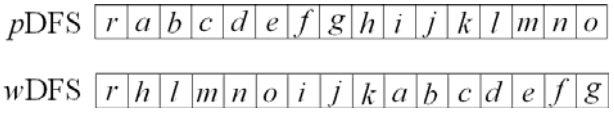


Figure 3. The broadcasts generated by the pDFS and wDFS schedules respectively for the R-tree in Figure 1.

Recall that a broadcast packet corresponds to a node in the R-tree. The un-shaded area in Figure 2 indicates the basic content of a broadcast index node. In order to effectively prune nodes which are irrelevant to a *k*NN search when exploring a node, we add an entry, *l-entry*, for each child. The *l*-entry of a child *v*', *l*(*v*'), is the number of leaves in the subtree rooted at *v*'. Figure 2 shows a broadcast node considered in this paper and the shaded area presents the *l*-entry of each child of node *r*. The number of leaves rooted at the child node *a* is four (i.e. *l*(*a*)=4). We will discuss it in more details in the next section.

4: EXACT *k*NN SEARCH ALGORITHM

This section introduces our exact *k*NN search algorithm, *w-disk*, on a broadcast R-tree. We will show that our *k*NN search algorithm can find the *exact k*NN efficiently and analyze the time complexity for exploring a node in the R-tree. Algorithm *w-disk* will determine an imaginary circle *C* centered at the query point *p* using *maxdist* as the radius. With such a circle and the *l*-entry added in the child's information in the broadcast, the algorithm can decide which node and its descendants are irrelevant to the *k*NN and exclude them for further exploring to achieve a shorter tuning time and latency.

Suppose algorithm *w-disk* starts from the beginning of the broadcast cycle (i.e. the root of the R-tree). For a *k*NN search at query point *p*, we let *U* denote the union of *C-List* and *R-List*. Amid all the nodes in *U*, there is at least one node *u* having the following property

$$\sum_{\substack{v \in U \\ \max dist(v) \le \max dist(u)}} l(u) \ge k \qquad (1)$$

where $l(v)$ is the number of leaves in the subtree rooted at *v*. Among the nodes having the above property, we refer to the node whose *maxdist* is *minimum* as the *Pnode*. The *Pnode* is used to prune the nodes irrelevant to the *k*NN. Based on the Pnode *u*, one can generate a circle C_u centered at *p* with radius *maxdist*(*u*). Suppose there are *n* MBRs inside C_u and the corresponding nodes are $u_1, \ldots, u_n$. We denote the total number of leaves in the subtrees rooted at $u_1, \cdots, u_n$ as

$$S_u = \sum_{i=1}^{n} l(u_i) \qquad (2)$$

where $l(u_i)$ is the number of leaves in the subtree rooted at u_i and $S_u \ge k$.

Initially, the Pnode is a pseudo-node and the radius of the corresponding circle is ∞. The algorithm starts with receiving the root and then explores the root. During the exploration, all the children of the root are inserted into *C-List* since all the MBRs of the children are in the corresponding circle of the Pnode. After the insertion, a new Pnode is calculated and used to prune the irrelevant nodes in the current and next explored node. The next node to be explored is the node closest to the currently explored node in the broadcast in *C-List*.

In general, suppose node *v* is the next node to be explored, algorithm *w-disk* works as follows. Assume that the Pnode determined in the previous explored node is *u* and the corresponding circle C_u is centered at *p* with radius *maxdist*(*u*). Consider that node *v* is received from the broadcast channel. There are two cases for node *v*. First, when node *v* is a leaf node, we then insert *v* into *R-List*. If *R-List* is full, we remove the one having the maximum *maxdist* among all the nodes in *R-List* including node *v*. Then, we consider the next node to be explored from *C-List* as before and the process continues.

The other case is that node *v* is an index node. For each child *v*' of *v*, the algorithm first uses C_u to decide whether *v*' can be ignored. Should *mindist*(*v*) be greater than *maxdist*(*u*), child *v*' can be ignored since it is impossible for the leaves in the subtree rooted at *v*' to be in the *k*NN; otherwise, insert *v*' into *C-List*. The algorithm then calls FINDPNODE() to find a new Pnode *u*' among all the nodes in the current *C-List* and *R-List*. It is not difficult to find a new Pnode *u*' since both of the lists are sorted by *maxdist* in a non-decreasing order. With the Pnode *u*' and its corresponding circle $C_{u'}$, algorithm deletes all the nodes in *C-List* and *R-List* whose MBRs are outside $C_{u'}$ and keeps all the nodes whose MBRs intersect with $C_{u'}$. Then, the algorithm extracts the next node to be explored from *C-List*. Figure 4 shows the high-level description about exploring a node in algorithm *w-disk*.

Algorithm *w-disk* starts from the beginning of the broadcast cycle (i.e., the root) and uses the above algorithm to explore a node. The algorithm stops when *C-List* is empty (i.e. there is no next node to be explored). We now use the R-tree in Figure 1 to illustrate how algorithm *w-disk* works. Consider a *k*NN search at the query point *p* with *k* = 3 with a *p*DFS data broadcast schedule. Algorithm *w-disk* first explores the root *r*. During the exploration, nodes *a* and *h* are placed into *C-List* with the order of *a*, *h* since *maxdist*(*a*) < *maxdist*(*h*) (Step 1.2). Then the algorithm needs to find a new Pnode from *C-List* (Step 1.3). Node *a* is the new Pnode and the corresponding circle is C_a because *maxdist*(*a*) < *maxdist*(*h*) and $S_a > k$. Having the new Pnode *a*, the algorithm examines all the nodes in *C-List* to discard the irrelevant nodes (Step 1.4). However, the MBR of *h* intersects with C_a and *h* is thus kept in *C-List*. After exploring the root *r*, the next node to be explored is node *a* since node *a* is broadcast earlier than node *h*

Algorithm *k*NN-Explore(*v*)
/* u is the Pnode used in the previous explored node */
(1) **if** node *v* is a leaf node **then**
  (1.1) **INSERT** *v* into *R-List*
  else /* node v is an index node */
  (1.2) **for** each child *v*' of *v* **do**
    if *mindist*(*v*') ≤ *maxdist*(*u*) **then**
      INSERT *v*' into *C-List*
  (1.3) *u*'=**FINDPNODE**();
  (1.4) **DELETE** the nodes of which *mindist* > *maxdist*(*u*') from the both lists
(2) let *w* be the node closest to the currently explored node in *C-List*
(3) *k*NN-Explore(*w*)
End Algorithm *k*NN-Explore

Algorithm FINDPNODE()
(1) Scan the nodes in *C-List* and *R-List* by *maxdist* in a non-decreasing order using the way like the merge sort until the first node *u* whose $S_u \geq k$;
(2) **return** *u*

Figure 4. Client algorithm for exploring a node in the *k*NN search.

(Step 2) and the algorithm waits for node *a* in the broadcast to proceed the search process.

When exploring node *a*, both of *a*'s children are inserted into *C-List* since the *mindist* of each child of *a* is smaller than *maxdist*(*a*). After the insertion, the order of nodes in *C-List* is *e*, *b* and *h*. Recall that the nodes in *C-List* are ordered by the *maxdist* in a non-decreasing order. It is not difficult to find the new Pnode *b* with the corresponding circle C_b as follows. The algorithm first considers node *e* since *maxdist*(*e*) is smallest among all the nodes in *C-List* and *R-List*. However, $S_e = l(e) = 2 < k = 3$, the algorithm next considers node *b* for the new Pnode. Because the MBR of *e* is in C_b and C_b contains more than $k = 3$ leaves, i.e. $S_b = l(e) + l(b) = 4 > 3$, node *b* is the new Pnode and the algorithm then uses C_b to decide the irrelevant nodes in both *C-List* and *R-List*. The relation among nodes *e*, *b* and h and C_b is shown in Figure 5. The algorithm then extracts node *b* from *C-List* to be the next node to be explored since *b* is broadcast before *e* and *h*. The process then proceeds in the same way.

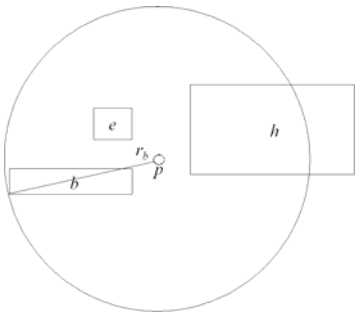


Figure 5. The relation among nodes *e*, *h*, and the Pnode *b* when exploring node *a*.

For the rest of this section, we show the correctness and the time complexity of algorithm *w-disk*. Due to the space limitation, we state the theorems without proof.

Theorem 1. *Given a kNN search at query point p, algorithm w-disk can finds the exact k nearest neighbors.*

Theorem 2. *For a broadcasted R-tree having height h and fanout B, it takes O(B · max{k, hB}) time to explore a node.*

5: EXPERIMENTAL RESULTS

In this section, we present our experimental results and compare our *k*NN search algorithm *w-disk* with the revised conventional approach *w-conv* and the improved algorithm *w-opt* in [4]. The cost measures include the tuning time, latency and memory usage. When discussing each measure, we include the optimal cost for comparison. Besides, we also compare the impact on the performance resulting from the two broadcast schedules, *p*DFS and *w*DFS, discussed in Section 3.

We use R*-trees [2] as the index tree on point data in the experiment. The trees have 150,000 leaves and the node fanout between 12 and 24. The point data are generated using a uniform distribution within the unit square and correspond to the leaves. The value of *k* varies from 1 to 210. For each value of *k*, data reported is the average of 1,000 different *k*NN searches with different query points selected uniformly.

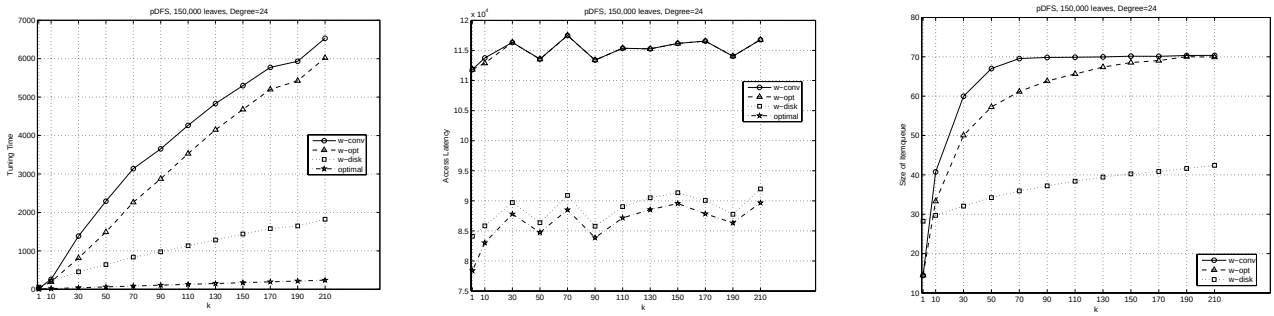
5.1: TUNING TIME

The tuning time reflects the power consumption for the *k*NN search at the mobile clients. Figure 6(a) shows the comparison of the tuning time for three algorithms on the R*-tree with fanout 24 and 150,000 leaves. The tree has a height of 6 and a total of 159,185 nodes. The *k*NN search starts from the beginning of a broadcast cycle. The broadcast schedule in Figure 6 uses *p*DFS. The *x*-axis reflexes the different value of *k*.

Recall that *w-opt* uses a conservative approach to determine *k*thdist in an earlier stage; therefore, can prune the nodes effectively. With *l*-entry added in the broadcast node, our algorithm *w-disk* can decide the range of the *k*NN more accurately. Hence, *w-disk* can avoid exploring more nodes not in the resulting *k*NN. The experimental results indicate that our *w-disk* algorithm can explore fewer nodes than the other two algorithms; therefore, leads to a fewer tuning time. This trend becomes obvious as the value of *k* increases. In general, the difference in the tuning time becomes big as *k* increases. Figure 6(a) also shows the optimal tuning time. However, we conjecture that it is very hard to achieve.

5.2: ACCESS LATENCY

This section compares the access latency of different *k*NN search algorithms on broadcasted R*-trees. As shown in Figure 6(b), algorithm *w-disk* achieves the best



(a) Tuning Time (b) Latency (c) Memory Usage

Figure 4. Performance Comparisons for different values of *k* using algorithms *w-conv*, *w-opt*, and *w-disk* on an R*-tree with fanout 234 and 150,000 leaves.

latency which is closest to the optimal latency . The other two algorithms yield almost the same latency.

In *w-opt*, some redundant nodes are kept for the result to generate the *k*thdist. Using the *k*thdist, the nodes to be explored later are determined and stored . The *k*thdist changes as the algorithm proceeds but the nodes stored to be further explored are not checked using the new *k*thdist; therefore, may result in a longer latency. Such a case also occurs in *w-conv*. Since *w-disk* examines the *C-List* and *R-List* each time when a new Pnode is determined, *w-disk* leads to a much shorter latency.

5.3: MEMORY USAGE

The size of *C-List* denotes the amount of storage used during the *k*NN search processing. Figure 6(c) shows the comparison of the maximum amount of storage used for these three algorithms. The results show that *w-disk* uses fewest storage space among these three algorithms and *w-opt* performs better than *w-conv*.

Algorithm *w-disk* uses the Pnode to prune the irrelevant nodes when exploring a node. Such a Pnode is derived by considering the number of leaves in its corresponding circle; therefore, leads to a better approximation to the resulting *k*NN. Furthermore, *w-disk* uses the previous Pnode to delete the irrelevant children and the new Pnode to delete the irrelevant nodes in *C-List* and *R-List*. As a result, *w-disk* needs fewer storage to execute a *k*NN search.

5.4: *p*DFS v.s. *w*DFS

We now discuss the impacts result from the two broadcast schedules *p*DFS and *w*DFS in Section 3. The broadcast schedule using *w*DFS broadcasts the larger subtrees first in DFS fashion. The experimental results show that the broadcast schedule using *w*DFS can achieve a shorter tuning time than the one using *p*DFS. This conclusion holds in all our experiments. Figure 7(a) presents the comparison of the tuning time using *w*DFS and *p*DFS broadcast schedules respectively. Broadcasting the node which has a larger subtree size first allows the mobile clients to have a better approximation for the *k*NN in an earlier stage since more MBRs can be obtained earlier. Such an impact results in a fewer tuning time no matter which algorithm is applied. On the contrary, the broadcast schedule using *w*DFS leads to a longer latency. This trend is shown in Figure 7(b). Broadcasting the node which has a larger subtree size first forces the query process to wait to the very end for the relevant node with a smaller subtree size. However, the broadcast schedule using *p*DFS does not need to wait for such a case. In general, the broadcast schedule using *p*DFS outperforms the one using *w*DFS in average in terms of the latency.

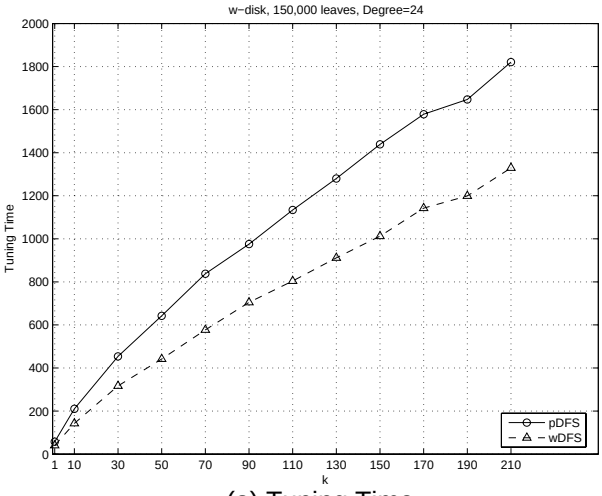
6: CONCLUSIONS

In this paper we propose an effective *k*NN search protocol on broadcasted R-trees in a wireless broadcast environment. In the broadcast environments, two directions are considered when designing the protocols. One is to consider how the server broadcasts the data. The other is what the corresponding process is on the client side. Previous work focused on either side. We consider the server and client aspects respectively. By adding an additional entry in the broadcast on the server side, our *k*NN algorithm at the client achieves fewer tuning time and shorter latency with fewer storage. We also consider two different broadcast schedules based on DFS. The results of the extensive experiments validate that our mechanisms achieve the objectives. On the broadcast R-trees, many types of queries have been studied. Our work in this paper tailors toward one broadcast R-tree for different kinds of query

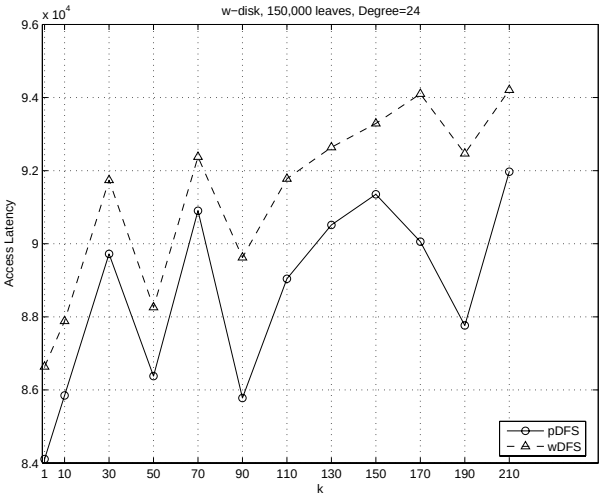
REFERENCES

[1] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcasts. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 183–194, 1997.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 322–331, 1990.



(a) Tuning Time



(b) Latency

Figure 5. Comparisons of pDFS and wDFS broadcast schedules on tuning time (a) and latency (b) for different values of k using algorithm w-disk.

[3] V. Gaede and O. G¨unther. Multidimentional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[4] B. Gedik, A. Singh, and L. Liu. Energy efficient exact *k*NN search in wireless broadcast environments. In *12th ACM International Workshop on Geographic Information Systems*, pages 137–146, 2004.

[5] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD Conference on Management of Data*, pages 47–57, 1984.

[6] S. Hambrusch, C.-M. Liu, W. G. Aref, and S. Prabhakar. Broadcasting indexed multidimensional data. To appear in *Data and Knowledge Engineering*.

[7] S. Hambrusch, C.-M. Liu, and S. Prabhakar. Broadcasting and querying multi-dimensional index trees in a multichannel environment. To appear in *Information Systems*.

[8] S. E. Hambrusch, C.-M. Liu, W. G. Aref, and S. Prabhakar. Query processing in broadcasted spatial index trees. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 502–521, 2001.

[9] S. Hameed and N. Vaidya. Efficient algorithms for scheduling data broadcast. *ACM/Baltzer Journal of Wireless Networks*, 5(3):183–193, 1999.

[10] A. Hurson and Y. Jiao. Data broadcasting in a mobile environment. In *Wireless Information Highway*, chapter 4, pages 96–154. IRM Press, 2004.

[11] T. Imieli´nski, S. Viswanathan, and B. R. Badrinath. Data on air: Organization and access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):353–372, 1997.

[12] S. Khanna and S. Zhou. On indexed data broadcast. *Journal Computer and System Sciences*, 60:575–591, 2000.

[13] C.-M. Liu. *Broadcasting and blocking large data sets with an index tree*. PhD thesis, Purdue University, West Lafayette, IN, 2002.

[14] N. Roussopoulos, S. Kelley, and F.Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, 1995.