

Energy Efficient Exact k NN Search in Wireless Broadcast Environments

Bugra Gedik
Georgia Institute of Tech.
bgedik@cc.gatech.edu

Aameek Singh
Georgia Institute of Tech.
aameek@cc.gatech.edu

Ling Liu
Georgia Institute of Tech.
lingliu@cc.gatech.edu

ABSTRACT

The advances in wireless communication and decreasing costs of mobile devices have enabled users to access desired information at any time. Coupled with positioning technologies like GPS, this opens up an exciting domain of location based services, allowing a mobile user to query for objects based on its current position. Main bottlenecks in such infrastructures are the draining of power of the mobile devices and the limited network bandwidth available. To alleviate these problems, *broadcasting* spatial information about relevant objects has been widely accepted as an efficient mechanism. An important class of queries for such an infrastructure is the k -nearest neighbor (k NN) queries, in which users are interested in k closest objects to their position. In this paper, we describe mechanisms to perform *exact* k NN search on conventional sequential-access R-trees, and optimize established k NN search algorithms. We also propose a novel use of histograms for guiding the search and derive analytical results on maximum queue size and node access count. In addition, we discuss the effects of different broadcast organizations on search performance and challenge the traditional use of Depth-First (dfs) organization. We also extend our mechanisms to support k NN search with non-spatial constraints. While we demonstrate our ideas using a broadcast index, they are equally applicable to any kind of sequential access medium like tertiary tape storage. We validate our mechanisms through an extensive experimental analysis and present our findings.

1. INTRODUCTION

With the coming era of ubiquitous computing, the popularity of mobile communications and emergence of positioning technologies like GPS have laid a strong foundation for location based services (LBSs) [6, 21]. Location based services provide users with information that is specialized to their location, for example, the nearest gas station or the five closest restaurants.

In mobile wireless environments, LBSs are accessed through a common wireless channel, which connects the users to the service provider. There are two important constraints of such mobile wireless environments: (i) Limited network bandwidth, and (ii) Power constrained user devices (mobile units). Efficient mechanisms to overcome these constraints will eventually play a pivotal role in providing such location based services.

One straight-forward way of providing location-based service is to let each mobile unit query the LBS server by establishing a one-to-one connection and have the server reply to mobile units on a per-query basis. However, this *on-demand* access model has a few critical drawbacks. First, due to large number of users that need access to the service, it creates a processing bottleneck on the server side and contention on the wireless medium. Second, it fails to exploit the similarity of content desired by all users.

Another way of providing location-based service is to use the *broadcast* access model, in which information is continuously broadcasted on the wireless medium and each mobile unit tunes-in to the broadcast and reads relevant information whenever a query is issued. This prevents the costly transmission of queries from individual mobile devices to the server, thus reducing the processing and bandwidth bottlenecks. As a result, the *broadcast* model is considered most suited to such scenarios and has been an important topic of interest in the research community [13, 5, 26, 2, 3, 29, 12]. An important challenge in this model of access is to devise efficient indexing and searching mechanisms for *energy efficient* querying of location dependent broadcast data. The goal is to minimize energy consumption at the mobile unit, thus increasing its lifetime.

One essential class of queries is the k -nearest neighbor queries, often categorized as the “ k NN Search on the Air” problem [30] in the current context. In totality, it can be described as broadcasting location dependent data, together with a spatial index on the wireless medium and searching this broadcast to answer k NN queries in an energy efficient manner. To illustrate the problem, consider a battlefield scenario where several mobile units report their positions to a central server through a wireless channel. The server builds a spatial index on this position information and securely broadcasts it to the battlefield. Then the mobile devices can tune in and process the broadcast to answer spatial queries on the broadcasted position information. One such query can be posed by a soldier as: “Give me the positions and names of the 10 nearest friendly units”. In a different scenario, taking place in a commercial setting, the server can broadcast locations of various restaurants to answer queries like: “Give me the positions of the 5 nearest restaurants”.

In this paper, we study the problem of exact k NN search on R-trees [8] in wireless broadcast environments. Our results also apply for k NN search on serialized R-trees for any sequential access medium. We chose R-trees as the basis of our work because of its two primary advantages. First, it can be used to support other important kinds of queries, like range queries. This prevents developing exclusive, incompatible solutions for each class of queries. Second, it is well studied and established as an efficient indexing mechanism.

Our main technical contributions are as follows:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

- We develop an algorithm, called *w-opt*, which introduces revisions and optimizations over well established *k*NN search algorithms, and provides better performance on wireless mediums.
- We describe the effects of different broadcast organizations on search performance and show that commonly employed depth first serialization of the tree, under certain conditions, results in reading many nodes that do not contribute to the query result, thus causing higher energy consumption.
- We propose the use of histograms for guiding the search, and develop an associated algorithm, called *w-hist* algorithm. It uses simple equi-spaced grid histograms to supplement the index and achieve further pruning of nodes, resulting in lower number of node accesses and smaller memory requirements.
- We also derive analytical results on maximum queue size and node access count, for uniform data distributions. Furthermore, we show how to extend the use of histograms for supporting *k*NN search with non-spatial constraints (we only consider equality constraints on *types*).
- We provide experimental results to evaluate the introduced mechanisms and understand the tradeoffs involved in using them. In our experiments, we use both synthetically generated uniform data and real data with skewed distribution.

The rest of the paper is structured as follows. Related work is discussed in Section 2 and an overview of *k*NN search on R-trees is presented in Section 3. Section 4 introduces techniques to optimize *k*NN search on serialized R-trees and Section 5 studies the index organization tradeoffs. In Section 6, the use of histograms is explained in the context of *k*NN search on the air and analytical results are derived for maximum queue size and node access count. Queries with type constraints are discussed in Section 7. Section 8 presents experimental results and Section 9 concludes the paper.

2. RELATED WORK

In this section we list three different areas of related research. The first two, R-tree indexes and indexing on the air, establish the basis of our work. The last one, spatial index broadcast, consists of work closely related to ours, which has considered energy efficient search on spatial data in wireless broadcast environments.

R-tree Indexes: R-trees [8] are spatial index structures widely used to index *n*-dimensional points or rectangles. Due to practicality of implementing them on secondary storage and their good search performance in low-dimensional spaces, R-trees have enjoyed wide deployment. R-trees can be thought of as the multidimensional version of B⁺-trees. An R-tree node consists of (i) a minimum bounding rectangle (*mbr*) which encompasses *mbr*s of all nodes under its branch, (ii) *mbr*s of its children nodes, and (iii) pointers to its children nodes. A node at the leaf level contains *mbr*s of data objects and pointers to data objects, instead of child *mbr*s and pointers. Figure 1 shows an example R-tree structure at the top and the set of indexed points together with the node *mbr*s at the bottom.

The research on R-trees is still active, especially in the field of mobile object indexing [16, 23]. Several variations of R-trees exist including R*-trees [4] and R⁺-trees [25]. R*-trees have been shown to work well for various data and query distributions and in this paper we use R*-trees for performance

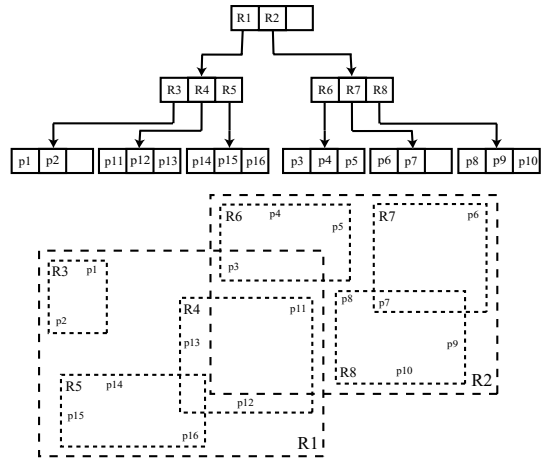


Figure 1: Illustration of an R-tree index

evaluation.

Indexing on the Air: In wireless broadcast, it is crucial that energy is conserved on the mobile unit side when answering queries on the broadcasted data. To alleviate the vast energy consumption problem of searching un-indexed data, air indexes were introduced in [13]. Air indexes trade latency in order to reduce energy consumption. Using the index, the mobile unit can selectively tune-in only to the relevant portion of the broadcast and thus optimize its energy consumption. However, the inclusion of the index increases the total size of the broadcast cycle and thus increases latency. This latency involved in searching broadcasted data is measured as *access latency*, which is defined as the time difference between the point at which a query is posed and the point at which result of the query is fully computed. The energy consumption is measured as *tune-in time*, which is defined as the total time during which the mobile unit was listening to data from the wireless medium. Both access latency and tune-in time are measured in terms of number of packets, where a packet corresponds to the physical unit of access on the broadcast medium, similar to disk blocks. Similar to [13], our work also assumes that a single index node corresponds to a single packet.

Air indexing strives to decrease tune-in time while keeping the increase in access latency due to the broadcast of extra index information minimal. The (1, *m*) indexing approach [13], widely adopted by many other researchers, interleaves data and index on the medium for the purpose of improving the access latency. The (1, *m*) indexing approach can be applied to most of the tree based indexes, including ours. Hence, in our work, we only concentrate on improving the tune-in time associated with *index search* in broadcast environments.

Broadcast disks [2] is another commonly used [3, 29, 12] technique for efficient data access. Broadcast disks build a memory hierarchy through the use of repetitive broadcasts by adjusting the occurrence frequency of data items based on their popularity. We do not consider broadcast disks, since our focus in this paper is on spatial data broadcast and air indexes form an ideal framework for our work.

Spatial Index Broadcast: Recently, researchers have considered energy efficient query evaluation over location dependent data in wireless broadcast environments [30, 10, 28, 15]. In [28], authors have discussed processing of nearest neighbor (*k* = 1) queries on location dependent data. Their approach is based on building a voronoi diagram [18] and broadcasting it with the use of a new index structure called D-tree. However their approach does not extend to *k* nearest neighbor search.

Algorithm 1: R-tree kNN search

```

1: ItemQueue ← {rootNode} // sorted by MINDIST
2: ResultQueue ← ∅ // sorted by MINMAXDIST, size ≤ k
3: while ItemQueue ≠ ∅ do
4:   item = ItemQueue.pop()
5:   if item is an object then
6:     ResultQueue.add(item)
7:   if all items in ResultQueue are objects then
8:     return ResultQueue
9:   end if
10: else if item is a node then
11:   Read item from the medium
12:   for all citem in item.childrenList do
13:     if MINDIST(citem, P) > kthdist then
14:       continue
15:     end if
16:     ItemQueue.add(citem)
17:     ResultQueue.add(citem)
18:   end for
19:   ResultQueue.remove(item)
20: end if
21: end while

```

Algorithm 2: R-tree kNN search adapted for wireless

```

1: ItemQueue ← {rootNode} // sorted by appearance order
2: ResultQueue ← ∅ // sorted by MINMAXDIST, size ≤ k
3: while ItemQueue ≠ ∅ do
4:   item = ItemQueue.pop()
5:   if MINDIST(item, P) > kthdist then
6:     continue
7:   else if item is an object then
8:     ResultQueue.add(item)
9:   else if item is a node then
10:    Read item from the medium
11:    for all citem in item.childrenList do
12:      if MINDIST(citem, P) > kthdist then
13:        continue
14:      end if
15:      ItemQueue.add(citem)
16:      ResultQueue.add(citem)
17:    end for
18:    ResultQueue.remove(item)
19:   end if
20: end while
21:

```

In addition, the indexing structures are exclusive to this class of queries. In [10], authors have studied the problem of answering range queries over location dependent data for memory limited devices, but their techniques are not extendable to nearest neighbor queries. In [30], authors have studied processing of k nearest neighbor queries on location dependent data. However, they only provide approximate search techniques. Moreover, [30] favors a list based structure in place of R-trees for small values of k . Also, their experimental comparison is based on R-trees, not R^* -tree variants, which are known to be more efficient. In our work, instead of introducing yet another indexing structure, we describe techniques to adapt and revise k NN search on R-tree family for wireless broadcast and more importantly also provide *exact* results.

3. BACKGROUND

In this section we give a brief overview of k NN search on R-trees. The description is based on Roussopoulos et. al's [22] algorithm for k NN search on R-trees. Before describing the algorithm, we give definitions of three measures that are essential for the description of the algorithm and will be used in the rest of the paper. For a given query point P , a node N , and an object O ,

- MINDIST(N, P) is the minimum distance from P to N 's mbr. MINDIST(O, P) is the minimum distance from P to O 's mbr (in case O is a point, O 's mbr reduces to its position).
- MAXDIST(N, P) is the maximum distance from P to N 's mbr, where MAXDIST(O, P) is equal to MINDIST(O, P).
- MINMAXDIST(N, P) is the maximum possible minimum-distance between P and the mbr of closest object residing in N 's mbr. MINMAXDIST(O, P) is equal to MINDIST(O, P).

Note that, the first two definitions imply that, no object residing under node N 's mbr can have a MINDIST value for point P less than MINDIST(N, P) or larger than MAXDIST(N, P). The third definition implies that, the object which resides under node N 's mbr and has the smallest MINDIST value for point P , has a MINDIST value of at most MINMAXDIST(N, P).

The conventional k NN search algorithm keeps two data structures to guide its operations. These are *ItemQueue* and *ResultQueue*. Both of them store either node or object identifiers together with their mbrs. *ItemQueue* is sorted on the MINDIST measure and does not have a predefined size. On the other hand, *ResultQueue* is sorted on the MINMAXDIST measure and can have at most k entries. We define the variable *kthdist* such that it takes the value ∞ if there are less than k entries in *ResultQueue*, otherwise it takes the value of MINMAXDIST measure for the k th item in *ResultQueue*.

The algorithm works iteratively. Initially *ItemQueue* contains only the root node. At each iteration, the topmost item in *ItemQueue* is popped. If the item is a node (say N) entry, then the node is read from the medium. Entries for the children nodes of N , whose MINDIST values from the query point are smaller than the *kthdist*, are added to *ItemQueue* and *ResultQueue*, and N 's entry is removed from *ResultQueue* if it is there. If the initially popped item is an object, then it is added only to *ResultQueue* and the termination condition is checked. In case all entries in *ResultQueue* refer to objects then the search halts. Algorithm 1 gives the pseudo code of k NN search on R-trees.

4. KNN SEARCH FOR WIRELESS MEDIUM

While searching a *serialized* R-tree in the wireless broadcast scenario, using Algorithm 1 as a base, it is not possible to sort *ItemQueue* on the MINDIST measure. Since the MINDIST ordering of tree nodes is not consistent with their order of appearance in the broadcast, reading a node from the medium based on the topmost item in the MINDIST sorted *ItemQueue* may result in leaving behind other tree nodes that have entries in *ItemQueue*. As these other nodes are left behind on the medium, accessing them in the future steps of the algorithm will require a wait until the next index broadcast, which is prohibitive in terms of access latency. As a result, the items in *ItemQueue* have to be sorted based on their appearance order on the medium. In accordance with this observation, previous work [30, 15] on this topic have considered searching serialized R-trees for k nearest neighbors using queues sorted on appearance order. Furthermore, when an item is popped from *ItemQueue*, we can first check whether the MINDIST value of the item is larger than *kthdist*. If so, it is safe to prune the

item and proceed with the next iteration¹. It is also important to mention that the stop condition in Algorithm 1 (lines 7-8) is no more valid as it is not guaranteed that the rest of the items in *ItemQueue* cannot generate an object closer than the current k , since the queue is no more sorted on MINDIST measure. As a result, the search halts when the *ItemQueue* becomes empty. Algorithm 2 gives the pseudo code of k NN search on R-trees adapted for wireless medium.

The adapted version of k NN search for wireless medium is obviously far from being optimal due to its almost unguided exploration of the nodes. The efficiency of the search, in terms of tune-in time, relies on the pruning capabilities of the algorithm and the organization of the index. Next, we illustrate that some cases that do not occur in the original version of the algorithm, but appear in the adapted version can be pruned by a simple yet effective optimization.

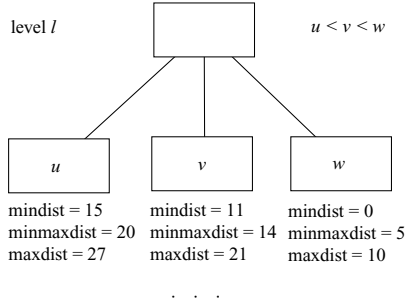


Figure 2: The topmost two levels of an example tree

4.1 The w -opt Algorithm

In this section, we describe the w -opt algorithm and how it improves k NN search performance on wireless medium. Before giving the details, we first illustrate a particular inefficiency of the adapted algorithm with an example. This forms the bases of our improved algorithm.

Consider searching the tree depicted in Figure 2 for two nearest neighbors. Clearly, the candidates for the two nearest neighbors should come only from nodes under w , since its mbr covers at least two objects (one at distance at most 5 and another at distance between 5 and 10) closer than any other object under node u and v . Now, assume that nodes u, v and w are organized on the medium such that u appears before v and v before w . As a result, after the root node is processed, node u 's entry appears on top of *ItemQueue* since the queue is sorted by appearance order. *ResultQueue* on the other hand is sorted based on MINMAXDIST, thus consists of entries of w and v in order and $kthdist$ is equal to the MINMAXDIST of v , which is 14. Next, u 's entry is popped from the queue and is discarded since its MINDIST is larger than the $kthdist$ ($15 > 14$). In the next iteration of the algorithm, v 's entry is popped from the queue and the node v is read from the medium as it does not satisfy the pruning constraint, its MINDIST, which is 11, is smaller than the $kthdist$, which is 14. This is an important pitfall of the adapted algorithm. Node v is read even when it will not contribute to the result! The original algorithm would have explored w before v because the *ItemQueue* is sorted by MINDIST. In fact, the original algorithm will examine several other nodes (with MINDIST values less than node v 's), until the node v 's entry comes to the front of the queue. Then v will have a much higher chance of being discarded. This is because $kthdist$ will improve as better and

¹Notice that this pruning condition is never satisfied when *ItemQueue* is sorted on the MINDIST measure.

better nodes are explored.

In Figure 2, it is indeed possible to prevent node v from being read from the medium by the adapted algorithm. If the minimum fanout of the tree is f_{min} , then we know that there are at least f_{min}^{l-1} objects under node w 's mbr (assuming the leaf nodes are at level 1 and the root node is at level l). Given this knowledge, at the time when we add w 's entry to *ItemQueue*, we can say that there is one object at most MINMAXDIST(w, P) away from the query point and at least $f_{min}^{l-1} - 1$ objects at most MAXDIST(w, P) away from the query point, where P is the query point. Returning back to the figure, this means there is one object at most 5 unit away from the query point and $f_{min}^{l-1} - 1 \geq 1$ objects at most 10 units away from the query point. This information allows us to discard v in the next iteration since its MINDIST is 11 and we already know there exists an object at most 5 units away and another object at most 10 units away from the query point. It is important to notice that the additional information employed, which uses the MAXDIST measure and the existence of f_{min}^i objects under a node at level i , helps us to cut down the number of nodes read from the medium, because it suppresses the unguided node exploration nature of the adapted algorithm.

Utilizing the above observation, we improve the adapted k NN search algorithm as follows:

- While adding a node entry, say node N at level i , with its MINMAXDIST measure to *ResultQueue*, we also insert $f_{min}^i - 1$ additional entries with the MAXDIST measure of node N . *ResultQueue* is sorted on the associated measures of the entries. When we explicitly remove a node from *ResultQueue*, we remove all entries associated with that node.

In the rest of the paper we refer to the R-tree k NN search algorithm designed for a random access medium as *the conventional algorithm*, adaptation of the conventional algorithm to wireless broadcast medium as *the wireless conventional algorithm* (w -conv for short) and the improved version of the adapted algorithm as *the optimized wireless algorithm* (w -opt for short).

5. DFS VS. BFS

The way a spatial index is organized on the broadcast medium impacts the number of index nodes read by the k NN search algorithm, thus affecting the tune-in time. Previous work on range and k NN search in broadcast environments [28, 10, 30, 15] that has considered using R-trees, used a depth first search (dfs) order serialization of the tree, mostly because the conventional algorithm is based on a heuristically guided dfs . We present an argument in favor of serializing the tree based on breadth first search (bfs) order and in Section 8 we experimentally show that bfs is actually the better choice.

For a given query point P , let $Q_{P,k}$ denote the position of k th nearest neighbor of point P . A k NN search algorithm on R-trees is said to be *optimal* if it only accesses nodes whose mbr's intersect with the circle centered at P with radius equal to the distance between P and $Q_{P,k}$ [24].

Figure 3 shows a query point P and the circle formed around it using its distance from

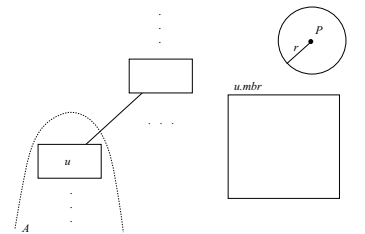


Figure 3: Example illustrating the downside of dfs organization

its k th nearest neighbour. Let us call this circle *result circle* of the query point. Since the k NN search on wireless medium is not optimal, it is possible to read a node from the medium such that the mbr of the node does not intersect with the result circle of the query point. Figure 3 shows one such example tree node u . If tree is serialized on *dfs* order, then the algorithm will access several other nodes (from the nodes of the tree under the area A in Figure 3) under node u until the algorithm is done with the branch rooted at node u without contributing any objects to the result. In fact, we have the following result,

THEOREM 1. *For a dfs serialized tree, if the w -opt k NN search algorithm reads a node N_i at level i (the leaf nodes are at level 1) whose MINMAXDIST is smaller than the k thdist, then it must read at least i nodes from the branch rooted at node N_i irrespective of whether N_i intersects with result circle or not.*

Proof: Since the algorithm visits nodes in *dfs* order on the broadcast medium, only the nodes under N_i 's branch can be accessed until the branch has totally passed on the medium. Let O be the closest object under N_i 's mbr to the query point. Since N_i 's MINMAXDIST is smaller than the k thdist, O 's distance to the query point is smaller than the value of k thdist at the time N_i is popped from the queue. Furthermore, no node under N_i 's branch can have a MINMAXDIST value smaller than O 's distance to the query point. (This follows since otherwise we'll have another object under N_i 's mbr that is closer to the query point than O , which is a contradiction.) As a result, k thdist cannot have a value smaller than O 's distance to the query point, at any time before the algorithm leaves N_i 's branch behind. It directly follows that any node under N_i 's branch whose mbr contains O must be read, since its MINDIST can be at most equal to O 's distance to the query point which cannot be larger than k thdist. There are at least i nodes whose mbrs contain O under N_i 's branch since N_i is at level i . Thus the result is proved. \square

The *bfs* ordering clearly does not share the same problem. This is because the nodes are serialized level by level and there is no recursive containment relationship between successive nodes within a level. However, *bfs* serialization may have a larger memory requirement due to the growth in *ItemQueue*. We investigate this issue analytically in Section 6 and experimentally in Section 8.

6. THE *W-HIST* ALGORITHM

In this section we introduce the *w-hist* algorithm, which improves the pruning capabilities of wireless k NN search algorithms with the use of simple histograms. Histograms have been profoundly used in databases for the purpose of selectivity estimation of range queries [20, 14, 17, 9]. Although various types of histograms have been introduced in the literature, we employ a grid like histogram that stores the number of objects located under each cell. Since our aim is to reduce the tune-in time, the histogram itself should be small enough not to increase the tune-in time, as it has to be read in order to extract useful information from it. Using complex histograms will not allow this.

The histograms can be used to obtain an upper bound on size of the result circle of a k NN query. Let A denote the area of the square region which contains all objects. We denote a histogram with cell size c as H_c . H_c partitions the area of interest as an equi-spaced grid with cell size equal to c . H_c has $\lceil \sqrt{A}/c \rceil^2$ square cells of size $c \times c$. A cell is denoted as $H_c\{i, j\}$, where i and j are cell indices. Each cell stores the

number of objects that lie under the cell's boundaries², denoted as $H_c[i, j]$. Figure 4 illustrates an example 7x7 histogram.

Note that given a query point P and any subset of k objects from the object set, the distance between P and the farthest object in the subset is larger than the radius of the result circle. As a result, any circle centered at point P that covers some set of histogram cells such that the total number of objects located under these histogram cells is larger than k , covers the result circle. More formally, let $C(P, r)$ denote a circle centered at point P with radius r and let r_* denote the radius of the result circle for k nearest neighbors of point P . Then we can state:

$$\text{If } \exists Q \text{ s.t. } \sum_{(i,j) \in Q} H_c[i, j] \geq k, \text{ and} \quad (1)$$

$$\forall (i, j) \in Q, H_c\{i, j\} \subset C(P, r)$$

Then $r \geq r_*$.

Consider that the histogram cells are sorted based on their distance from the query point P and are organized into a list. Then we define Q_+ to be the set of non-empty histogram cells ($H_c[i, j] > 0$) formed by picking cells from the sorted list until the sum of the cell contents reach or exceed k . Now we define the *pruning circle*, denoted as PC , which is used to prune nodes during the k NN search. The PC satisfies the left-side of (1) and is defined as follows: Pruning circle is the smallest circle centered at point P that covers the histogram cells in Q_+ . The radius of the pruning circle is denoted as r_+ .

The radius of the pruning circle, r_+ , can be calculated by a linear scan of the histogram cells, by keeping only $O(k)$ state. This is simply because the set Q_+ can be of at most size k . The closest k cells to the query point can be stored and maintained during the scan and this set can be pruned at the end of the scan to get Q_+ . Once we have Q_+ , the maximum of maximum-distances of cells in Q_+ to the query point P gives r_+ .

Figure 4 illustrates how r_+ is calculated for an example histogram and query point with $k = 4$. For the histogram and the query point in Figure 4, the four closest non-empty cells are the ones marked as c , b , e , and f (The first six cells, which yield four non-empty cells, according to their closeness to the query point are a , b , c , d , e , and f). The object count of the cell c (which is 1) and the sum of the object counts for the cells c and b (which is 3), are both smaller than 4. However, the sum reaches 5 when we add the cell e to the list. Thus we have $Q_+ = \{c, b, e\}$. The maximum distance of e to the query point is the largest among the cells in Q_+ . As a result it gives the value of r_+ .

Utilizing the histogram and the pruning circle extracted from it for a given query point, we can modify the wireless k NN search algorithm by adding the following pruning condition:

- When a newly discovered node is to be added to *ItemQueue*, say node N , it can be discarded if its mbr does not intersect with the pruning circle, i.e. if $N.mbr \cap PC = \emptyset$.

We name the optimized wireless algorithm which also uses the histograms as *the histogram supported optimized wireless*

²We assume objects are points, otherwise the center of each object can be used to determine which grid cell it belongs.

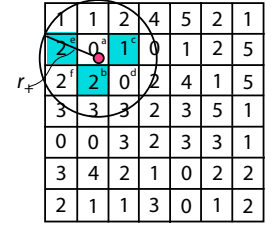


Figure 4: Example illustrating the usage of an histogram for $k = 4$

n	Number of objects
A	Area of the square region covering all objects
c	Histogram cell size
PC	Pruning circle
r_+	Radius of PC
N_i	Tree node at level i
l	Height of the tree
f	Minimum fanout of the tree
s_i	Average side length of a node mbr at level i
k	number of NNs searched
QS_{max}	Maximum size of <i>ItemQueue</i> during the search
AC_{tot}	Number of nodes/packets accessed i.e. tune-in time

Table 1: Notations used

algorithm, *w-hist* for short. In Section 8 we experimentally show that histograms are really effective in improving the tune-in time and decreasing the maximum size of the *ItemQueue*. We also examine the sensitiveness of different index serialization approaches to the use of histograms. In the rest of this section, we derive upper bounds on the expected size of *ItemQueue* and the expected number of total nodes accessed by the k NN search algorithm, when searching trees organized in *bfs* manner. The notations used in the rest of the section are summarized in Table 1.

6.1 Queue Size

For devices with scarce memory, like sensor devices [11], the memory requirement of the k NN search algorithm may become a restriction. In the context of spatial broadcast, work presented in [10] studied methods to bound the memory requirement for range searches on serialized spatial broadcast indexes. In our wireless k NN search algorithm, only structure that does not have a fixed size is *ItemQueue*. However, by using arbitrarily fine grained histograms we can limit the queue size. Theoretically, at one extreme case, the maximum queue size will take its minimum possible value, which is achieved when the pruning circle becomes identical to the result circle. We have the following upper bound for expected value of the queue size:

THEOREM 2. *Assuming a uniform data and query distribution, an upper bound on the expected maximum size (in terms of nodes) of the ItemQueue for a bfs serialized tree is given as a function of n, A, c, f, k as follows:*

$$E[QS_{max}] < \max_{i \in [1..l]} (f^i * \frac{s_{l-i}^2 + \pi * \delta^2 + 4 * \delta * s_{l-i}}{s_i^2 + \pi * \delta^2 + 4 * \delta * s_i}),$$

$$\text{where } l = 1 + \lceil \log_f \frac{n}{f} \rceil, \delta = 2 * \sqrt{(c^2 * \lceil \frac{k * A}{n * c^2} \rceil) / \pi},$$

$$\text{and } s_j = \sqrt{A/n} * (\sqrt{f^j} - 1)$$

Proof: Let us denote the number of items in the queue after all nodes at level $l - i + 1$ are processed as QS_i . Then the queue size can be considered as a branching process. Initially we have,

$$E[QS_0] = 1$$

To get an upper bound on the queue size, we will only consider pruning due to the pruning circle PC , thus we have:

$$E[QS_i] < f * P\{N_i.child.mbr \cap PC \neq \emptyset \mid N_i.mbr \cap PC \neq \emptyset\} * E[QS_{i-1}], \text{ for } i \in [1..l]$$

We have $P\{N_i.child.mbr \cap PC \neq \emptyset \mid N_i.mbr \cap PC \neq \emptyset\} = \frac{P\{N_i.child.mbr \cap PC \neq \emptyset\}}{P\{N_i.mbr \cap PC \neq \emptyset\}} = \frac{s_{i-1}^2 + \pi * r_+^2 + 4 * r_+ * s_{i-1}}{s_i^2 + \pi * r_+^2 + 4 * r_+ * s_i}$. Figure 5 shows

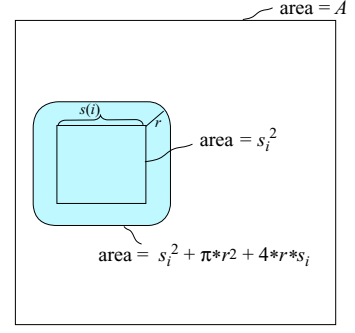


Figure 5: Node mbr at level i intersecting with the pruning circle with radius r

that for a fixed mbr, the query point should lie inside the shaded area in order for PC to intersect with the mbr. Then the derivation follows, as the probability that a randomly located circle with radius r intersects with a randomly located square of side length s_i is approximately³ equal to $(s_i^2 + \pi * r^2 + 4 * r * s_i) / A$. From induction we have,

$$E[QS_i] < f^i * \frac{s_{l-i}^2 + \pi * r_+^2 + 4 * r_+ * s_{l-i}}{s_i^2 + \pi * r_+^2 + 4 * r_+ * s_i}, \text{ for } i \in [1..l] \quad (2)$$

We have $l = 1 + \lceil \log_f \frac{n}{f} \rceil$ ([7]) and $s_i \approx \sqrt{A/n} * (\sqrt{f^i} - 1)$ ([27]). Average size of Q_+ is $\lceil \frac{k * A}{n * c^2} \rceil$ and we leave it to the reader to verify that $r_+ < \delta = 2 * \sqrt{(c^2 * \lceil \frac{k * A}{n * c^2} \rceil) / \pi}$. Integrating these into (2) proves the result. \square

6.2 Tune-in Time

Histograms can also be used to derive an upper bound on the number of nodes read by the algorithm, i.e. tune-in time in terms of packets. Similar to the result on maximum queue size, we have:

THEOREM 3. *Assuming a uniform data and query distribution, an upper bound on the expected number of packets/nodes read by the algorithm is given as follows:*

$$E[AC_{tot}] < hsize + \sum_{i=0}^{l-1} (f^i * \frac{s_{l-i}^2 + \pi * \delta^2 + 4 * \delta * s_{l-i}}{s_i^2 + \pi * \delta^2 + 4 * \delta * s_i}),$$

$$\text{where } hsize = \lceil 4 * \lceil \sqrt{A}/c \rceil^2 / \text{packet_size} \rceil \quad (3)$$

Proof: Assuming each cell of the histogram stores a 4byte integer and *packet_size* denotes the size of packets in bytes, size of the histogram is given by $\lceil 4 * \lceil \sqrt{A}/c \rceil^2 / \text{packet_size} \rceil$. The second term in the equation is derived in a similar way to Theorem 2, by assuming that only the pruning circle is used for pruning nodes of the tree during the search. \square

Note that increasing the histogram size may result in increasing the tune-in time as the histogram itself has to be read from the medium. In fact, using a histogram cell size smaller than the cell size that minimizes equation 3 will increase the tune-in time.

While the analytical results in this section are based on uniform data distribution, we evaluate real-life skewed data in our experimental results in Section-8.

7. SEARCH WITH NON-SPATIAL PREDICATES

The objects indexed by a spatial index can have non-spatial attributes that may need to be taken into account when answering queries [19]. For instance, going back to our example

³boundary conditions are not considered

scenario in Section 1, the query can ask only for k nearest *Chinese* restaurants instead of general restaurants. In this section we consider how to answer k NN queries that may specify an optional constraint on a single attribute, namely the *type* of the objects being queried.

We denote the number of distinct types as t , where out of n objects n_i of them belong to type i , thus $\sum_{i=1}^t n_i = n$. In this paper we only consider two types of k NN queries, (1) queries that do not specify a type and (2) queries that specify a single type. In order to support queries with type constraints, we need to modify both the search algorithm and the index organization. We look into two different methods:

i) *t-index*

In *t-index* method, we use t separate spatial indexes each indexing only its associated type. There is also a lookup structure that has pointers for each type pointing to the beginning of the index associated with that type. Although this is really efficient for queries with type constraints, for queries without constraints on types, it will require to lookup all indexes. In order to improve the performance of queries without type constraints, the order of the indexes can be selected such that an index corresponding to type i comes before the one corresponding to j on the broadcast medium if $n_i > n_j$. This enables us to restrict the solution space earlier, as the types with more number of objects tend to give a better approximation to the actual result.

ii) *t-hist/1-index*

In *t-hist/1-index* method, we use a single spatial index which indexes all objects. Although this is really efficient for queries without type constraints, it is not possible to search k nearest neighbors when a type constraint is present. To enable the processing of such queries, we also include t histograms on the broadcast, each one built for a particular type. There is also a lookup structure that has pointers for each type pointing to the beginning of the histogram associated with that type. Moreover the leaf nodes of the tree now also mark the type of each object. Queries with type constraints can now be processed on the index by only using the pruning circle derived from the associated histogram of the given type. In order to improve the performance of queries with type constraints, we can also prune a node whose mbr does not intersect with a non-empty cell of the histogram of the associated type. We name this latter optimization as *t-hist/1-index/hp* method. Note that it can improve the performance especially for the types that have small number of objects belonging to them.

In Section 8 we compare the performance of both the methods. Note that it is only fair to compare them when the total index size (together with histograms) occupied by the two methods is same. In other words, we should have $t * hsize + isize(n) \approx \sum_{i=1}^t isize(n_i)$, where $isize$ is size of the spatial index and is given as $isize(n) = \sum_{i=1}^{1+\lceil \log_f \frac{n}{f} \rceil} f^{i-1}$.

8. EXPERIMENTS

In this section we present our experimental results used to evaluate the methods introduced in this paper and understand tradeoffs involved in using them. We describe three sets of experiments. The first set of experiments investigate the improvement provided by *w-opt* search and *bfs* serialization of the index. The second set of experiments investigate the effect of using histograms on tune-in time and the sensitivity of different serializations to the usage of histograms. The third and final set of experiments compare *t-hist/1-index* and *t-index* approaches for k NN queries with type constraints. We use two

types of data in our experiments, (i) uniformly distributed set of points in unit square and (ii) a skewed dataset composed of city locations in Greece from [1] (scaled to fit unit square). The query points are selected at uniform random from the unit square. In all experiments, we employed an R*-tree [4] index.

8.1 *w-opt* Search and *bfs* Serialization

We describe experimental results with regard to effects of *w-opt* search and *bfs* serialization on tune-in time and queue size under three different scenarios: (1) varying packet(node) size, (2) varying number of objects, and (3) varying k .

8.1.1 Varying Packet Size

Figure 6 plots tune-in time (in terms of packets) and maximum size of *ItemQueue* (in terms of nodes) as a function of packet (node) size in bytes. In this experiment k is taken as 10 and 5000 uniformly distributed objects are used. The line labelled as *conventional* represents the k NN search algorithm on a *random access medium* and is used as a baseline. The results show that, *w-opt* algorithm shows up to 33% improvement in tune-in time for a *dfs* organization and up to 30% improvement in tune-in time for a *bfs* organization, when compared to *w-conv*. Another observation is that *bfs* organization provides up to 55% improvement over *dfs* organization. Also, even in the worst case the best tune-in time achieved using the best wireless search algorithm and organization (*bfs/w-opt*) is only twice the tune-in time of the baseline. The queue size results show that *bfs* organization have larger memory requirement when compared to *dfs*, but the *w-opt* algorithm helps decreasing the memory requirement for both organizations.

The differences in both tune-in time and maximum queue size values of different approaches vanish as the packet size increases to large values. This is because, for very large packet size values, the index fits into very small number of packets and ever algorithm ends up reading all the packets to reach appropriate nodes. If we increase the number of objects (thus the index size) in accordance with the increasing packet size, observations for small packet sizes still hold.

8.1.2 Varying Number of Objects

Figure 7 plots tune-in time and maximum size of the *ItemQueue* as a function of number of indexed objects (uniformly distributed). In this experiment k is taken as 10 and packet size is fixed to 1024 bytes. Clearly, the tune-in time will increase with increasing number of objects, since the total index size increases and there are more candidate nodes that need to be read from the medium. Notice that *w-opt* algorithm provides up to 40% improvement in tune-in time for *dfs* organization and up to 20% improvement for *bfs* organization. Furthermore, as the number of objects increase, the *bfs* organization shows an increasing advantage over *dfs* organization in terms of tune-in time (up to 53% improvement for *w-opt*). This shows our claim that in many scenarios, the *bfs* organization is actually a better choice. Also, Figure 7 shows that the *w-opt* algorithm provides up to 21% improvement in maximum queue size for *bfs* organization. The improvement in maximum queue size is marginal for *dfs* organization (around 3%). Comparing amongst the two organizations with *w-opt*, *dfs* organization provides up to 45% improvement in maximum queue size as compared to *bfs*.

Figure 9 plots the same measures with k set to 5. Again similar trends are observed, *w-opt* search and *bfs* organization prevailing over other configurations when tune-in time is considered. However, comparing Figure 9 and Figure 7 reveals that the improvement in tune-in time increases with increasing k . In fact one can prove that for NN search ($k = 1$) *w-opt*

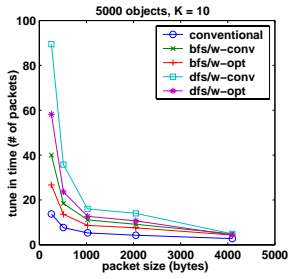


Figure 6: Tune-in time and maximum *ItemQueue* size for different packet sizes

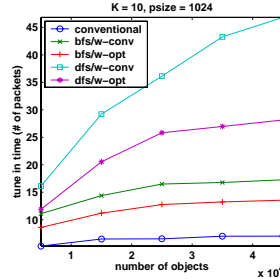
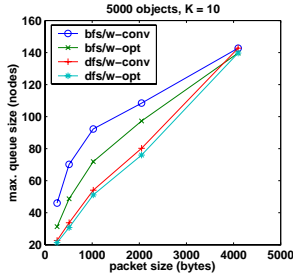


Figure 7: Tune-in time and maximum *ItemQueue* size for different number of objects when $k=10$

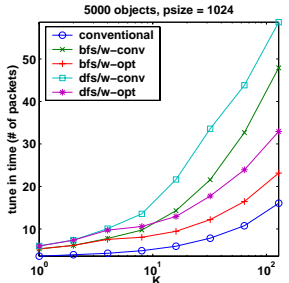
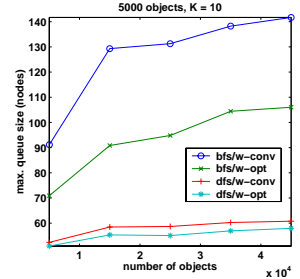


Figure 8: Tune-in time and maximum *ItemQueue* size for different k 's

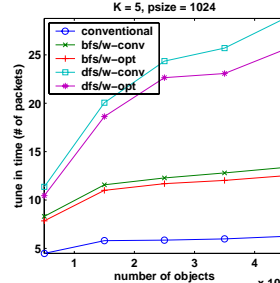
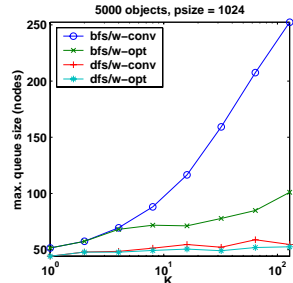


Figure 9: Tune-in time and maximum *ItemQueue* size for different number of objects when $k=5$

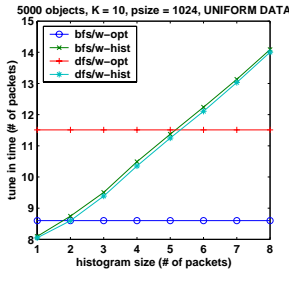
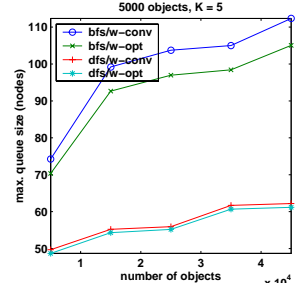


Figure 10: Tune-in time and *ItemQueue* size as a function of histogram size for uniform data

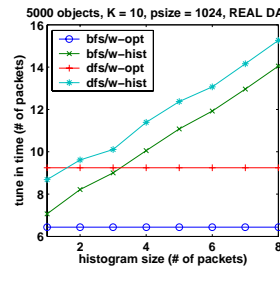
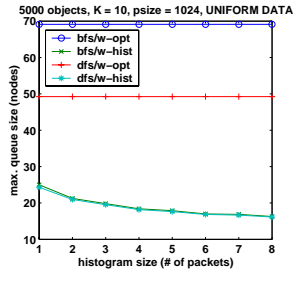


Figure 11: Tune-in time and *ItemQueue* size as a function of histogram size for city locations data

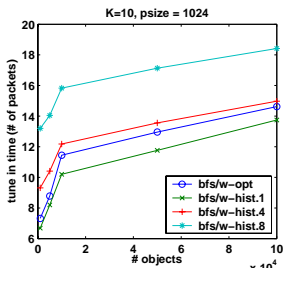
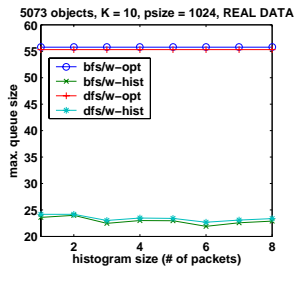


Figure 12: Tune-in time as a function of number of objects for different histogram sizes

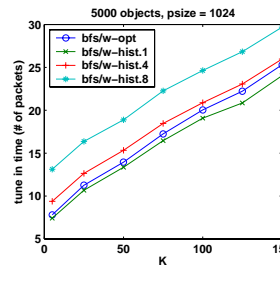
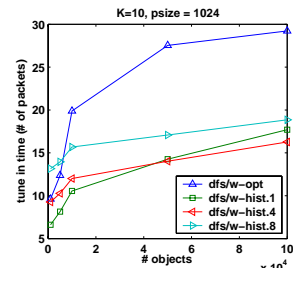


Figure 13: Tune-in time as a function of k for different histogram sizes

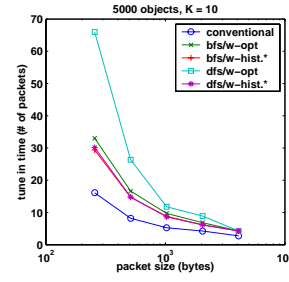
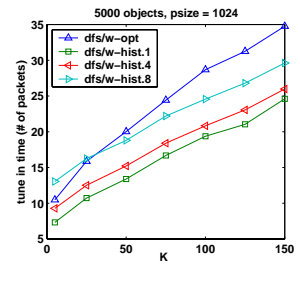


Figure 14: Tune-in time and maximum *ItemQueue* size for different packet sizes and approaches using no histograms or histograms with analytically derived sizes

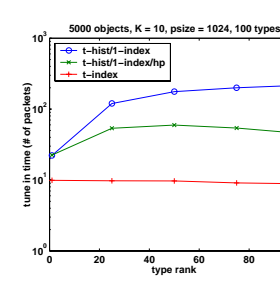
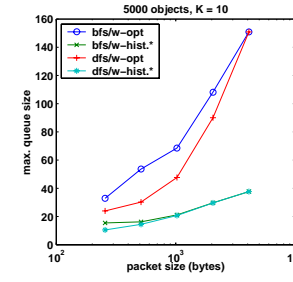


Figure 15: Tune-in performance for queries with type constraints

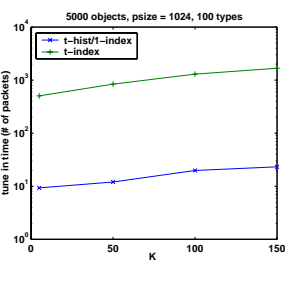


Figure 16: Tune-in performance for queries without type constraints

reduces to *w-conv*.

8.1.3 Varying k

Figure 8 plots tune-in time and maximum size of the *ItemQueue* as a function of k . In this experiment packet size is fixed to 1024 bytes and 5000 uniformly distributed objects are used. From the tune-in time graph we observe that, after $k = 4$ the *w-opt* algorithm starts to provide significant improvement over *w-conv* and after $k=8$ *w-opt* performs better than *w-conv* independent of the index serialization order. Queue size results show that memory requirement of *bfs* organization grows fast with increasing k when *w-conv* is used and significantly drops when *w-opt* is used. This shows that the use of *w-opt* algorithm is crucial for *bfs* organization, if the number of items requested is high.

8.2 Histograms and *w-hist* search

8.2.1 Varying Histogram Size

Figure 10 and Figure 11 plot tune-in time and maximum size of *ItemQueue* as a function of histogram size (in terms of packets). In these experiments k is taken as 10 and packet size is fixed to 1024 bytes. Figure 10 uses 5000 uniformly distributed objects and Figure 11 uses a data set consisting of real city locations (5073 points). The results from Figure 10 show that even a single packet histogram improves the tune-in time. In fact, for this experimental setup, histograms with larger sizes do not provide better tune-in times as the additional pruning capability achieved with the fine-graininess of the histogram cannot compensate the cost of reading the histogram itself. More interestingly, the effect of using a histogram is more prominent with the *dfs* organization and with histograms the best performance achieved with *bfs* and *dfs* organizations are effectively the same. Nevertheless, Figure 11 shows that with skewed data the latter observation no more holds and *bfs* organization with *w-opt* search outperforms all alternatives. This does not indicate that histograms are useless for skewed data sets. This is because Figure 11 also shows that a small histogram can significantly (more than 50%) decrease the queue size while only increasing the tune-in time marginally (around 5%).

8.2.2 Varying Number of Objects and k

Figure 12 plots tune-in time as a function of number of objects for different histogram sizes. Similarly, Figure 13 plots tune-in time as a function of k for different histogram sizes. Both experiments use 1024 byte packets and uniformly distributed objects. k is set to 10 in Figure 12 and 5000 objects are used in Figure 13. Figure 12 shows that for larger number of objects it is better to use larger histograms. It is also observed that *dfs* organization is more sensitive to the increase in the number of objects. As seen from the graph corresponding to *dfs*, using a 4 packet histogram becomes more efficient than using a single packet histogram as number of objects increase. And if we keep increasing the number of objects, according to the displayed trend, using an 8 packet histogram will become more efficient than using a 4 packet histogram. Although a similar phenomenon can be observed for *bfs* organization, the gap between tune-in times of approaches that use different histogram sizes change very slowly for *bfs* organization (no line crossings are observed for *bfs* organization in Figure 12).

8.2.3 Varying Packet Size

Figure 14 plots tune-in time and maximum size of *ItemQueue* as a function of packet size for four different approaches: *bfs/w-opt*, *dfs/w-opt*, *bfs/w-hist.** and *bfs/w-hist.**. Here *w-hist.**

corresponds to the case in which a histogram whose cell size minimizes Equation 3, is used. Again in this experiment k is taken as 10 and 5000 uniformly distributed objects are used. The figure shows that for uniform data distributions, Equation 3 can provide a histogram size that improves the tune-in performance.

8.3 Queries with Non-spatial Predicates

In order to compare *t-hist/1-index* and *t-index* methods for supporting evaluation of k NN queries with type constraints, we setup a scenario in which we have 100 different types. The number of objects belonging to each type follows a Zipf distribution with parameter 0.8 where the total number of objects is 5000. We perform two experiments using this scenario. In the first experiment we compare tune-in times of k NN queries with type constraints. Figure 15 plots the result as a function of type rank. Type with rank 1 has the most number of objects belonging to it. In the second experiment we compare tune-in times of k NN queries without type constraints. Figure 16 plots the result as a function of type rank.

As expected, Figure 15 shows that *t-index* performs much better for queries with type constraints. On the other hand Figure 16 shows that *t-index* performs poorly for queries without type constraints. Conversely, *t-hist/1-index* performs much better for queries without type constraints and poorly for queries with type constraints. In spite of *t-hist/1-index*'s poor performance for type constrained queries, *t-hist/1-index/hp* method, which is powered by the histogram pruning optimization described in Section 7, achieves significant improvement in tune in-time, especially for queries having constraints on infrequent types. Although it does not outperform *t-index* for type constrained queries, *t-hist/1-index/hp* shows a good balance between two types of queries and can be a good choice for workloads that contain sufficiently large number of queries without type constraints.

9. CONCLUSIONS

In this paper, we explored the issue of energy efficient k NN search on broadcasted R-tree indexes over location-dependent data. We proposed an optimization technique which improves the tune-in time of k NN search and discussed tradeoffs involved in organizing the index on the broadcast medium. Furthermore, we investigated the use of histograms as a technique to improve tune-in time and memory requirement of k NN search. We also studied the problem of k NN search with non-spatial constraints. We showed that:

- The introduced *w-opt* search technique significantly decreases tune-in time, irrespective of how the index is organized (*bfs* or *dfs*) on the medium.
- Organizing the index in *bfs* manner provides considerably better tune-in time but has a higher memory requirement due to queue size.
- Using histograms can further improve tune-in time for uniformly distributed data with the improvement being more for *dfs* organization in comparison to *bfs* organization.
- On skewed data sets, histograms improve the tune-in time for *dfs* organization. For *bfs* organization they significantly cut the maximum queue size in return for a minor increase in tune-in time.
- The use of histograms can be extended to support answering k NN queries with *type* constraints, which yields

better performance when compared to building separate indexes for each type, for workloads containing sufficiently large number of queries without type constraints.

As future work, we plan to continue working on energy-efficient spatial data broadcast on wireless environments, especially on the issues of caching and pre-fetching in the context of continuous and adaptive k NN search for moving points.

10. REFERENCES

- [1] City locations data set. <http://www.rtreeportal.org/>, October 2003.
- [2] S. Acharya, R. Alonso, M. J. Franklin, and S. Zdonik. Broadcast disks: data management for asymmetric communications environment. In *SIGMOD*, 1995.
- [3] S. Acharya, M. J. Franklin, and S. B. Zdonik. Disseminating updates on broadcast disks. In *VLDB*, 1996.
- [4] 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.
- [5] M.-S. Chen, P. S. Yu, and K.-L. Wu. Indexed sequential data broadcasting in wireless mobile computing. In *ICDCS*, 1997.
- [6] K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Experiences of developing and deploying a context-aware tourist guide: The GUIDE project. In *MobiComm*, 2000.
- [7] C. Faloutsos, T. Sellis, and N. Roussopoulos. Analysis of object oriented spatial access methods. In *SIGMOD*, 1987.
- [8] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.
- [9] M. Hadjieleftheriou, G. Kollios, and V. Tsotras. Performance evaluation of spatio-temporal selectivity estimation techniques. In *SSDBM*, 2003.
- [10] S. Hambrusch, C. L. W. Aref, and S. Prabhakar. Query processing in broadcasted spatial index trees. In *SSTD*, 2001.
- [11] J. Hill, R. Szweczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J.Pister. System architecture directions for networked sensors. In *ASPLOS*, 2000.
- [12] Q. Hu, D. L. Lee, and W.-C. Lee. Performance evaluation of a wireless hierarchical data dissemination system. In *MobiComm*, 1999.
- [13] T. Imielinski, S. Viswanathan, and B. Badrinath. Energy efficient indexing on air. In *ICDE*, 1994.
- [14] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD*, 1995.
- [15] D. Lee, W. Lee, J. Xu, and B. Zheng. Data management in location-dependent information services: Challenges and issues. *Pervasive Computing*, 1(3), 2002.
- [16] M. L. Lee, W. Hsu, C. S. Jensen, B. Cui, and K. L. Teo. Supporting frequent updates in R-trees: A bottom-up approach. In *VLDB*, 2003.
- [17] M. Muralikrishna and D. DeWitt. Equi-depth histograms for estimating selectivity factors for multidimensional queries. In *SIGMOD*, 1988.
- [18] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley, 2000.
- [19] D.-J. Park and H.-J. Kim. An enhanced technique for k -nearest neighbor queries with non-spatial selection predicates. *Multimedia Tools and Application*, 19(1), 2003.
- [20] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD*, 1996.
- [21] Q. Ren and M. H. Dunham. Using semantic caching to manage location dependent data in mobile computing. In *MobiComm*, 2000.
- [22] N. Roussopoulos and F. V. S. Kelley. Nearest neighbor queries. In *SIGMOD*, 1995.
- [23] S. Saltinis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD*, 2000.
- [24] T. Seidl and H. P. Kriegel. Optimal multi-step k -nearest neighbor search. In *SIGMOD*, 1998.
- [25] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+ -tree: A dynamic index for multidimensional objects. In *VLDB*, 1987.
- [26] N. Shivakumar and S. Venkatasubramanian. Energy-efficient indexing for information dissemination in wireless systems. *Journal of Mobile Networks and Nomadic Applications*, December 1996.
- [27] Y. Theodoridis, E. Stefanakis, and T. K. Sellis. Efficient cost models for spatial queries using r -trees. *TKDE*, 12(1):19–32, 2000.
- [28] J. Xu, B. Zheng, W. Lee, , and D. Lee. Energy efficient index for querying location-dependent data in mobile broadcast environments. In *ICDE*, 2003.
- [29] P. Xuan, S. Sen, O. Gonzalez, J. Fernandez, and K. Ramamritham. Broadcast on demand: Efficient and timely dissemination of data in mobile environments. In *Real-Time Technology and Applications Symposium*, 1997.
- [30] B. Zheng, W. Lee, and D. Lee. Search k nearest neighbors on air. In *MDM*, 2003.