Processing Transitive Nearest-Neighbor Queries in Multi-Channel Access Environments

Xiao Zhang*, Wang-Chien Lee*, Prasenjit Mitra*†, Baihua Zheng‡
*Department of Computer Science and Engineering, †College of Information Sciences and Technology
The Pennsylvania State University
‡School of Information Systems, Singapore Management University
{xiazhang, wlee}@cse.psu.edu, pmitra@ist.psu.edu, bhzheng@smu.edu.sg

ABSTRACT

Wireless broadcast is an efficient way for information dissemination due to its good scalability [10]. Existing works typically assume mobile devices, such as cell phones and PDAs, can access only one channel at a time. In this paper, we consider a scenario of near future where a mobile device has the ability to process queries using information simultaneously received from multiple channels. We focus on the query processing of the transitive nearest neighbor (TNN) search [19]. Two TNN algorithms developed for a single broadcast channel environment are adapted to our new broadcast enviroment. Based on the obtained insights, we propose two new algorithms, namely Double-NN-Search and Hybrid-NN-Search algorithms. Further, we develop an optimization technique, called approximate-NN (ANN), to reduce the energy consumption in mobile devices. Finally, we conduct a comprehensive set of experiments to validate our proposals. The result shows that our new algorithms provide a better performance than the existing ones and the optimization technique efficiently reduces energy consumption.

Keywords

Multi-Channel access, transitive nearest neighbor, query processing, query optimization, approximate nearest neighbor

1. INTRODUCTION

Wireless broadcast has been used widely in various applications (e.g., TV, Radio and GPS). It efficiently uses limited bandwidth to facilitate information dissemination to an arbitrary number of users simultaneously. This feature has attracted a lot of interests and effort from the research community to develop wireless data broadcast techniques, such as [10], [7] etc., in the past decade.

Most prior research on wireless data broadcast assumes that a mobile device can only monitor and receive data from one channel at a time. Given data broadcast in multiple channels on air, a mobile device has to "switch" among channels in order to receive data from other channels. However, with technological advances, this assumption no longer holds. In the near future (even today), mobile devices (e.g.,a portable equipped with multiple wireless radio interfaces or a dual-mode/dual-standby cell phones) will be able to access to multiple channels simultaneously [1]. In this paper, we assume a mobile device has the ability to process queries using the information simultaneously received from multiple channels and focus on the query processing of transitive nearest neighbor (TNN) search - a new query type that involves multiple datasets [19]. To the best of our knowledge, this is the first research on query processing over a simultaneous access of multiple broadcast channels.

Here, we first define transitive nearest neighbor (TNN) search:

Given a query point p, and two datasets S and R, TNN returns a pair of objects $(s, r) \in S \times R$ such that $\forall (s', r') \in S \times R$, $(dis(p, s) + dis(s, r)) \leq (dis(p, s') + dis(s', r'))$ where $dis(p, s)$ represents the Euclidean distance between the two points p and s. S and R correspond to two types of spatial objects distributed in a certain area.

Many applications of TNN queries exist in our daily life. For example, Mr. Smith is new to a city and he wants to find a post office to send his friends some post cards first and then go to a restaurant to have dinner. TNN gives him a post office and a restaurant with minimal total travel distance.

Two TNN algorithms, namely *Approximate-TNN-Search* algorithm and *Window-Based-TNN-Search* algorithm[19], have been proposed in the one-channel wireless broadcast environment. However, the *Approximate-TNN-Search* algorithm may fail to answer TNN queries over real datasets, while the *Window-Based-TNN-Search* algorithm was not designed for simultaneous access to multiple channels. Thus, our research goal is to devise efficient algorithms for multiple channel broadcast environments. The TNN algorithms we developed is based on an *estimate-filter* query processing paradigm, which consist of two phases: 1) determine/estimate a search range that contains all qualified data objects; and 2) filter unqualified objects from this search range. (see Section 3.1 for more details and example) We propose new algorithms, namely, *Double-NN-Search* algorithm and *Hybrid-*

[1]A dual-mode dual-standby cell phones [3] allows its users to stay on-line and send/receive signals in both GSM and CDMA networks simultaneously.

EDBT'08, March 25–30, 2008, Nantes, France.

NN-Search algorithm. They use different strategies in phase 1 and introduce more parallelism than *Window-Based-TNN-Search* in query processing. These algorithms employ *exact nearest neighbor* (eNN) search algorithm to obtain a search range (during the estimate phase). Our research shows that instead of eNN search, an approach based on *approximate nearest neighbor (ANN)* search may reduce the overall energy consumption. We propose a new heuristic to perform approximate nearest neighbor search on R-tree. The heuristic is dynamically optimized based on R-tree heights and sizes of the two datasets involved in the TNN query. We also conduct a comprehensive set of experiments to show the effectiveness of these algorithms and optimization techniques.

The contributions of this study are summarized as follows:

- We propose two algorithms for processing TNN queries in a wireless broadcast setting where mobile devices can simultaneously access multiple channels. These algorithms significantly reduce the access time of TNN queries.
- We propose two new distance metrics (*MinTransDist* and *MinMaxTransDist*) which allow our new algorithms to effectively reduce the search range.
- We propose an optimization technique for TNN query processing which uses approximate-NN search on R-tree to substitute exact-NN in the original algorithms. We also propose a dynamic approach to determine the parameter of the approximation. This optimization technique effectively reduces the energy consumption, w.r.t. using eNN.

The rest of the paper is organized as follows. Section 2 discusses the preliminaries and related work. Section 3 discusses the principles in answering TNN queries and introduces two existing algorithms in details and analyzes their deficiencies. Section 4 proposes our new algorithms and gives the definitions of new distance metrics. Section 5 introduces an efficient query optimization technique and proposes several heuristics to reduce the search cost. A comprehensive evaluation of these algorithms and the optimization technique is given in Section 6, and finally, Section 7 concludes the paper and gives a roadmap of future work.

2. PRELIMINARY

In this section, we first describe a system model of multiple channel wireless broadcast to introduce the system components, basic concepts and constraints of wireless broadcast environments. We then review the R-tree index, a fundamental spatial index structure which is closely related to our work, and some related spatial queries.

2.1 System Model

There are two different information access mechanisms available in wireless broadcast environments for mobile users: 1) *on-demand access* and 2) *broadcast access*. In the *on-demand access* scheme, a mobile client sends a query to the server through a dedicated point-to-point wireless channel. The server, which is responsible for processing the query, returns the answer directly to the mobile client. On the other hand, in *broadcast access* scheme, the server broadcasts the data in publicly available wireless channels in the unit of

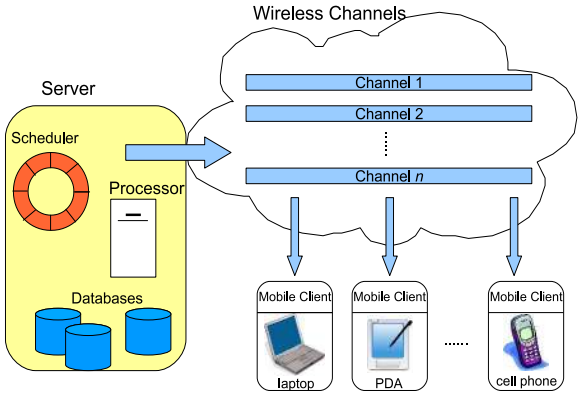


Figure 1: System Model

page (or packet, frame). The mobile client, after receiving a query from the user, tunes into the broadcast channels and downloads the interested pages. It then processes the query locally and finally returns the result to the user. Compared to the on-demand access mode, *broadcast* scheme conserves the limited resource of wireless bandwidth by allowing an arbitrary number of users to access the same information simultaneously. This scheme can also address the privacy issues in location-based services, since the clients can obtain location-based information without disclosing their own locations to the server.

In this paper, we focus on TNN query processing in the broadcast scheme. The system model of our study consists of three major components: 1) the server, 2) the mobile clients, and 3) the wireless broadcast channels. The server maintains and disseminates information (in terms of data objects) of interest to users. It is responsible for scheduling the broadcast program. The mobile client is location-aware (e.g., equipped with GPS)[2]. Upon receiving a TNN query from its user, a mobile client tunes into the wireless channels and downloads data from multiple channels simultaneously to facilitate query processing. Figure 1 gives a high-level view of our system model.

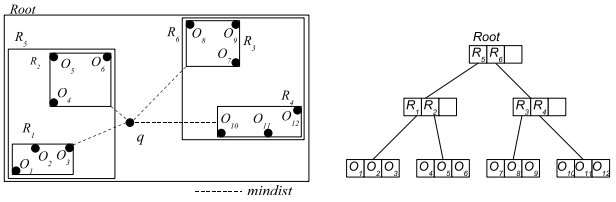
In broadcast channels without any index information, a mobile client has to stay active and downloads all the data objects to process a query. This approach is inefficient as it consumes a lot of energy. Energy consumption is a major concern due to the limited power source of the mobile clients. Imielinski, Viswanathan, and Badrinath proposed *air indexing* to tackle this issue. Air indexing interleaves index information with data objects in the wireless channels, which allows a mobile client to first probe the index to get the arrival time of the interested data objects, and then turn into doze mode to save energy. When the desired data is about to be broadcast, the mobile client wakes up and tunes into the broadcast channel to download the data. $(1, m)$ is a representative interleaving technique [10], which divides the dataset into m equal-sized fractions and broadcasts the entire index preceding every one of the m fractions.

We use two performance metrics: *access time* and *tune-in time* [7, 9, 10] to measure the effectiveness of our algorithms. The former is the time elapsed from the moment a query is

[2]In this paper, we use these the terms mobile devices and mobile clients interchangeably.

issued till the moment the query is satisfied. Obviously, users want the answer as fast as possible; thus the access time needs to be short. The latter is the time a mobile client stays active to receive the required data for query processing, which is used to represent the energy consumption in the literature. Query processing in broadcast schemes aims at answering a query within a short time, with small energy consumption. Since *access time* and *tune-in time* are both proportional to the number of pages accessed, we measure the two metrics in the number of pages accessed during the query processing.

2.2 Related Work



(a) Objects and MBRs (b) R-tree Index Structure

Figure 2: NN Search on R-tree

R-tree index, first proposed by Guttman [5], is a well-known spatial index. It recursively groups nearby objects into minimal bounding rectangles (MBRs) until the whole region is covered by one root node. Figure 2(a) depicts the MBRs of a rectangular region covering 12 objects. Assuming the fanout of the R-tree node is three, the corresponding R-tree structure is depicted in Figure 2(b). If the objects are available a priori, packing algorithms, such as Nearest-X [16], Hilbert Sort [11] and STR [12], can be applied to build the R-tree in order to achieve the best performance.

R-tree can support multiple queries efficiently. Take window queries as an example. The search starts at the root node, and follows those nodes with MBRs overlapping with the queried window. Often, NN searches use a branch-and-bound strategy to traverse the R-tree[4, 6, 15]. Different heuristics based on some distance metric are proposed to refine the search range and prune the search space. Among all the algorithms, *Best-First (BF)* algorithm [6] is most efficient. It maintains a priority queue of candidate nodes, sorted according to ascending order of*mindist*, the minimal distance to a query point.

However, the performance of the *Best-First* algorithm deteriorates severely in terms of access time in the context of broadcast because we are using a physically linear media (wireless broadcast channels) to represent a tree structure of R-trees. In broadcast, data object/index information is only available when it is on the air. Once the packet containing the desired information is missed, the mobile client has to wait until the next time the object is broadcast again. In other words, random access, which is commonly supported in resident storage (e.g., disk and memory), is not allowed in wireless broadcast systems. The performance of the best-first algorithm, which employs backtracking to guarantee the effectiveness, deteriorates severely when data are broadcast. Figure 3 gives one example. It assumes that the R-tree is broadcast in a preorder traversal in a broadcast cycle. R_2, due to a smaller *mindist* to q, is visited first. Then, R_1 is accessed. However, it is already broadcast and is not available until the next time it is broadcast. The access time is prolonged significantly. Therefore, the search algorithms must take into consideration the linear property of broadcasting.

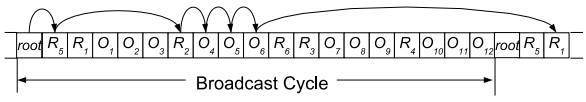


Figure 3: Linear Access in Broadcast Channel

In [19], two algorithms were proposed for answering TNN queries in dingle-channel broadcast environments, namely, *Window-Based-TNN-Search* algorithm and *Approximate-TNN-Search* algorithm. The algorithms and their problems will be discussed in detail in the next section. Researchers have studied two other general versions of TNN: *optimal sequenced route* (OSR) [17] query and *trip planning query* (TPQ) [13]. Given a set of points P where each point belongs to a specific category, a starting point p and a destination e, OSR finds a route of minimum length starting from p passing through at least one point from each category in a specified order, and ends at e; while TPQ finds the shortest route regardless of the order. However, these queries were studied in disk-based databases, not in the wireless broadcast environments.

Lin et al., [14] introduced several heuristics to perform ANN search on R-trees. However, the *threshold* in their work, which is an important parameter in approximation, is static. Their algorithms did not take into consideration several factors, such as R-tree height, that will affect the performance. Besides, the goal of the previous work was different from ours. They did not aim to minimize the search cost.

In the mobile database literature, research on object retrieval and object organization in multi-channel broadcast environments have been reported. Sun, et al., [18] proposed two algorithms to retrieve objects given the distribution of desired pages on parallel broadcast channels using a minimum number of passes and switches (in order of priority) between the channels. Hurson et al. proposed two algorithms to distribute objects of different sizes onto parallel broadcast channels to achieve the minimum broadcast cycle length and to preserve the clustering property [8]. However, they assumed that a mobile client can only access one channel at any one time. Neither did they deal with query processing in broadcast environments.

3. PROBLEM ANALYSIS

In this section, we first discuss the basic ideas in answering TNN queries and the two TNN algorithms proposed in [19]. A natural question arises is whether these ideas can be adapted for the multi-channel access environment. We use an example to show how they work. Then, we adapt these algorithms into our new multi-channel access environment and perform an analysis.

3.1 Answering TNN Query

First, we use a running example to illustrate how TNN is answered in traditional broadcast environment in which a client can tune into one channel at one time. Given two datasets S and R, with S={s_1, s_2, s_3, s_4} and R={r_1, r_2,

Notation	Description
$dis(p,s)$	Euclidean distance between points p and s
$p.\mathrm{NN}(S)$	The nearest neighbor of point p in dataset S
$circle(p,d)$	A circle with point p as the center and a radius of length d
$p.TNN(S,R)$	The answer pair of objects in datasets S and R of the TNN query with point p as the query point

Table 1: Terminology Definition

r_3, r_4}, we assume two broadcast channels co-exist on air, with one broadcasting dataset S and the other broadcasting dataset R. R-trees are used to index the data points.In broadcast environment, R-tree pointers refer to the arrival time of the data pages.We assume that the broadcast program is organized according to $(1, m)$ scheme, i.e., an R-tree for the dataset (e.g., S or R) is broadcast m times inside one broadcast cycle. In order to facilitate the following description, Table 1 lists the notations and their definitions.

A brute force approach can simply retrieve all the objects from datasets S and R and evaluate each pair of objects $(s, r) \in S \times R$. The pair giving the minimum transitive distance is returned as the result. However, this approach obviously is inefficient. It requires the mobile client to download all the data objects, thus results in a large storage requirement in the client as well as a large tune-in time. Since a TNN query only cares about one pair of objects with the minimal transitive distance, efficient algorithms should avoid the retrieval of any unnecessary objects. Zhen, Lee and Lee suggested an *estimate-filter* query processing paradigm [19]:

Estimate: find a search range from the query point p by searching the index;

Filter: filter unqualified data objects in the search range determined earlier to find the pair of objects with the minimum transitive distance.

After determining the final answer, the mobile client may later retrieve the desired data object and its associated attributes. An important goal for the design of TNN algorithms is to obtain a small search range in the estimate phase in order to reduce the tune-in time in the filter phase. Meanwhile, we devise optimization techniques (see Section 5) to reduce the tune-in time in the estimation phase without increasing the tune-in time in the filtering phase.

In the estimate phase, the proposed algorithms differ most significantly in how they determine the search range. In the filter phase, two range queries are issued to retrieve all objects in S and R in the search range. After the locations of these objects are obtained, a join is computed to find the minimum transitive distance and the corresponding pair of objects. Finally, the mobile client turns into sleep mode and wakes up when the pair of answer objects are on air to retrieves the attributes associated with the objects. Because the size of the search range has a direct impact on the number of objects retrieved, the search range should be small to reduce the tune-in time. However, it also should be large enough so that it includes the answer pair. Zheng, Lee and Lee [19] have provided the following theorem that can be used to help prune the search range.

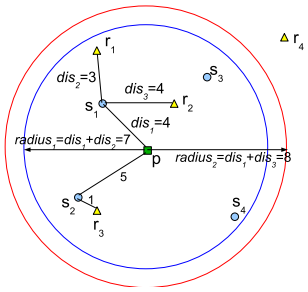


Figure 4: Search Range Determination

THEOREM 1. *Given a query point p and a pair of objects (s, r) ∈ S×R, let d = dis(p, s) + dis(s, r). If s′ ∉ circle(p, d) with s′ ∈ S, it is guaranteed that s′ ∉ p.TNN(S,R). Similarly, if r′ ∉ circle(p, d) with r′ ∈ R, it is guaranteed that r′ ∉ p.TNN(S,R).*

The **Window-Based-TNN-Search** algorithm determines the search range by issuing two nearest neighbor queries. The first NN query finds p's nearest neighbor s in S, i.e. $s = p.\mathrm{NN}(S)$. Then it issues the second nearest neighbor query to find s's nearest neighbor r in R, i.e. $r = s.\mathrm{NN}(R)$. The transitive distance $d = dis(p, s) + dis(s, r)$ is used as the search radius. Then two window queries are issued at p to retrieve objects from S and R. Finally, a join is applied to find the minimum transitive distance and the corresponding pair of objects. The following example shows how this approach works.

Suppose a TNN query is issued at point p as shown in Figure 4. The Window-Based-TNN-Search algorithm first issues an NN query to retrieve the nearest neighbor s_1 of p in S. Then it issues the second NN query to retrieve the nearest neighbor r_1 of s_1 in R. The search range is therefore determined to be $circle(p, d)$, where $d = dis(p, s_1) + dis(s_1, r_1)$ (the inner circle in Figure 4). Then two window queries are issued and objects (s_1, s_2, s_3, s_4) and (r_1, r_2, r_3) are scheduled to be retrieved. The transitive distances between p and different combinations of object pair (s, r) are calculated and compared. The final answer (s_2, r_3) with the minimum transitive distance is then output.

The **Approximate-TNN-Search** algorithm performs an approximate estimate of search range at the estimate phase. This algorithm is very efficient but does not guaranteed to produce the correct answer. The window-based TNN algorithm needs two index traversals to determine the search range. However, the approximate-TNN-Search algorithm saves the two NN queries by determining the search range using the following equation [19]:

$$r_k(S) = \ln(n) \times \sqrt{\frac{k}{(\pi \times n)}}, \text{where } n = |S|. \qquad (1)$$

Given a dataset S in which the points are uniformly distributed in a unit square region, a circle with radius $r_k(S)$ encloses at least k objects. Therefore, the radius of the search range in a TNN query can be derived by $d = r_k(S) + r_k(R)$, where $k = 1$. The radius can be easily scaled to a square of other size. This approach saves two index traversals to allow the algorithm to go directly into the filter phase and only introduces a small computational overhead. How-

ever, it does not guarantee to contain the answer in the approximated search range.

3.2 Deficiencies of Existing Algorithms

The algorithms mentioned above are sequential in nature and do not utilize the fact that the two datasets may be broadcast on two channels simultaneously. Since a mobile device may monitor and download pages from both channels simultaneously, the two NN queries can be processed in parallel. The two range queries in the filter phase can be processed in a similar fashion.

The deficiency of the two algorithms mentioned above are as follows. For the Window-Based-TNN-Search algorithm, the second NN query takes the output of the first NN query as its query point. As a result, it must wait until the first NN query is finished. For the approximate-TNN-Search algorithm, the equation mentioned above only suits datasets where the data points are uniformly distributed. For skewed datasets, this approach fails to determine a search range that guarantees to contain the final answer pair of objects (see Section 6.3). Besides, even with uniformly distributed datasets, the search range provided by the equation is unnecessarily large. More objects are enclosed in this range and more pages have to be downloaded from the broadcast channels, compared to other TNN algorithms. As a result, the tune-in time of the approximate-TNN-Search algorithm is very large even if it avoids the two NN queries in the estimate phase. In next section, we propose new algorithms, namely, *Double-NN-Search* algorithm and *Hybrid-NN-Search* algorithm, which not only guarantee correct answers but also fully exploit the parallel access ability of the mobile device.

4. NEW ALGORITHMS FOR TNN

In this section, we introduce two new algorithms, namely the Double-NN algorithm and the Hybrid-NN algorithm. Both algorithms allow parallel access of broadcast data in both phases to save access-time. Our algorithms find a pair of objects (s, r) and use the transitive distance determined by p and (s, r) as the search range.

4.1 Double-NN-Search Algorithm

The algorithm executes two nearest neighbor queries from the query point p on the two channels at the earliest opportunity, i.e., as soon as the index roots appear in the two channels. After both NN queries are completed, it then uses $d = dis(p, s) + dis(s, r)$ as the radius of the search range, where $s = p.\mathrm{NN}(S)$ and $r = p.\mathrm{NN}(R)$.

In the running example shown in Figure 4, Double-NN-Search algorithm retrieves s_1 (i.e. $p.\mathrm{NN}(\text{S})$) and r_2 (i.e. $p.\mathrm{NN}(\text{R})$). Thereafter, the search range is fixed to *circle*(p, d), where $d = dis(p, s_1) + dis(s_1, r_2)$ (the outer circle). Then two range queries are issued to access data in both channels in this range. Objects (s_1, s_2, s_3, s_4) and (r_1, r_2, r_3) are retrieved and the final answer pair (s_2, r_3) is obtained by calculating and comparing the transitive distances. The pseudo code of the Double-NN-Search algorithm is given in Algorithm 1.

Algorithm 1 Double-NN-Search Algorithm

Input: query point p, R-tree index S for dataset S, R-tree index R for dataset R
Output: transitive nearest neighbor (s, r) to the query point p
Procedure:

```
1: candidate_set_S ← ∅;
2: candidate_set_R ← ∅;
3: s ← p.NN(S); r ← p.NN(R); {these two nearest neighbor queries are processed in parallel}
4: d ← dis(p, s) + dis(s, r);
5: let circle w center at p with radius length d;
6: candidate_set_S ← p.range_query(S, w);
   candidate_set_R ← p.range_query(R, w);
   {these two range queries are processed in parallel}
7: for each object s_i in candidate_set_S do
8:   if dis(p, s_i) < d then
9:     for each object r_j in candidate_set_R do
10:      if dis(p, s_i) + dis(s_i, r_j) < d then
11:        d ← dis(p, s_i) + dis(s_i, r_j);
12:        s ← s_i;
13:        r ← r_j;
14:      end if
15:    end for
16:  end if
17: end for
18: return (s, r);
```

4.2 Hybrid-NN-Search algorithm

In Hybrid-NN-Search algorithm, when the NN search in one of the two channels is completed before the other, we can use the result to guide the search in the unfinished channel in order to find a smaller search range. Before describing the algorithm, first, we define some distance metrics that will be used in our new algorithm and then discuss the Hybrid-NN algorithm in detail.

4.2.1 Useful Distance Metrics

The two metrics, introduced in this sub-section, act on two points (p and r) and a MBR (M_S) of an R-tree node. Both metrics find some transitive distance from point p, to a point on the MBR M_S, then to point r.

DEFINITION 1 (MINTRANSDIST). *Given a starting point p, a MBR M_S, an ending point r, MinTransDist(p, M_S, r) finds a point s on MBR M_S and returns dis(p, s)+dis(s, r), such that for any point s' on MBR M_S other than s, dis(p, s')+dis(s', r)≥MinTransDist(p, M_S, r).*

MinTransDist gives the minimum possible transitive distance from a point to an MBR then to another point, which can be used as the lower bound of the transitive distance if a point from the MBR is chose. We derive a method to calculate *MinTransDist* as follows. There can be three cases:

1. If line segment pr (line between p and r) intersects with MBR M_S, *MinTransDist(p, M_S, r)=dis(p, r).*

2. If not in 1, let ℓ_i ($1 \le i \le 4$) be the four sides of MBR M_S, if p and r are on the same side of ℓ_i, find r'_i which is the symmetric point of r w.r.t. ℓ_i. If line segment pr'_i intersects with ℓ_i, then *MinTransDist(p, M_S, r)*=$\min_i\{dis(p, r'_i)\}$.

3. If not in 1 or 2, let v_i ($1 \le i \le 4$) be the four vertices of MBR M_S, *MinTransDist(p, M_S, r)*=$\min_{1\le i\le 4}\{dis(p, v_i) + dis(v_i, r)\}$.

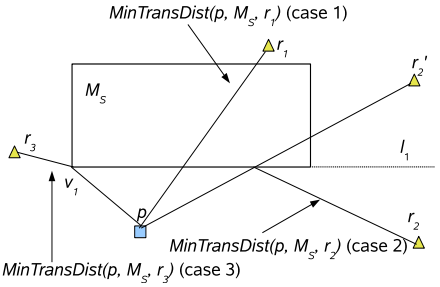


Figure 5: Distance Metrics

LEMMA 1. *The above method gives the correct MinTransDist from p to M_S to r.*

PROOF. The completeness and the soundness of Case 1 and Case 2 are obvious. In Case 3, an ellipse is determined which uses p and r as two foci and the transitive distance as the major axis. Suppose v_1 gives the minimum transitive distance. v_2, v_3, v_4 are outside of the ellipse. Since Case 1 and Case 2 does not hold, any points on the four sides of M_S other than v_1 are outside of the ellipse. Then any points inside of M_S are outside of the ellipse. Thus, $dis(p, v_1)+dis(v_1, r) < dis(p, u)+dis(u, r)(u \in M_S, u \neq v_1)$. Since $v_1 \in M_S$, this method gives a tight lower bound. □

Figure 5 gives an example of three minimum transitive distances in each of the three cases.

DEFINITION 2 (MAXDIST). *Given a starting point p, a line segment ℓ, an ending point r, $MaxDist(p, \ell, r)$ is defined as:*

$$MaxDist(p, \ell, r) = \max_{i=1,2}\{dis(p, v_i) + dis(v_i, r)\} \quad (2)$$

where v_i $(i = 1, 2)$ are the two end points of line segment ℓ.

LEMMA 2. *given a starting point p, a line segment ℓ and an ending point r, $MaxDist(p, \ell, r)$ gives a tight upper bound for all the transitive distances from p to any points on ℓ, to r.*

PROOF. An ellipse is determine, which uses p and r as the foci and $MaxDist(p, \ell, r)$ as the length of the major axis. One end point of ℓ is on the ellipse and the other is either on or inside of it. Therefore all other points $v \in \ell$ are inside of this ellipse, and thus making $dis(p, v) + dis(v, r) < MaxDist(p, \ell, r)$. Since one of the two endpoints provides the transitive distance equal to $MaxDist(p, \ell, r)$, this upper bound is tight. □

DEFINITION 3 (MINMAXTRANSDIST). *Given a starting point p, an MBR M_S, an ending point r, MinMaxTransDist(p, M_S, r) is defined as:*

$$MinMaxTransDist(p, M_S, r) = \min_{1 \le i \le 4}\{MaxDist(p, \ell_i, r)\} \quad (3)$$

where $\ell_i (1 \le i \le 4)$ are the four sides of MBR M_S.

LEMMA 3. *given a starting point p, an MBR M_S enclosing a point dataset S, an ending point r, $\exists s \in S, dis(p, s) + dis(s, r) \le MinMaxTransDist(p, M_S, r)$*

PROOF. The definition of *MinMaxTransDist(p, M_S, r)* uses the *MBR face property*[15]. This property means that every face of MBR of an R-tree node contains at least one point in the point dataset. $\forall v \in \ell$ (ℓ is a side of MBR M_S), the transitive distance $dis(p, v) + dis(v, , r)$ is bounded by *MaxDist(p, ℓ, r)*, according to Lemma 2. Since *MinMaxTransDist* chooses the smallest one among the four sides, it provides a tight upper bound. □

From the previous definitions and lemma, we can deduce that given a starting point p, a rectangle M_S, an ending point r, *MinTransDist(p, M_S, r)* and *MinMaxTransDist(p, M_S, r)* serve as lower and upper bounds, respectively, of transitive distance from p to M_S to r.

4.2.2 Algorithm Description

Since datasets of different sizes may be involved in the TNN query, the lengthes of the indices for them are different. It is very likely that the two NN search in the estimate phase will finish at different time. When the NN search in one channel generates the final result before the other one, hybrid algorithm uses this result to guide the unfinished search in the other channel to get a result that can minimize the transitive distance between the three points, in order to get a smaller search range for the next phase. Hybrid-NN algorithm achieves this goal by either switching the query point of the NN search in channel 2 or changing the distance metrics in the search in channel 1.(note that the transitive distance measure is not symmetric with respect to the two channels). In Figure 4 if the NN search on dataset R finishes before the NN on dataset S, and r_2 is returned as the NN to p, Hybrid-NN returns s_3, which can minimize the transitive distance from p to s_3 to r_2, instead of s_1. Three cases exist and the behavior of Hybrid-NN algorithm is described below:

1. If NN searches in both channels are not finished. Follow the algorithm as Double-NN algorithm.

2. If the NN search in Channel 1 finishes before the NN search in Channel 2, let s be the result of NN search in Channel 1, i.e. $s = p.NN(S)$. Use s to replace p to be the new query point of the NN search in Channel 2 and find $r \in R$, which is the nearest neighbor to s on the remaining portion of the R-tree for dataset R.

3. If NN search in Channel 2 finishes before the NN search in Channel 1, let r be the result of NN search in Channel 2, i.e. $r = p.NN(R)$.Change the distance metrics in the NN search in Channel 1. Use *MinTransDist* and *MinMaxTransDist* to perform branch-and-bound search on the remaining part of the R-tree for dataset S. Find $s \in S$ which gives the minimum transitive distance from p to s to r on the remaining portion of R-tree.

4.2.3 Updating and Pruning Strategy

Like traditional NN algorithms, Hybrid-NN algorithm keeps two upper bounds for the NN queries on each channel and updates them during the traversal of R-trees. These upper bounds are used to prune the R-tree nodes that do not contain the answer. Hybrid-NN algorithm uses two priority queues to keep track of the R-tree nodes that should be visited in the two channels. R-tree nodes are pushed into the

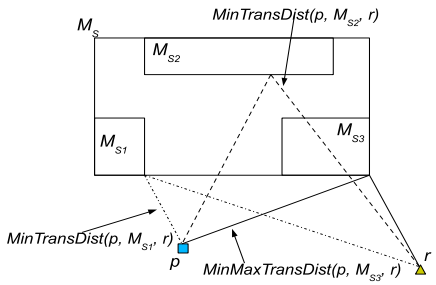


Figure 6: Updating and Pruning

queues and sorted in ascending order based on their arriving time in the two broadcast channels. Initially, Hybrid-NN algorithm pushes the two R-tree roots into the queues whenever they are available on air.

In Case 2, although the Hybrid-NN algorithm switches the target (the query point) of the NN search, the updating and pruning strategy are the same as traditional NN algorithms. The difference between traditional NN algorithms and Case 2 of Hybrid-NN algorithm lies in how to obtain the initial upper bound. When the NN search in Channel 1 finishes before the NN search in Channel 2, Hybrid-NN algorithm switches the query point of the second NN search to the point returned by the first NN search. If a temporary result is generated in channel 2, the upper bound is set to be the transitive distance from query point p to its nearest neighbor in the first channel, then to this temporary result. After that, for each node in the priority queue MBR_queue for Channel 2, Hybrid-NN computes the *MinDist* between the new query point and the children MBRs of this node. The smallest *MinDist* is used to update the upper bound if the upper bound is larger than this *MinDist*.

The upper bound updating strategy for Case 3 uses *transitive distance*, *MinMaxTransDist* and Lemma 3. Initially, it pushes the root of R-tree for dataset S into MBR_queue. When it is detected for the first time that Hybrid-NN algorithm is in Case 3, an initial upper bound update is performed. If there is a temporary point s' given as a potential NN to query point p in dataset S, Hybrid-NN algorithm uses $dis(p, s') + dis(s', r)$ to update the old upper bound. Then, since all the MBRs that will be visited are contained in a queue *MBR_queue*, for all MBRs M_i in *MBR_queue*, Hybrid-NN algorithm computes *MinMaxTransDist* and finds the minimum one:

$$z = \min_{M_i \in MBR\_queue} \{MinMaxTransDist(p, M_i, r)\}$$

If z is less than the current upper bound, then update the upper bound.

After the initial upper bound update, Hybrid-NN algorithm proceeds with query processing. When an MBR M_S of an intermediate node N_{inter} of the R-tree for dataset S is being visited, for all the children MBRs enclosed by M_S $\{M_{Si} : 1 \leq i \leq |N_{inter}|\}$ ($|N_{inter}|$ denotes the number of children of node N_{inter}), Hybrid-NN algorithm computes the *MinMaxTransDist* and finds the minimum one:

$$z = \min_{1 \leq i \leq |N_{inter}|} \{MinMaxTransDist(p, M_{Si}, r)\}$$

The upper bound is updated if and only if z is smaller than the current upper bound.

When an MBR M_S of a leaf node N_{leaf} is being visited, for all points $\{O_i : 1 \leq i \leq |N_{leaf}|\}$ ($|N_{leaf}|$ denotes the number of points in it) it encloses, Hybrid-NN algorithm computes the transitive distances and finds the minimum one:

$$z = \min_{1 \leq i \leq |N_{leaf}|} \{dis(p, O_i) + dis(O_i, r)\}$$

and update upper bound if and only if z is smaller than the current upper bound.

Pruning strategy uses *MinTransDist*. If an MBR M_S has $MinTransDist(p, M_S, r) > upperbound$, then this MBR is discarded and will not be downloaded from the broadcast channel. Figure 6 gives an example of updating and pruning. MBR M_{S3} has a *MinMaxTransDist* smaller than the old upper bound, then upper bound is updated with its value. MBR M_{S1} and M_{S2} has *MinTransDist* larger than the new upper bound. It is guaranteed that they do not contain the final answer, therefore they are pruned.

Algorithm 2 Hybrid-NN-Search Algorithm (Case 3)

Procedure:

```
1: initial_upperbound_update();
2: while MBR_queue is not empty do
3:    M ← MBR_queue.pop();
4:    if MinTransDist(p, M, r) > upperbound then
5:       continue;
6:    else
7:       wait until M is on air;
8:       if M is an intermediate node then
9:          for each child MBR M_i of M do
10:            z = min{MinMaxTransDist(p, M_i, r)};
11:            if upperbound > z then
12:               upperbound ← z
13:            end if;
14:         end for
15:         prune children M_i using MinTransDist
16:         sort Mi in ascending order based on arrival time;
17:         MBR_queue.push(M_i);
18:      else
19:         for each point s_i in M do
20:            z = min{dis(p, s_i) + dis(s_i, r)}
21:            if upperbound > z then
22:               upperbound ← z;
23:               s ← s_i
24:            end if
25:         end for
26:      end if
27:   end if
28: end while
29: return (s, r);
```

Algorithm 2 shows the updating and pruning of Hybrid-NN algorithm when Case 3 occurs, given r as the nearest neighbor of query point q generated in Channel 2.

4.2.4 Adjustments in Implementation

When Hybrid-NN algorithm starts with Case 1, it processes two NN queries in parallel. When an intermediate R-tree node is encountered, traditional NN algorithms first prune the children of this node and then push the remain-

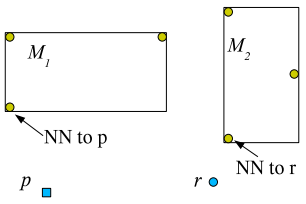


Figure 7: Updating and Pruning

ing children into a priority queue (MBR_queue). One problem with this approach is that when Hybrid-NN algorithm changes the query point or distance metrics to form a new query, the MBR which contains the answer to that new query may have been pruned. Figure 7 gives an example. MBR M_2 is pruned during the NN query processing which takes p as the query point. However, M_2 may contain the NN to the new query point r. In order to remedy this problem, the algorithm delays the pruning process of the NN search. When Hybrid-NN algorithm visits a non-leaf R-tree node, it pushes all the children nodes into MBR_queue. When an node is popped out of the queue, either *MinDist* (for Case 1 & 2) or *MinTransDist*(for Case 3) is computed and compared with the upper bound. The node is pruned if its distance metric is larger than the upper bound; otherwise, it is visited.

Note that this adjustment does not affect the correctness of the answer if no query point switching or distance metrics changing is made. One concern of this adjustment is that since Hybrid-NN algorithm is pushing more MBRs into the queue than typical NN algorithms, the size of the queue will become larger. However, the increase in queue size is not significant. Let M be the maximum fanout value, and H be the height of the R-tree, the worst-case queue size is $(H-1) \times (M-1)$. In our experiment, the R-tree for the dataset containing nearly 100,000 points has $H = 10$ and $M = 3$. Therefore, the mobile device only has to allocate memory space for $9 \times 2 = 18$ MBRs in its queue. This memory consumption is affordable.

5. OPTIMIZATION OF TNN ALGORITHMS

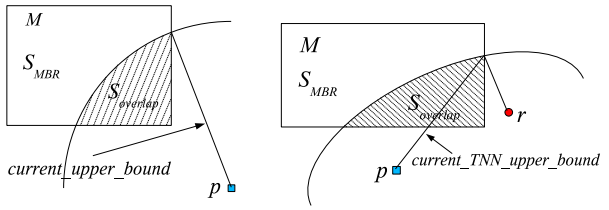
While our algorithms use *exact* search to provide a precise search range in order to reduce tune-in time in the filter phase, here we explore an optimization technique by using approximate NN search on R-tree to reduce tune-in time in the estimate phase (while not introducing much increase in tune-in time in the filter phase). The goal is to reduce the overall tune-in time in two phases.

In an exact nearest neighbor (eNN) search, traditional algorithms prune an R-tree node only when it is guaranteed that this node does not cover the answer of an NN query. When the MBR of an R-tree node is obtained, the minimum distance between this MBR and the query point is computed and compared with the current upper bound. If the former is larger, then it is guaranteed that this node does not contain the answer and thus can be pruned. Otherwise this node has to be visited and pushed into the queue. Approximate nearest neighbor search, on the other hand, aims to find a point that is not too far away from the query point. It relaxes the pruning condition so that more R-tree nodes can be pruned, compared with eNN. An R-tree node satisfying the ANN pruning condition (as will be discussed in the next subsection) is pruned, even if it is possible to contain the exact nearest neighbor to the query point.

When an ANN is used in the estimate phase of TNN query processing, the tune-in time in this phase is reduced since it visits less R-tree nodes compare to eNN. Since ANN does not guarantee to find the best/exact NN to the query point, the point ANN finds is farther to the query point than the exact NN. Thus, the search range determined may be larger than using the eNN and the tune-in time in the filtering Phase is increased. However, the ANN approach represents a new tradeoff between two phases from the previous approach. Our goal is to strive a good balance in this tradeoff in order to reduce the overall tune-in time. Also note that ANN optimization technique does not affect the final answer to the TNN query. Even though an ANN search may produce a larger search range, our algorithm assure that the TNN answer is included in this range (see Theorem 1).

5.1 ANN Pruning Condition

ANN estimates the probability that an R-tree node contains the real nearest neighbor to the query point p. During the R-tree traversal of the ANN search, the upper bound is obtained and updated in the same way as in the exact NN search: upper bound is set to be infinity at the beginning of ANN query processing and updated using the value of either the smallest $MinDist$ between p and an MBR or the smallest distance between p and any real point objects. ANN uses a new heuristic to estimate the probability that a node contains a point q such that $dis(q,p) \leq$ *current_upper_bound*, where *current_upper_bound* is the latest upper bound of ANN obtained during R-tree traversal. If this probability is smaller than a threshold α, it is pruned; otherwise it is preserved in the queue and waits to be visited.



(a) Circle-Rectangle (b) Ellipse-Rectangle

Figure 8: Overlap Area

The distribution of the children or real objects of an MBR of an R-tree node cannot be obtained unless this node is visited. Since the pruning decision has to be made before visiting a node, our heuristic assumes that the children or real objects are uniformly distributed in the MBR of this node and uses the following method to estimate the probability and do the pruning. An example is given in Figure 8(a).

HEURISTIC 1 (CIRCLE-RECTANGLE OVERLAP). *Given a query point p, an MBR M of an R-tree node, the upper bound upper_bound of an ANN query, let $S_{overlap}$ denotes the overlap area between M and a circle which takes p as the center and upper_bound as the radius; S_{MBR} denotes the area of M. The R-tree node is pruned if $S_{overlap}/S_{MBR} \leq \alpha$, where α is a pre-defined pruning threshold.*

Similarly, approximate search can also be applied to Case

3 of Hybrid-NN algorithm. ANN estimates the probability that an MBR contains a point that gives a transitive distance to p and r smaller than the current upper bound. For Case 3 of Hybrid-NN algorithm, we use the following heuristic to do the pruning and Figure 8(b) gives a corresponding example.

HEURISTIC 2 (ELLIPSE-RECTANGLE OVERLAP). *Given a starting point p, an ending point r, an MBR M of an R-tree node, the upper bound upper_bound of an TNN query in Case 3 of Hybrid-NN algorithm, let $S_{overlap}$ denotes the overlap area between M and an ellipse which takes p and r as two foci and upper_bound as the length of the major axis; S_{MBR} denotes the area of M. The R-tree node is pruned if $S_{overlap}/S_{MBR} \leq \alpha$, where α is a pre-defined pruning threshold.*

Note that the MBR which gives the latest upper bound has to be preserved and visited even if its ratio between overlap area and area of MBR is smaller than α. Otherwise the ANN algorithm will have a probability that it does not reach any leaf nodes of R-tree and no actual points will be retrieved. Also note that the value of threshold α varies between 0 and 1. When α is 0, ANN becomes eNN. A smaller α introduces limited approximation and may not be helpful to reduce the tune-in time. As α gets closer to 1, more R-tree nodes satisfy the pruning condition and ANN prunes out more R-tree nodes and the result of ANN becomes further to the query point p; thus results in a large search range and the penalty of tune-in time increase in the filter phase will be dramatic. The value of α should be chosen carefully. Different factors affecting the decision in choosing an α is discussed in the following subsection.

5.2 Factors Affecting Approximation Quality

The depth of an R-tree node and the height of the R-tree have an impact on the approximation quality. R-tree nodes with a smaller depth(near the root) covers a larger area and contains more data points than nodes with a large depth(near the leaves). Therefore if a node near the root is pruned, the penalty for finding a point close to the query point may be large. If a node near the leaves is pruned, the penalty is small and affordable. A fixed value for α may not be suitable for all R-tree nodes(as shown in our experiment). In our ANN algorithm, the value of α is determined dynamically during the traversal of the R-tree. When visiting a node near the root, ANN sets α close to 0 so that the search is close to eNN; while visiting a node close to leaves, ANN sets α close to 1 so that it prunes out a large number of nodes to reduce the tune-in time in the estimate phase and not introducing a large increase in the search range. We use the following equation to determine α:

$$\alpha = \frac{Node\_depth}{Rtree\_height} \times factor \tag{4}$$

where $Node\_depth$ is the number of level a node resides from the root and $Rtree\_height$ is the total number of levels an R-tree has. $factor$ is used to adjust the value of α to get the optimal optimization quality. It is determined by whether Double-NN, Window-Based-TNN or Hybrid-NN is used. As shown later in our experiments, for Double-NN and Window-Based-TNN algorithms, $factor = 1$, while for Hybrid-NN algorithm, $factor = 1/150$ or $1/200$.

Different densities of the two datasets involved in TNN query also have an impact on the value of α. E.g. the density of dataset S is larger than that of dataset R. Since S and R are covering the same region, points in S are closer to each other, while points in R are sparse. Therefore, given the same α, ANN on R contributes more to the increase in search range than the ANN on S. More points in S will be enclosed in this search range than those in R. These points in S will have to be retrieved in the range query in the filter phase. The tune-in time penalty in the range query on S will counteract the reduction in tune-in time in query processing on dataset R(as shown in our experiment). Taking into consideration the difference in densities, we set the value of α to be close or equal to 0 for the dataset with a smaller density of the two to reduce the tune-in time penalty in the dataset with a larger density.

6. PERFORMANCE EVALUATION

In this section, we present the result of experiments evaluating the performance of the Double-NN, Hybrid-NN algorithms and the two algorithms adapted from [19]. Two metrics are used in these experiments - *access time* and *tune-in time*, both measured by the number of pages accessed. The access time is the larger of the access times in both channels and the tune-in time is the sum of two tune-in times in both channels. The datasets used in these experiments include synthetic and real datasets. The points in one set of the synthetic datasets are uniformly distributed in a $39,000 \times 39,000$ square region. Eight datasets are generated for the first dataset involved in TNN query with densities $10^{-7.0}$, $10^{-6.6}$, $10^{-6.2}$, $10^{-5.8}$, $10^{-5.4}$, $10^{-5.0}$, $10^{-4.6}$ and $10^{-4.2}$, each having 152, 382, 960, 2,411, 6,055, 15,210, 38,206 and 95,969 points. Another set of eight uniform datasets are generated as the second dataset, with the same density range and area, but different points from the first set. For the second set of synthetic datasets, 16 datasets having sizes ranging from 2,000 to 30,000 with 2,000 increment. For simplicity, we use *UNIF(E)* to denote the first synthetic datasets, where E represents a power of ten; and we use the number of points to denote the second synthetic datasets. The real datasets include *CITY* and *POST* datasets, both extracted from [1]. The former contains nearly 6,000 cities and villages of Greece in a $39,000 \times 39,000$ square region, while the latter contains more than 100,000 post offices in the northeast of the United States in a $1,000,000 \times 1,000,000$ square region. When datasets with different areas are used, they are scaled to the same area.

In the experiments, R-tree is used as the air index. We arrange the R-tree in a depth-first order in the broadcast channels. The reason is stated in section 2.1. Due to the limited memory size of the mobile client, such a queue may not be possible to maintain. As discussed in Section 2.2, previous R-tree algorithms introduce backtracking in NN query, which deteriorates the performance severely in terms of access time in the broadcast environment due to its linear delivery property. Therefore, we maintain the priority queue of the candidate R-tree nodes according to their arrival time, so that backtracking is avoided. Because the location of the points in all the datasets are known a priori, and no insertion and deletion are involved, we use STR packing algorithm to build the R-tree in order to achieve the best performance [12]. (*1, m*) interleaving technique [10] is applied to arrange the index and data on both channels.

For each set of experiments, 1,000 query points are generated randomly in the same region as the datasets. We

Parameter	Size
size of an index pointer	2 bytes
size of a coordinate	4 bytes
size of a data content	$1k$ bytes
page capacity	64 - 512 bytes

Table 2: Parameter Setting

assume that when the mobile client first tune-in to the two broadcast channels, the roots of the two R-tree indices may not be available immediately and the mobile client has to wait to different time periods to get the two roots. Therefore, two random numbers are generated to simulate the waiting time to get the two roots. Also we assume that the computational overheads, including the computing time for the search range determination of Approximate-TNN-Search algorithm, the join step of all the algorithms to find the answer and ANN pruning condition checking are small and thus can be neglected. Other parameters in our experiments are the same as in [19] and listed in Table 2.

6.1 Algorithms with Exact Search

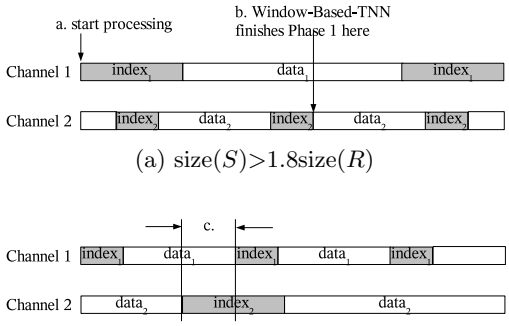
We evaluate the performance of our proposed algorithms and two existing ones with exact search. For the first set of synthetic datasets, we did experiments for each combination of two datasets with different densities ($8 \times 8 = 64$ sets of experiments). Different pruning strategies are tried out and different values for α are examined. Only part of the results are shown below as representatives due to the space limit. Please see [2] for full sets of experiments and evaluations. The page capacity is set to be 64 bytes.

6.1.1 Access Time

The access time of the four algorithms is only affected by the sizes of datasets and visiting orders of TNN queries. Figure 9 gives the results of access time of the four algorithms. Among these algorithms, the Approximate-TNN-Search always gives the best performance in terms of access time because it computationally estimates the search range. Therefore, it avoids the two index searches in the first phase and allows the algorithm to go directly into the second phase; whereas the other algorithms have to search on the indices to determine the search range. Note that the Double-NN and Hybrid-NN algorithms always have the same access time because they start receiving and end processing at exactly the same time. Double-NN and Hybrid-NN give a better access time than Window-Based-TNN-Search. Figure 9(a) and Figure 9(b) show the experiment results in which the size of either S or R is fixed to 10,000 points and the sizes of the other datasets varies from 2,000 to 30,000; while Figure 9(c) and Figure 9(d) show the variation of access times in a large range of densities of the two datasets.

Our experiments show that when $size(S) \geq size(R)/40$ and $size(S) \leq 1.8 \times size(R)$, Double-NN and Hybrid-NN give a better performance than Window-Based-TNN in terms of access time. The largest improvement in access time occurs when both datasets have similar sizes and Double-NN and Hybrid-NN reduce the access time by 7% to 15%. When the difference between the sizes of S and R becomes larger than the above range, access times of the three algorithms become similar.

When size(S)>1.8size(R), Window-Based-TNN has a high probability to finish the NN search on Channel 2 before the next root on Channel 1 arrives (on time point b in Figure 10(a)). Then it starts and ends the filter phase at the same time as Double-NN and Hybrid-NN. When size(S) < size(R)/40, Double-NN and Hybrid-NN give a shorter access time than Window-Based-TNN only when the starting time point of query processing falls in range c in Figure 13(b). As the difference between sizes of R and S increases, this probability reduces. Thus in the above two cases, all three algorithms give similar access time.



(a) size(S)>1.8size(R)

(b) size(S)<size(R)/40

Figure 10: Access Time Analysis

Also note that the access time is dominated by the processing time of the larger dataset of the two. This is because access time is the longer of the access times in the two channels and larger dataset usually result in a larger processing time.

6.1.2 Tune-in Time

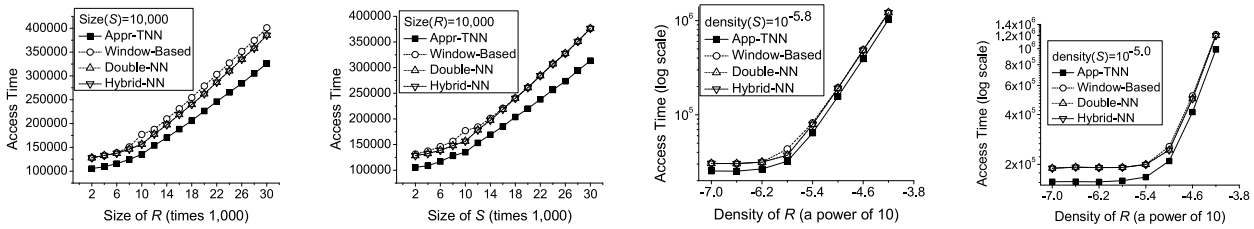
Figure 12 gives the tune-in time of Window-Based-TNN, Double-NN and Hybrid-NN algorithms. In each of the four sets of experiments, we fix the size of dataset S and changes the size of R.

When 0.01size(R) $\leq$ size(S) $\leq$ 0.4size(R), Hybrid-NN gives the best tune-in time among the three algorithms, as shown in Figure 11(a) and 11(b). In this case, Hybrid-NN algorithm finds a shorter search range than Double-NN and Window-Based-TNN, while taking similar tune-in time in the first phase to estimate the search range, and thus provides the smallest tune-in time.

When size(S)$\geq$ 0.4size(R), the decrease in tune-in time in filter phase of Hybrid-NN algorithm is countervailed by the increase in tune-in time in the estimate phase to find a short search range, thus rendering the overall tune-in time to be larger than the other two algorithms. Also, in this case, Double-NN and Window-Based-TNN algorithms have similar tune-in times, as shown in Figure 11(a) and 11(b).

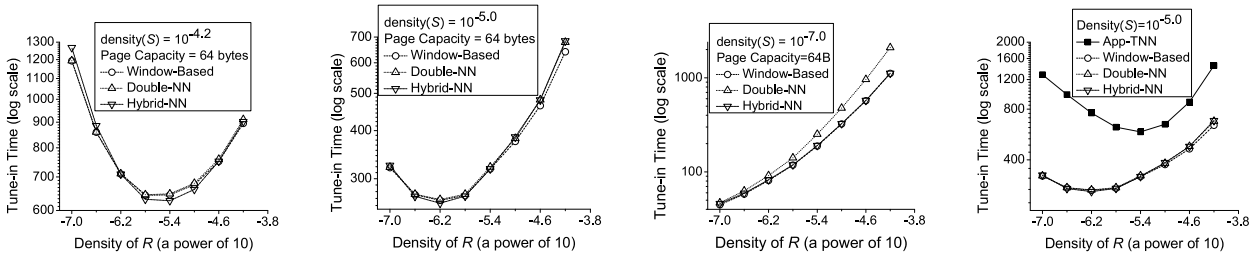
When size(S)< 0.01size(R), Window-Based-TNN gives the best tune-in time because the search range determined by Window-Based-TNN is smaller than those of the other two algorithms, while they uses similar tune-in times to determine the search range. Also note that as size(S) keeps increasing, the tune-in time of Hybrid-NN gets closer to that of Window-Based-TNN. This is because the NN search in Channel 1 has a high probability to finish before the NN search in Channel 2. Hybrid-NN, in this case, acts similarly to Window-Based-TNN algorithm.

For Approximate-TNN algorithm, although it never fails



(a) $S = 10,000$ (b) $R = 10,000$ (c) $S = UNIF(-5.8)$ (d) $S = UNIF(-5.0)$

Figure 9: Access Times



(a) $S = UNIF(-4.2)$ (b) $S = UNIF(-5.0)$ (c) $S = UNIF(-7.0)$ (d) $S = UNIF(-5.0)$

Figure 11: Tune-in Time VS. Density

to generate the correct answer for TNN queries in our experiments, the tune-in time is significantly larger than all the other algorithms. One example is shown in Figure 11(d). Approximate-TNN algorithm generates an unnecessarily large search range by the equation and thus results in a dramatic increase in the tune-in time in the filter phase. This is especially severe when one of the two datasets is very sparse. A large amount of points on the dense dataset have to be retrieved and the penalty in tune-in time is large.

6.2 Optimization

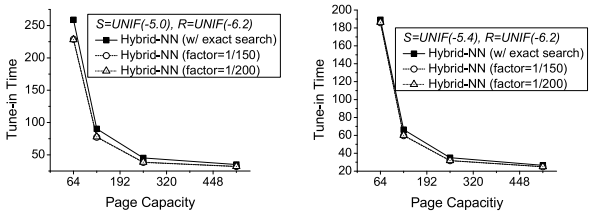
6.2.1 ANN vs. eNN

Our optimization technique with approximate-NN search reduces the tune-in time of all the three algorithms. Figure 12(a) gives the result of tune-in time improvement of Window-Based-TNN and Double-NN algorithms, with approximation factor $factor$ equals to 1. ANN reduces tune-in time in all the four page capacity settings. Due to the space limit, only page capacity of 64 bytes is given in Figure 12(a). Improvement in tune-in time ranges from 11% to 20%. The two datasets involved have the same size. The impact of difference between sizes are discussed in the following subsection, as well as the Hybrid-NN algorithm, which mainly works on datasets of different sizes.

6.2.2 Impact of Sizes of Datasets

If two datasets with different densities use the same strategy (using equation 4 with $factor = 1$) to determine the value of threshold α, the penalty on the denser dataset of the two will countervail the reduced tune-in time in the estimate phase and thus increases the total tune-in time. For datasets with different densities, we set α of the sparse dataset to be 0 (meaning to do exact NN search) and α of the denser dataset using equation 4 with $factor = 1$. Figure 12(b) and 12(c) give the performance of using this strategy to determine the values of α. The tune-in time of both Window-Based-TNN and Double-NN algorithms are reduced, no matter which visiting order is taken. Figure 12(d) shows that this strategy also reduces the tune-in time of TNN queries over real datasets. In this set of experiments, $S = CITY$ and $R = POST$, all the four page capacity settings are checked.

For Hybrid-NN algorithm, our experiments show that when $factor = 1/150$ or $1/200$, ANN optimization reduces the tune-in time. Figure 13 gives the results.

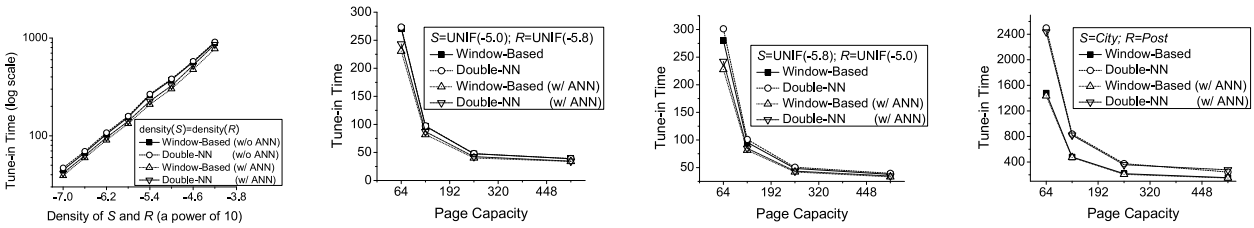


(a) S=UNIF(-5.0) (b) S=UNIF(-5.4)

Figure 13: Hybrid-NN with ANN

6.3 Distribution

The distribution has an important impact on the performance of the existing Approximate-TNN-Search algorithm. Approximate-TNN-Search estimates a correct search range only for uniformly distributed datasets. However, as skewed datasets are involved, the Approximate-NN-Search algorithm does not guarantee to provide an effective search range and therefore fails the TNN query. Table 3 gives the combination of distributions of datasets and the average fail rate. In *uni-*



(a) ANN vs. eNN (b) density(S)>density(R) (c) density(R)>density(S) (d) ANN on Real Dataset

Figure 12: Applying ANN Optimization Technique

Density Combination	Average Fail Rate
uni-uni	0%
uni-real	9.08%
real-uni	9.08%
real-real	43.2%

Table 3: Fail Rate

real and *real-uni* combinations, we use *CITY* dataset and change the eight uniform ones. For *real-real* combination, we use *CITY* and *POST*. We also uses three page capacities - 64 bytes, 128 bytes 256 bytes and 512 bytes, and average fail rates are calculated. Note that Double-NN and Hybrid-NN never fails to provide the correct answer to TNN queries.

7. CONCLUSION AND FUTURE WORK

In this paper, we visit a scenario of near future where a mobile device can access multiple channels simultaneously and study the query processing of Transitive Nearest Neighbor Search. We adapt two existing algorithms to the new environment, analyze their deficiencies and propose our new algorithms. A comprehensive simulation has been conducted and results show that our new algorithms effectively reduces the access time and cases in which our algorithms reduces tune-in time is stated and analyzed. We also propose a novel optimization technique - using ANN to substitute eNN to save tune-in time. Factors affecting the optimization quality are discussed. Experiments show that our optimization method effectively reduces tune-in time of all algorithms.

This work starts new research of query processing in a broadcast environment where clients may access to multiple channels simultaneously. We plan to study the processing of generalized TNN queries, including 1) more than 2 datasets are involved, and allocated on multiple wireless channels, 2) the visiting order of the types of objects of interest is not specified, and 3) a complete travel route, which includes the route to return to the source point after visiting all interested types of objects, is queried. Besides these variants of TNN, we will also study the query processing of other types of queries in this new environment. We will continue the effort in this direction and work out new solutions.

8. ACKNOWLEDGEMENT

Wang-Chien Lee was supported in part by the National Science Foundation under Grant no. IIS-0328881, IIS-0534343 and CNS-0626709.

9. REFERENCES

[1] Spatial datasets. http://dias.cti.gr/ ytheod/research/datasets/spatial.html.
[2] Technical report of tnn. http://www.cse.psu.edu/∼xiazhang/technicalreport.html.
[3] Utstarcom unveils t66 cdma/gsm dual-mode phone. http://www.slashphone.com/111/4690.html.
[4] K. Cheung and W.-C. Fu. Enhanced nearest neighbour search on the r-tree. *SIGMOD Record*, 1998.
[5] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, 1984.
[6] G. Hjaltason and H. Samet. Distance browsing in spatial databases. *TODS*, 1999.
[7] Q. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *Proceedings of ICDE*, pages 157–166, 2000.
[8] A. R. Hurson, Y. Chehadeh, and J. Hannan. Object organization on parallel broadcast channels in a global information sharing environment. In *IPCCC*, 2000.
[9] T. Imielinski, S. Viswanathan, and B. Badrinath. Power efficiency filtering of data on air. In *EDBT*, 1994.
[10] T. Imielinski, S. Viswanathan, and B. Badrinath. Data on air: organization and access. *TKDE*, 1997.
[11] I. Kamel and C. Faloutsos. On packing r-trees. In *Proceedings of CIKM*, pages 490–499, 1993.
[12] S. Leutenegger, M. Lopez, and J. Edgington. Str: a simple and efficient algorithm for r-tree packing. In *Proceedings of ICDE*, pages 497–506, 1997.
[13] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng. On trip planning queries in spatial databases. In *Proceedings of SSTD*, 2005.
[14] K.-I. Lin, M. Nolen, and K. Kommeneni. Utilizing indexes for qpproximate and on-line nearest neighbor queries. In *Proceedings of IDEAS*, 2005.
[15] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of SIGMOD*, 1995.
[16] N. roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed r-trees. In *Proceedings of SIGMOD*, pages 17–31, 1985.
[17] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *VLDB Journal*, 2007.
[18] B. Sun, A. R. Hurson, and J. Hannan. Energy-efficient schedualing algorithms of object retrieval on indexed parallel broadcast channels. In *ICPP*, 2004.
[19] B. Zheng, K. C. Lee, and W.-C. Lee. Transitive nearest neighbor search in mobile environments. In *Proceedings of SUTC*, pages 14–21, 2006.