Location-based Spatial Queries with Data Sharing in Mobile Environments

Wei-Shinn Ku, Roger Zimmermann, and Chi-Ngai Wan

Computer Science Department
University of Southern California
Los Angeles, California 90089
[wku, rzimmerm, cwan]@usc.edu

July 6, 2005

Abstract

Mobile clients feature increasingly sophisticated wireless networking support that enables real-time information exchange with remote databases. Location-dependent queries, such as determining the proximity of stationary objects (e.g., restaurants and gas stations) are an important class of inquiries. We present a novel approach to support nearest-neighbor queries from mobile hosts by leveraging the sharing capabilities of wireless ad-hoc networks. We illustrate how previous query results cached in the local storage of neighboring mobile peers can be leveraged to either fully or partially compute and verify spatial queries at a local host. The feasibility and appeal of our technique is illustrated through extensive simulation results that indicate a considerable reduction of the query load on the remote database. Furthermore, the scalability of our approach is excellent because a higher density of mobile hosts increases its effectiveness.

1 Introduction

Location-based queries are of interest in a number of applications, for example, geographical information systems. An example query might be "find the nearest gas station" or "find the three nearest Italian restaurants." Increasingly such queries are issued from mobile clients.

In this study we propose an approach that leverages short-range, ad-hoc networks to share information in a peer-to-peer (P2P) manner among mobile clients to answer location-based nearest neighbor queries. The efficiency of our approach is derived from the observation that the results of spatial queries often exhibit spatial locality. For example, if two mobile hosts are close to each other, the result sets of their kNN queries for a specific object type may overlap significantly. Through *mobile cooperative caching* [3] of the result sets, query results can be efficiently shared among mobile clients.

Figure 1 shows an example. At time T the mobile query point Q can establish contact with two other mobile hosts within its communication range: P_1' and P_2'. Both of these clients in the past executed a 1NN query for the nearest gas station when they were located at P_1 and P_2, respectively[1]. The results that they obtained and cached were $<n2, P_1>$ and $<n4, P_2>$. These two tuples represent candidate solutions for Q's own 1NN query. Through a local verification process Q can determine whether one of the solutions obtained from its neighbors is indeed its own nearest gas station. Note that the current

[1] In our notation we use the object identifier to represent its position coordinates.

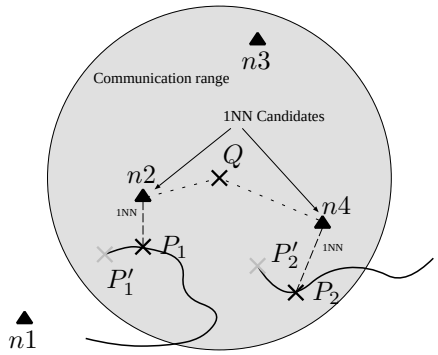

Figure 1: Example of 1NN peer-to-peer result sharing.

location of the neighboring hosts, P_1' and P_2', has no specific significance, as long as they are within the communication range of Q.

The contributions of this study are as follows. In our methodology we first identify a set of characteristics that enable the development of effective sharing methods. We then introduce a set of algorithms that aid in the decision process within this distributed environment to verify whether the data items received from neighboring clients provide a complete, partial, or irrelevant answer to the posed query. Our initial method verifies results from a single neighbor, and we then extend it to work with multiple neighboring clients. Finally, through extensive simulation experiments we explore the benefits of our approach under different parameter sets (e.g., mobile host density, wireless transmission range).

The rest of this paper is organized as follows. Section 2 introduces the related work for processing NN and kNN queries. Our approach is detailed in Section 3 and the experimental results obtained from our simulation model are contained in Section 4. Finally, Section 5 concludes the paper and outlines future research directions.

2 Related Work

The existing work relevant to our approach can broadly be classified into two areas, namely *nearest neighbor query processing* and *cache management in mobile environments*.

Nearest Neighbor Query Processing One of the most fundamental operations performed on spatial data sets is to find the nearest neighbors of a query object – either one (1NN) or a specific number k (kNN).

R-trees [9] and their derivatives have been a prevalent method to index spatial data and increase query performance. To find nearest neighbors, Roussopoulos et al. [15] proposed a branch-and-bound algorithm that searches an R-tree in a depth-first manner. The best-first NN algorithm proposed by Gísli et al. [10] retains a heap with the entries of the nodes visited so far and it always expands the first entry in the heap. The algorithm is optimal since it visits only the minimally necessary nodes and it reports nearest neighbors in ascending order according to their distance to the query point. It can be applied without a-priori knowledge of the number k of queried nearest neighbors. Both the depth-first and best-first algorithms are designed for stationary objects and query points. They may be used when moving objects infrequently issue nearest neighbor queries (single-step search).

With the emergence of mobile devices attention has focused on the problem of continuously finding k nearest neighbors for moving query points (k-NNMP). A naive approach might be to continuously issue kNN queries along the route of a moving object (multi-step search). This solution results in repeated server accesses and nearest neighbor computations and is therefore inefficient. One method to reduce

the computational complexity is to sample the trajectory instead of treating it as a continuous curve. Song and Roussopoulos [18] utilize partial results from queries launched at previous sampled positions to reduce the accesses to the server.

Tao et al. [19] investigated the problem of finding the continuous nearest neighbor of every point along a segment, assuming that the trajectory is known and can be described in closed form with a function of reasonable complexity. Their approach pre-computes so-called *split-points* that divide the object path into segments during which the nearest neighbor result sets remain unchanged. This method is efficient, but requires knowledge of the object path (or at least an approximation).

In the early work, nearest neighbor searches were based on the Euclidean distance between the query object and the sites of interest. However, in many applications objects cannot move freely in space but are constrained by a network (e.g., cars on roads, trains on tracks). Therefore, in a realistic environment the nearest neighbor computation must be based on the spatial network distance, which is more expensive to compute. A number of techniques have been proposed to manage the complexity of this problem [12, 13, 17].

Cache Management in Mobile Environments Caching is a key technique to improve data retrieval performance in widely distributed environments. Leveraging the combined resources of several cooperating caches has been proposed to improve file system [5] and Web performance [20]. In conventional mobile environments, wireless connections are treated as extensions of the wired infrastructure. Hence, mobile clients retrieve information from database servers via intermediate base-stations. With the increasing deployment of new peer-to-peer wireless communication technologies (e.g., IEEE 802.11x and Bluetooth) there exists a new information sharing alternative known as *peer-to-peer cooperative caching*. With this technique mobile hosts communicate with neighboring peers in an *ad-hoc* manner to share information rather than having to rely on the communication link to the remote information sources.

Peer-to-peer cooperative caching can bring about several distinctive benefits to a mobile system: improving access latency, reducing server workload and alleviating point-to-point channel congestion. As a disadvantage, it may increase the communication overheads among mobile hosts.

Recently, a number of techniques have been proposed to address caching in ad-hoc peer-to-peer networks. The COoperative CAching (COCA) [3] scheme investigates the effects of client activity levels, data replication, and cache size. The benefits of clustering mobile clients into groups are investigated in [4].

Semantic caching stores additional information such as query descriptions with the data items in the local cache. The metadata is used to determine whether a new query is fully answerable from the cache. In that case no communication with the server is required. If the query can only partially be answered, then it may be trimmed and sent to the server. In either case, the database processing complexity and the amount of data transferred can be significantly reduced [14]. One form of cachable metadata is the data access path which may be used to redirect future accesses to nearby cache nodes [21]. Another possibility is to cache the index that supports the objects. The cached index enables the objects to be reused for common types of queries [11].

Addressing specifically the nearest neighbor search for stationary objects, Zheng et al. [22] proposed an index based on Voronoi diagrams in conjunction with a semantic cache to enhance the efficiency of the search.

3 Sharing Based Nearest Neighbor Queries

The fundamental idea of our methodology is to leverage the cached results from previous queries at reachable mobile hosts for answering spatial queries at the local host. In some circumstances the query

results obtained from peer hosts can only provide a partial answer or no answer at all. To achieve scalability it is imperative that a mobile client can locally determine whether the result set from its neighbors provides a full, partial or no answer. As a novel component in our methodology we present a verification algorithm that can certify if a result object is part of the solution set. We term such an object *certain*. If the object is not guaranteed to be part of the result set, we call it *uncertain*. The first variant of our verification procedure validates object certainty from a single peer. We then extend the process to multiple peers. If no full set of certain objects can be retrieved from neighboring peers, the query is forwarded to a spatial database server including the acquired partial result. The database search efficiency can be improved by utilizing the partial query results from clients.

In this study we use the k-nearest neighbor (kNN) search as an example of spatial queries and explain in detail how they are processed by cooperating mobile hosts. Specifically, in Section 3.1 we introduce the infrastructure that we assume for our work. Next, Section 3.2 presents algorithms for verifying query results from neighboring peers and the exact ranking of verified Points Of Interest (POI). Section 3.3 explains how to use our metrics to decrease the server load for processing kNN queries. Section 3.4 illustrates the extension of our algorithms for solving spatial network kNN queries.

3.1 Assumed Infrastructure

Figure 2 depicts our operating environment with two main entities: remote spatial databases and wireless mobile hosts. We are considering mobile clients, such as cars, that are instrumented with a global positioning system (GPS) for continuous position information. Furthermore, we assume that two-tiers of wireless connections are available on future automobiles. Traditional, cellular-based networks (such as utilized by the OnStar service) allow medium range connections to base-stations that interface with the wired Internet infrastructure. A second type of short-range networks allow ad-hoc connections with neighboring mobile clients. Technologies that enable short range communication include, for example, IEEE 802.11x. Benefiting from the power capacities of vehicles, we assume that each mobile host has a significant transmission range and virtually unlimited lifetime. The architecture can also support hand-held mobile devices. However, then power consumption becomes an additional parameter which we are not currently considering.

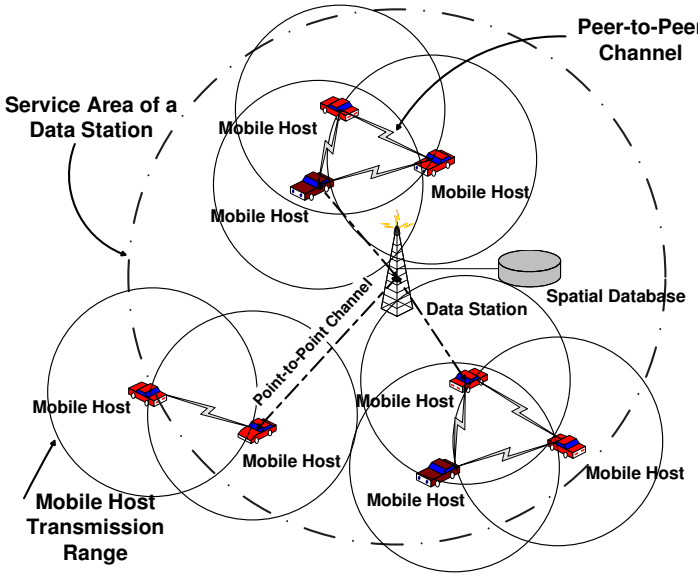

Figure 2: Example system environment.

3.2 Euclidean Distance Nearest Neighbor Queries

With our assumed infrastructure, a mobile host Q has two choices to find the solution to a kNN query. First, it can send the query directly to the database server for execution. However, since spatial searches are expensive to compute, the server may become a bottleneck when the number of mobile hosts grows. An alternative solution is to collect NN data from peers and harvest these existing query results for completing Q's query. We term the latter approach a *Sharing-based Euclidean distance Nearest Neighbor* (SENN) query. We will subsequently extend the technique to make use of the network distance.

We propose two approaches to process NN information obtained from peers for fulfilling NN queries of a mobile host Q. The single peer NN verification, also called $k\text{NN}_{single}$, attempts to verify k certain objects by sequentially verifying the returned NN data from each peer. If this is not possible, then the multiple peer NN verification process, $k\text{NN}_{multiple}$, attempts to complete the verification process with several peers.

3.2.1 Single Peer NN Verification: $k\text{NN}_{single}$

The objective of the $k\text{NN}_{single}$ method is to verify if the point of interest data returned from each peer can be valid nearest neighbors of a mobile host Q. In order to verify if a returned POI object n_i is one of the top k nearest neighbors of a mobile host Q, we utilize the spatial relationship between mobile hosts and their POIs as follows.

Lemma 3.1 *Let Q and P_1 be two mobile hosts, and let P_1 have k nearest neighbors, $n_1, n_2, \ldots, n_k$ which are sorted in ascending order according to their distance to P_1. If Dist$(Q, n_i) + \delta >$ Dist(P_1, n_k) then n_i cannot be verified as one of the top k-nearest neighbors of Q.*

In the above lemma, Dist(Q, n_i) is the Euclidean distance between Q and n_i, δ is the Euclidean distance between Q and P_1, and Dist(P_1, n_k) is the Euclidean distance between P_1 and its cached farthest nearest neighbor n_k. An illustration of Lemma 3.1 is shown in Figure 3. The nearest neighbor n_2 of mobile host P_1 – which is a peer of mobile host Q – cannot be verified as one of the top k-nearest neighbors of Q. An uncertain area exists in the circle which takes Q as its center point and Dist(Q, n_2) as its radius. A point of interest n_i may be located in that area with Dist(Q, n_i)<Dist(Q, n_2). Therefore, we can only classify n_2 as an *uncertain nearest neighbor*.

Lemma 3.2 *Let Q and P_1 be two mobile hosts, and let P_1 have k nearest neighbors, $n_1, n_2, \ldots, n_k$ which are sorted in ascending order according to their distance to P_1. For any nearest neighbor n_i of P_1, if Dist(Q, n_i) + $\delta \leq$ Dist(P_1, n_k) then n_i is one of the top k-nearest neighbors of Q.*

Proof Assume $n_i \notin k$NN of Q. Then, there exist $m_1, m_2, \ldots, m_k \in k$NN of Q such that $\forall m_j \in \{m_1, m_2, \ldots, m_k\}$, $\text{Dist}(Q, m_k) < \text{Dist}(Q, n_i)$. We can identify a point M located at the intersection of the extension of the line from P_1 to Q and the circumference of the circle with center Q and radius $\text{Dist}(Q, n_i)$ (identified as circle C_Q in Figure 4). Then, $\forall m_i \in k$NN of Q:

$$\text{Dist}(P_1, m_i) < \text{Dist}(P_1, M) \tag{1}$$

Recall that we assume the following inequality holds:

$$\text{Dist}(Q, n_i) + \delta \leq \text{Dist}(P_1, n_k) \tag{2}$$

Because the circle C_Q (with center Q and radius $\text{Dist}(Q, M)$) is fully covered by the circle C_{P1} (with center P_1 and radius $\text{Dist}(P_1, M)$) it follows that

$$\text{Dist}(Q, n_i) + \delta = \text{Dist}(Q, M) + \delta = \text{Dist}(P_1, M) \tag{3}$$

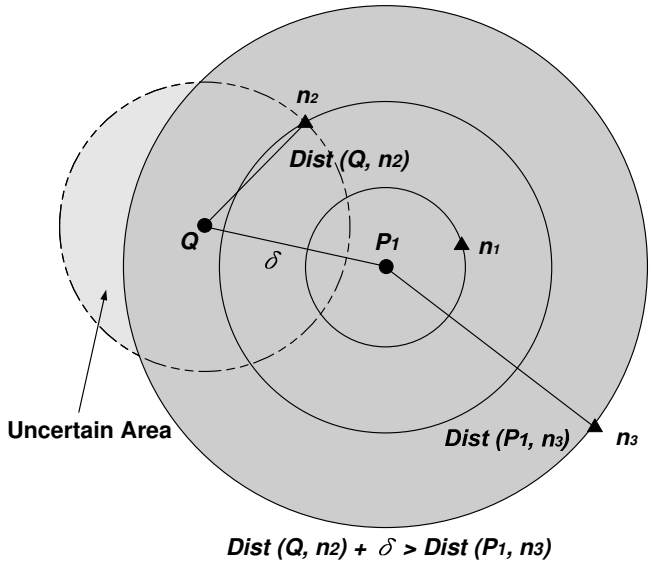

Figure 3: Verification of a point of interest with an uncertain area.

By Equations 1, 2 and 3, $\forall m_i \in k$NN of Q, $\text{Dist}(P_1, m_i) < \text{Dist}(P_1, n_k)$. Thus $n_k \notin k$NN of P_1. However, this contradicts the assumption that $n_k \in k$NN of P_1. Therefore, n_i must be one of the top k-nearest neighbors of Q. □

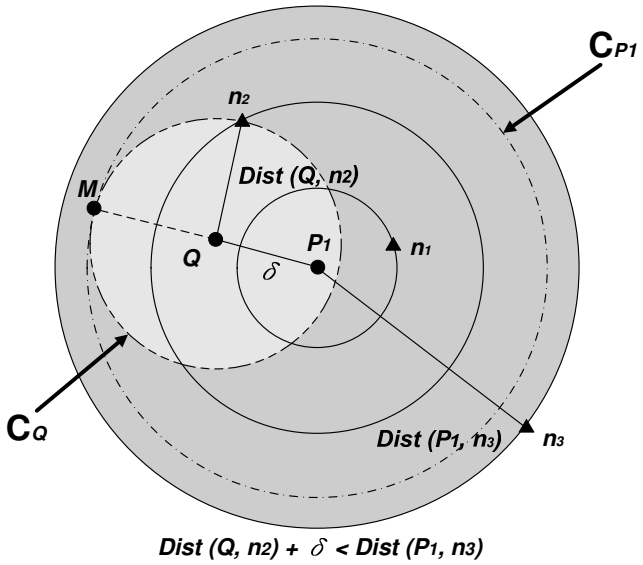

Figure 4: Verification of a certain point of interest.

An illustration of Lemma 3.2 is shown in Figure 4. The nearest neighbor n_2 of mobile host P_1, which is a peer of mobile host Q, can be verified as the nearest neighbors of Q and is termed a *certain nearest neighbor*. Because the Euclidean distance between n_2 and Q plus the Euclidean distance between Q and P_1, δ, is no greater than the Euclidean distance between P_1 and its presently cached farthest nearest neighbor, n_3.

We observe quite intuitively that the cached query locations which are located closer to the query host Q have a higher likelihood (but are not guaranteed) to be able to provide useful (i.e., adjacent)

points of interest. We therefore will use Heuristic 3.3 to guide the order in which the cached NN query results returned from neighboring peers are processed.

Heuristic 3.3 *Let NN_{set} be a result set of nearest neighbor objects and their query points returned from the peers of the query mobile host Q. Sorting NN_{set} in ascending order according to the distance between each query point to the location of Q may save computation time during processing by a subsequent search algorithm.*

Lemma 3.4 $\forall$ *x, y, and Q, if Dist(x, Q) < Dist(y, Q) and if $y \in$ kNN of Q, then $x \in$ kNN of Q and Rank(x, Q) < Rank(y, Q).*

Since $\text{Dist}(x, Q) < \text{Dist}(y, Q)$, we know that x is closer to Q than y. According to the definition of nearest neighbor and $y \in k$NN of Q, we conclude that $x \in k$NN of Q and $\text{Rank}(x, Q) < \text{Rank}(y, Q)$.

Lemma 3.5 $\forall$ *x, y, and Q, if Dist$(x, Q) \neq$ Dist(y, Q), then $x \neq y$.*

This follows from the definition of Euclidean distance – since we have $\text{Dist}(x, Q) \neq \text{Dist}(y, Q)$, $x \neq y$.

Lemma 3.6 $\forall$ *P and Q, Dist$(P, Q) = \delta$, $\forall n_i, n_j \in$ kNN of P, if Dist(P, n_i) < Dist(P, n_j) and Dist(Q, n_i) + $\delta \leq$ Dist(P, n_j), then Rank$(n_i, Q) < k$ and $n_i \in$ kNN of Q.*

Proof According to Lemma 3.2 and the assumed conditions, n_i must be one of the top k nearest neighbors of Q. Since we know that $n_j \in k$NN of P, there are at most $k-1$ POIs within circle C_P (with center P and radius $\text{Dist}(P, n_j)$). Therefore, we can verify at most $k-1$ POIs for Q by utilizing n_j and it follows that $\text{Rank}(n_i, Q) < k$. □

Lemma 3.7 $\forall$ *P and Q, Dist$(P, Q) = \delta$. Given $n_1, n_2, \ldots, n_k \in$ kNN of P and n_i sorted in ascending order to their distance to Q: Dist(Q, n_1) < Dist(Q, n_2) < ... < Dist(Q, n_k). If Dist$(Q, n_i) + \delta \leq$ Dist(P, n_j) and Dist(P, n_i) < Dist(P, n_j), then n_i is the i^{th} nearest neighbor of Q and Rank$(n_i, Q) = i$.*

Proof Since $\text{Dist}(Q, n_i) + \delta \leq \text{Dist}(P, n_j)$ and $\text{Dist}(P, n_i) < \text{Dist}(P, n_j)$, by Lemma 3.6 we observe:

$$\text{Rank}(n_i, Q) < k \tag{4}$$

Because we are given that $\text{Dist}(Q, n_1) < \text{Dist}(Q, n_2) < \ldots < \text{Dist}(Q, n_k)$, we deduce that there exist $i-1$ points $(n_1, n_2, \ldots, n_{i-1})$ which are closer to Q than n_i. Therefore, we derive:

$$\text{Rank}(n_i, Q) \geq i \tag{5}$$

From Equations (4) and (5), we conclude:

$$i \leq \text{Rank}(n_i, Q) < k \tag{6}$$

In the next step, we prove $\text{Rank}(n_i, Q) = i$ by contradiction. Suppose $\text{Rank}(n_i, Q) > i$, then there must exist a NN n_p, such that $n_p \notin \{n_1, n_2, \ldots, n_{i-1}\}$ and $\text{Dist}(n_p, Q) < \text{Dist}(n_i, Q)$. Therefore, $\forall n_x \in \{n_i, n_{i+1}, \ldots, n_k\}$, $\text{Dist}(n_p, Q) < \text{Dist}(n_x, Q)$. By Lemma 3.5, we observe $n_p \notin \{n_i, n_{i+1}, \ldots, n_k\}$. Consequently, $n_p \notin \{n_1, n_2, \ldots, n_k\}$ and $n_p \notin k$NN of P. As shown in Figure 5, n_p is inside the circle

with Q as its center point and $\text{Dist}(Q, n_i)$ as its radius. According to the characteristics of triangles we derive:

$$\text{Dist}(n_p, P) < \text{Dist}(n_p, Q) + \text{Dist}(Q, P) \tag{7}$$

and

$$\text{Dist}(n_p, Q) + \text{Dist}(Q, P) < \text{Dist}(n_i, Q) + \text{Dist}(Q, P) \tag{8}$$

Since we are given $\text{Dist}(n_i, Q) + \text{Dist}(Q, P) < \text{Dist}(P, n_j)$ and Equations (7) and (8), it follows that:

$$\text{Dist}(n_p, P) < \text{Dist}(P, n_j) \tag{9}$$

By Lemma 3.4 and Equation (9) we deduce that $n_p \in k\text{NN}$ of P, which contradicts what we concluded before, namely that $n_p \notin \{n_1, n_2, \ldots, n_k\}$ and $n_p \notin k\text{NN}$ of P. Therefore, the assumption $\text{Rank}(n_i, Q) > i$ cannot hold and it follows that $\text{Rank}(n_i, Q) \leq i$. Since we know that $\text{Rank}(n_i, Q) \geq i$ from Equation (5), we conclude that $\text{Rank}(n_i, Q) = i$. □

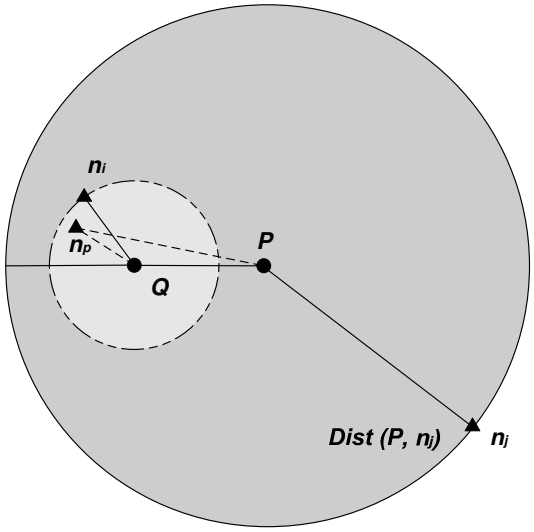

Figure 5: The spatial relationships between n_i, n_p, Q, and P.

The $k\text{NN}_{single}$ method maintains a heap H with the entries of certain and uncertain points of interest discovered so far (illustrated in Table 1). The size of H is determined by the total number of queried interest objects Q_k. Initially H is empty and the $k\text{NN}_{single}$ method processes the sorted NN_P in sequence. The heap H is updated according to the distance from the location of Q to a POI object and its certainty. If there exist uncertain nearest neighbor objects in H, a newly discovered certain NN object will replace an uncertain object and H maintains the certain objects in an ascending order of their Euclidean distance to Q. Uncertain objects exist in H only if the number of certain objects is less than Q_k. These uncertain objects are also stored in ascending distance order.

Consider the following example to illustrate the operation of $k\text{NN}_{single}$. Figure 6 illustrates the location of Q and its two closest mobile hosts, P_1 and P_2. Because the cached query location of mobile host P_1 is the closest to Q, $k\text{NN}_{single}$ starts its NN verification process from P_1. The single peer NN verification rule follows Lemma 3.2. Assuming that Q searches for four nearest neighbors, then after processing P_1 and P_2 the content of the heap H is as shown in Table 1. Based on the set NN_P of both peer P_1 and P_2, Q can retrieve two certain NNs, n_{2-P1} and n_{1-P1}, and two uncertain NNs, n_{3-P1} and n_{3-P2}.

Certain/Uncertain	C	C	UC	UC
Points of Interest	n_{2-P1}	n_{1-P1}	n_{3-P1}	n_{3-P2}
Distance to Q	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{5}$	$\sqrt{8}$

Table 1: The data structure of the heap H.

kNN$_{single}$ is executed iteratively with each peer in the nearest neighbor data set NN_P. If k elements in H are certain, the kNN query is fulfilled and H will remember the top k NN in sequence. Otherwise, we need to perform kNN$_{multiple}$ to expand the search space to include more candidate certain interest objects.

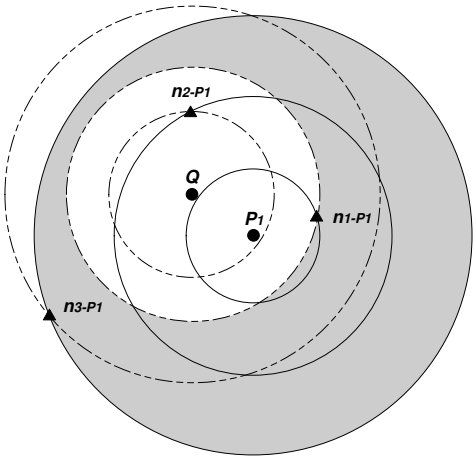

Fig. 6a. Mobile host Q retrieves two certain nearest neighbors from the NN set of peer P_1.

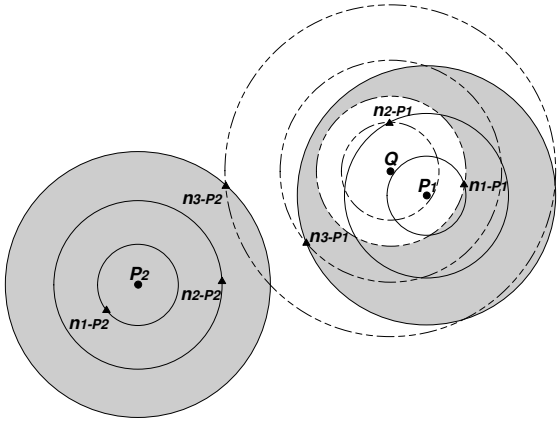

Fig. 6b. Mobile host Q can only retrieve uncertain nearest neighbors from peer P_2.

Figure 6: The mobile host Q and its two closest peers, P_1 and P_2.

3.2.2 Multiple Peer NN Verification: kNN$_{multiple}$

Under some conditions the kNN$_{single}$ method may not be able to verify all k nearest neighbors. Therefore, we extend the verification process to include results from multiple peers simultaneously. Figure 7 demonstrates an example in which a point of interest (n_{2-P3}) cannot be verified by the NN data set of a single peer; neither with peer P_3 nor with peer P_4. The kNN$_{multiple}$ method combines the area of all the peers, each bounded by the outermost NN circle, into a *certain region* R_c (the shaded area in Figure 7c). It is expensive to compute an exact solution for R_c. Therefore, we adopt a polygonization technique that transforms all the certain area circles into polygons to closely approximate the certain area reported by each peer. After this transformation the polygons can be merged together as a certain region R_c by performing the *MapOverlay* algorithm [6]. The kNN$_{multiple}$ verification technique is executed based on R_c similarly to kNN$_{single}$. Lemma 3.8 provides the rules for verifying nearest neighbors with multiple peers.

Lemma 3.8 *If the nearest neighbor data set NN_P is composed of data from j peers, the certain region R_c can be represented as:*

$$R_c = P_{1-area} \cup P_{2-area} \cup \cdots \cup P_{j-area}.$$

For any interest object n_i in R_c, the distance between Q and n_i is used as a radius to draw a circle C_{ni}. If C_{ni} is fully covered by R_c, then n_i is a certain NN of Q.

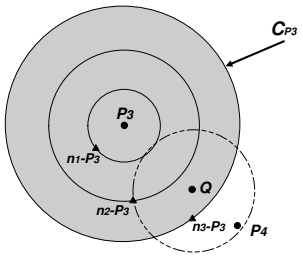

Fig. 7a. Mobile host Q cannot verify POI n_{2-P3} as a certain NN with peer P_3.

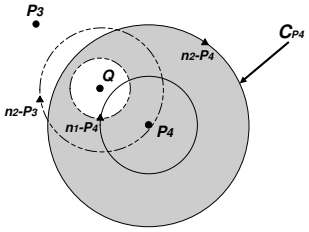

Fig. 7b. Mobile host Q cannot verify POI n_{2-P3} as a certain NN with peer P_4, either.

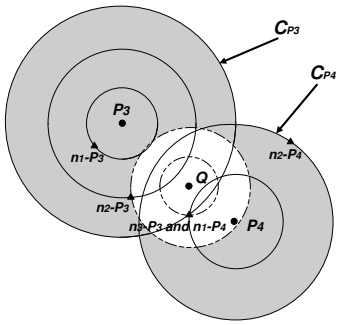

Fig. 7c. After merging the certain regions of peer P_3 and P_4, POI n_{2-P3} can be verified as a certain object.

Figure 7: An example of multiple peers NN verification. The point of interest n_{2-P3} can only be verified as a NN of Q based on the region of both peer P_3 and peer P_4.

3.3 Nearest Neighbor Query Pruning Bounds

We assume that the spatial database server executes an efficient k-nearest neighbor search algorithm based on R-tree indexing [10] for solving kNN queries. The NN search is supported with a priority queue containing the nodes visited so far. Initially the priority queue contains the entries of the R-tree root sorted according to their minimum distance (MINDIST) to the query point Q. In general most of the moving objects have executed either one or both $k\text{NN}_{single}$ and $k\text{NN}_{multiple}$ processes before forwarding kNN search queries to the server. Hence, it is worthwhile to calculate branch expanding upper and lower bounds from the entries in heap H to speed up the NN search process at the server. The heap H is in one of six different states after a mobile host has executed both the $k\text{NN}_{single}$ and $k\text{NN}_{muitiple}$ mechanisms without retrieving k certain objects:

- State 1: H is full and it contains both certain and uncertain entries.
- State 2: H is full and it contains only uncertain entries.
- State 3: H is not full and it contains both certain and uncertain entries.
- State 4: H is not full and it contains only certain entries.
- State 5: H is not full and it contains only uncertain entries.
- State 6: H contains no entry.

In State 1 there may exist some POIs which are closer to Q compared with the last element in H. Hence, we can consider the last entry of H as the final candidate nearest neighbor in the NN search and forward its distance attribute to the server as the branch expanding upper bound. In addition, the distance attribute D_{ct} of the last certain entry can be another bound, the branch expanding lower bound. Because we are certain about the POIs within the circle region C_r with radius D_{ct} and center point Q, the NN search algorithm executed in the server does not need to expand any minimum bounding rectangle which is completely covered by C_r. Conversely, when H is full and contains just uncertain entries, we can infer only the upper bound (State 2). In States 3 and 4 after the mobile host performed the two algorithms, there have merely less than k interest objects been found. Therefore, we can only

infer the lower bound from the distance attribute of the last certain element in H. In the last two states, H is not full and contains only uncertain entries or no entry at all. Consequently we cannot infer any bounds from them.

To take advantage of the two new bounds for kNN queries, we slightly modified the kNN best-first search algorithm such that it calculates one more metric, the maximum distance (MAXDIST), for pruning R-tree branches. MAXDIST indicates which MBRs are totally covered in region C_r and the algorithm does not need to expand them. Furthermore, we added two new MBR pruning strategies for the kNN search as follows:

1. Any MBR M with MAXDIST(Q, M) smaller than the branch expanding lower bound is pruned (downward pruning).
2. Any MBR M with MINDIST(Q, M) greater than the Euclidean distance from Q to the k^{th} element in H is discarded (upward pruning).

The complete algorithm of the sharing based kNN query is described in Algorithm 1.

Algorithm 1 SENN: Sharing-based Euclidean distance Nearest Neighbor query (Q, k)

```
/* Q is the query mobile host */
Query moving object peers within the communication range
Sort the peer query results NN_P according to their last query locations
for each element e of NN_P do
    kNN_single(Q, k)
end for
if a certain kNN set is found by kNN_single then
    return kNN set
else
    kNN_multiple(Q, k)
end if
if a certain kNN set is found by kNN_muitiple then
    return kNN set
end if
if the heap H is full and an uncertain kNN set is acceptable by the mobile host then
    return the uncertain kNN set
else
    /* The heap H is not full or an uncertain kNN set is not acceptable */
    Query the server with pruning upper bound and lower bound, if available
    /* Forward the branch expanding upper bound and the branch expanding lower bound found by
    kNN_muitiple */
    return the kNN set
end if
```

3.4 Spatial Network Nearest Neighbor Queries

In the real world, mobile objects often move on pre-defined networks (e.g., roads, railways, etc.). In this scenario, the spatial network distance provides a more exact estimation of the travel distance between two objects. Papadias et al. [13] have proposed a technique to solve spatial network nearest neighbor queries. However, to the best of our knowledge, sharing based spatial network nearest neighbor queries

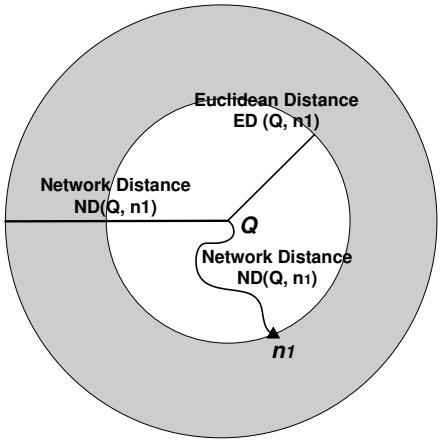

Fig. 8a. The 1^{st} Euclidean NN.

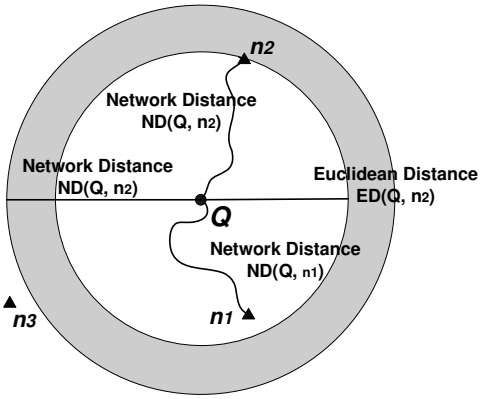

Fig. 8b. The 2^{nd} Euclidean NN.

Figure 8: Nearest neighbor search in a spatial network environment with the IER algorithm.

have not been studied before. Here, we extend our sharing based Euclidean distance nearest neighbor method to support applications in spatial network environments.

Leveraging existing methods, we assume a digitization process that generates a modeling graph from an input spatial network. The modeling graph contains three categories of graph nodes: the network junctions, the start/end points of a road segment, and other auxiliary points. The shortest path between two nodes can be computed with Dijkstra's algorithm [7], which is leveraged as the basis for computing the network distance between any two arbitrary points. For two nodes i and j, we observe that their Euclidean distance $ED(n_i, n_j)$ always provides a lower bound on their network distance $ND(n_i, n_j)$. We refer to this fact as the *Euclidean lower bound property*. Papadias et al. proposed two algorithms for nearest neighbor queries in spatial network databases: the *Incremental Euclidean Restriction* (IER) and the *Incremental Network Expansion* (INE). Here we extend the IER algorithm to utilize cached NN query results in a P2P sharing infrastructure.

Incremental Euclidean Restriction The IER algorithm is based on the multi-step kNN techniques [8, 16]. To execute a nearest neighbor search for query point Q, IER first retrieves the Euclidean distance nearest neighbor n_1 of Q and computes the Euclidean distance $ED(Q, n_1)$. Next it calculates the network distance from Q to n_1, $ND(Q, I1)$. Subsequently we can use Q as the center to draw two concentric circles with radii $ED(Q, n_1)$ and $ND(Q, n_1)$. Due to the *Euclidean lower bound property*, objects closer to Q than n_1 in the network must be within the network distance, $ND(Q, n_1)$. Therefore, the search space becomes the ring area between the two circles as shown in Figure 8a. In the next iteration, the second closest object n_2 is retrieved (by Euclidean distance). Since in our example $ND(Q, n_2) < ND(Q, n_1)$, n_2 becomes the current candidate for spatial network nearest neighbor and the search upper bound becomes $ND(Q, n_2)$. This procedure is repeated until the next Euclidean nearest neighbor is located beyond the search region (as n_3 is in Figure 8b). The extension of IER to a kNN search is straightforward.

In our P2P environment, we assume that each mobile host retains the data of the local spatial network modeling graph. A mobile host Q executes the SENN algorithm first to obtain k certain nearest neighbors (by Euclidean distance) and then calculates the network distance of the k objects based on its local spatial network modeling graph. The resulting objects are sorted in ascending order of their network distance to Q and the Euclidean distance of $ND(Q, I_k)$ between Q and the k^{th} object becomes the search upper bound, S_{bound}. Next, Q retrieves the subsequent Euclidean distance nearest

Algorithm 2 SNNN: Sharing-based Network distance Nearest Neighbor query(Q, k)

```
/* Q is the query mobile host */
Execute the Sharing-based Euclidean distance Nearest Neighbor (SENN) query algorithm for retrieving k nearest neighbors {n_1, ..., n_k}
for each object n_i do
   Compute its network distance to Q
end for
Sort {n_1, ..., n_k} in ascending order according to their network distance to Q
Set ND(Q, n_k) as the upper search bound S_bound
i = 1
repeat
   n_next= the object with the longest distance to Q which is returned by SENN(Q, k + i)
   if ND(Q, n_next)<ND(Q, n_k) then
      /* The next Euclidean NN is closer than the k^th NN */
      Replace n_k with n_next
      Sort {n_1, ..., n_k} in ascending order according to their network distance to Q
      Set S_bound=ND(Q, n_k)
   end if
   i = i+1
until ED(Q, n_next)>S_bound
```

neighbors incrementally from its peers or the spatial database server and keeps updating the k candidate spatial network NNs until the next Euclidean NN falls beyond the search region. This Sharing-based Network distance Nearest Neighbor (SNNN) algorithm is formalized in Algorithm 2.

4 Experimental Validation

To evaluate the performance of our approach we have implemented our sharing based spatial query algorithms within a simulator. The main objective of our peer-to-peer design is to increase scalability in two dimensions. First, the server access workload can be reduced as queries are answered directly by peers. Second, for the remaining queries that must be sent to the server our technique diminishes the search overhead by providing pruning-bounds for the R-tree algorithm. Consequently, the focus of our simulation is on quantifying the server load variations as a function of two main parameters, the *spatial query request rate* (SQRR) and *page access rate* (PAR). SQRR is for quantifying how many percent of the total client spatial queries are required to be processed by the spatial database server and PAR is for evaluating server side memory (primary and secondary) access rate for a sequence of spatial queries. We have performed our experiments with both synthetic and real-world parameter sets.

4.1 Simulator Implementation

Our simulator consists of two main modules, the *mobile host module* and the *server module*. The objective of the mobile host module is to generate and control the movements and query launch patterns of all mobile hosts. Each mobile host is an independent object which decides its movement autonomously. The server module processes spatial searches and is responsible for estimating the I/O load of the spatial database server. Spatial data indexing is provided with the well known R*-tree algorithm [1]. We implemented our SENN query algorithm in the mobile host module.

Parameter	Description
POI_{Number}	The number of point of interest in the system
MH_{Number}	The number of mobile host in the simulation area
C_{Size}	The cache capacity of each mobile host
$M_{Percentage}$	The mobile host movement percentage
$M_{Velocity}$	The mobile host movement velocity (mph)
λ_{Query}	The mean number of queries per minute
Tx_{Range}	The transmission range of queries
λ_{kNN}	The mean number of queried nearest neighbors
$T_{execution}$	The length of a simulation run

Table 2: Parameters for the simulation environment.

Each mobile host is implemented as an independent object that encapsulates all its related parameters such as the movement velocity $Move_{Velocity}$, the cache capacity C_{Size}, the wireless transmission range Tx_{Range}, etc. All mobile hosts move inside a geographical area and the measure can be decided by users (in our experiment, we adopt a 2 miles by 2 miles area and a 30 miles by 30 miles area). Additionally, user adjustable parameters are provided for the simulation such as execution length, the number of mobile hosts and their query frequency, the number of POIs, etc. Table 2 lists all of the simulation parameters.

The simulation is initialized by randomly choosing a starting location for each MH within the simulation area. The movement generator then generates trajectories with two different modes, the *free movement mode* and the *road network mode*. In the former mode, each mobile host moves obstacle-free within the environment and the movement velocity is fixed. The road network mode is more realistic since MHs follow an underlying road network and their travel speed s is determined by the speed limit on the corresponding road segment. We employed the *random waypoint* model [2] as our mobility model. Each MH selects a random destination inside the simulation area and progresses towards it. When reaching that location, it pauses for a random interval and decides on a new destination for the next travel period. This process repeats for all MHs until the end of the simulation.

Every simulation has numerous intervals (whose lengths are Poisson distributed) and at the end of such an interval, the simulator selects a random subset of the mobile hosts to launch spatial queries. The subset size is controlled via the λ_{Query} parameter (e.g., 500 queries per minute). These hosts then execute the SENN algorithm by interacting with their peers. A mobile host will first attempt to answer each spatial query from its local cache and via the SENN algorithm. If this is unsuccessful, the query will be forwarded to the remote database server. Each mobile host manages its local NN query result cache with a combination of the following two policies:

1. A MH only stores the query location (the coordinates where it launched the query) and all the certain nearest neighbors of the most recent query.

2. If a kNN query must be sent to the server, the MH will query for as many NN as its cache capacity allows (e.g., if the cache capacity is 10, the query will be for 10-NNs).

The single peer nearest neighbor verification process is implemented according to the algorithm detailed in Section 3.2.1. A MH sequentially verifies the candidate points of interest starting with the results obtained from its closest peer query location (using Euclidean distance). In the multiple peer verification algorithm of Section 3.2.2, multiple, potentially overlapping circles must be combined to provide the verification area. We utilize a polygonization technique that transforms all the peer certain area circles into polygons and then sequentially merges them into a *certain region* R_c by performing the

MapOverlay algorithm. Afterwards, a MH can verify POIs with the $k\text{NN}_{multiple}$ verification technique based on R_c.

4.1.1 Simulation Parameter Sets

To obtain results that closely correspond to real world conditions we obtained our simulation parameters from data sets that report, for example, car and gas station densities in urban areas. We term the two parameters sets based on these real-world statistics the *Los Angeles County* parameter set and the *Riverside County* parameter set.

- Points of Interest: We obtained information about the density of interest objects (e.g., gas stations, restaurants, hospitals, etc.) in the Greater Los Angeles area from two online sites: GasPriceWatch.com[2] and CNN/Money[3]. Because gas stations are commonly the target of kNN queries, we use them as the sample POI type for our simulations. The server load variations of other POI types are expected to be very similar.
- Mobile Hosts: We collected vehicle statistics of the Greater Los Angeles area from the Federal Statistics web site[4]. The data provide the number of registered vehicles in the Los Angeles and Riverside Counties (5,498,554 and 944,645, respectively). In our simulations we assume that about 10% of these vehicles are on the road during non-peak hours according to the traffic information from Caltrans[5]. We further obtained the land area of each county to compute the average vehicle density per square mile.

Parameter	Los Angeles County	Riverside County	Synthetic Suburbia	Units
POI_{Number}	16	5	11	
MH_{Number}	463	50	257	
C_{Size}	10	10	10	
$M_{Percentage}$	80	80	80	%
$M_{Velocity}$	30	30	30	mph
λ_{Query}	23	2.5	13	min^{-1}
Tx_{Range}	200	200	200	m
λ_{kNN}	3	3	3	
$T_{execution}$	1	1	1	hr

Table 3: The simulation parameter sets for the Los Angeles, the Riverside Counties, and the synthetic suburbia of a 2 miles by 2 miles area.

The Los Angeles and the Riverside County parameter sets represent very dense, urban area and a low-density, more rural area. Hence, for comparison purposes we blended the two real parameter sets together to generate a third, synthetic parameter set. The synthetic data set demonstrates vehicle and interest object densities in-between Los Angeles County and Riverside County, representing a suburban area. Table 3 lists the three parameter sets of a 2 miles by 2 miles area and Table 4 demonstrates the parameter sets of a 30 miles by 30 miles area.

[2] http://www.gaspricewatch.com
[3] http://money.cnn.com/
[4] http://www.fedstats.gov/
[5] http://www.dot.ca.gov/hq/traffops/saferesr/trafdata/

Parameter	Los Angeles County	Riverside County	Synthetic Suburbia	Units
POI_{Number}	4050	2160	3105	
MH_{Number}	121500	11700	66600	
C_{Size}	20	20	20	
$M_{Percentage}$	80	80	80	%
$M_{Velocity}$	30	30	30	mph
λ_{Query}	8100	780	4440	min^{-1}
Tx_{Range}	200	200	200	m
λ_{kNN}	5	5	5	
$T_{execution}$	5	5	5	hr

Table 4: The simulation parameter sets for the Los Angeles, the Riverside County, and the synthetic suburbia of a 30 miles by 30 miles area.

4.1.2 Road Network Generation

We generated our road network from the TIGER/LINE street vector data available from the U.S. Census Bureau[6]. The road segments belong to several different categories, such as primary highways, secondary and connecting roads, and rural roads. The segments associated with a different road classes are associated with different maximum driving speeds. Each mobile hosts monitors the speed limit on the road that it is currently traveling on and adjusts its velocity accordingly. One of the challenges when integrating road segments into a complete road network is to isolate intersecting paths and determine if they are indeed intersections. For example, freeways generally project many intersections in two-dimensional space, but many of them are over-passes or bridges. Our solution is to detect intersection points with the help of their endpoint coordinates. In addition, differing road classes let us distinguish over-passes from intersections.

4.2 Experimental Results with the Road Network Mode

We used all three input parameter sets – Los Angeles County, Riverside County, and Synthetic Suburbia – to simulate our peer sharing techniques in conjunction with the *road network mode*. We varied the following parameters to observe their effect on the system performance: the wireless transmission range, the cache capacity, the movement velocity, and the nearest neighbor number k. The performance metric in the mobile host module was SQRR. The primary difference between the three different parameter sets is the vehicle and the POI density. Hence, we utilized the simulation to verify the applicability of our design to different geographical and urban areas. All simulation results were recorded after the system reached steady state.

4.2.1 Effect of the Transmission Range

In our first experiment we varied the mobile host wireless transmission range from 10 meters to 200 meters, with all other parameters unchanged. We chose 200 meters as a practical upper limit on the transmission range of the IEEE 802.11 technology. Because of obstacles such as buildings, this range could diminish to 100 meters or less in urban areas. Figure 9 and 10 illustrate percentage of the queries that can be resolved by one peer, multiple peers or the server with the Los Angeles County, the Synthetic Suburbia, and the Riverside County parameter sets, respectively in the two simulation regions. As the

[6]http://www.census.gov/geo/www/tiger/

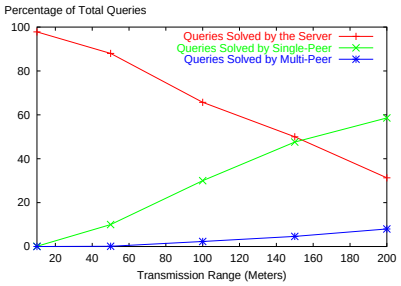

Fig. 9a. Los Angeles County.

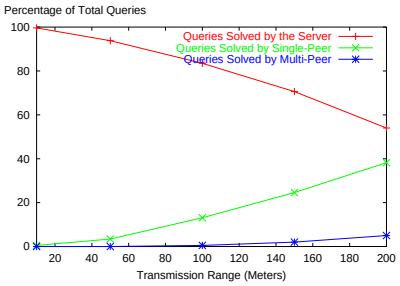

Fig. 9b. Synthetic Suburbia.

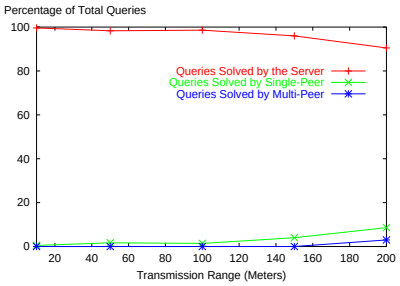

Fig. 9c. Riverside County.

Figure 9: The percentage of queries that are resolved by one peer, multiple peers and the server as a function of the wireless transmission range of a 2 miles by 2 miles area.

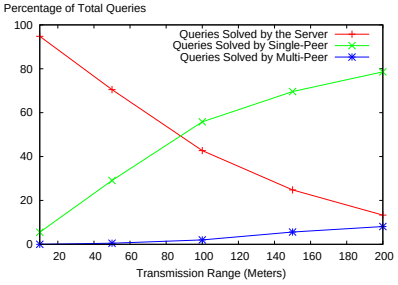

Fig. 10a. Los Angeles County.

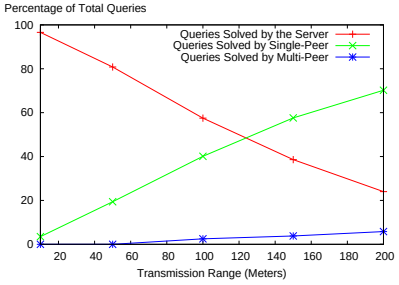

Fig. 10b. Synthetic Suburbia.

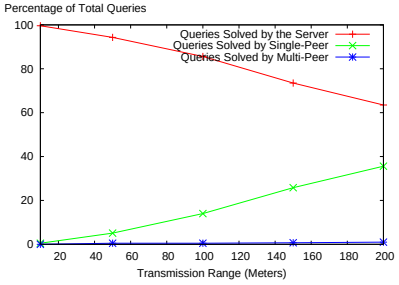

Fig. 10c. Riverside County.

Figure 10: The percentage of queries that are resolved by one peer, multiple peers and the server as a function of the wireless transmission range of a 30 miles by 30 miles area.

transmission range extends, an increased number of queries can be answered by surrounding peers. As expected, the effect is most pronounced in Los Angeles County, because of its high vehicle density. At a transmission range of 200 m only around $20\% \sim 30\%$ of the queries must be sent to the server.

4.2.2 Effect of the MH Cache Capacity

Next we varied the mobile host cache capacity. The cache capacity denotes how many nearest neighbor objects a mobile host can store. Figure 11 and 12 illustrate cache capacity changes from 1 to 9 and 4 to 20 respectively with the three parameter sets. In Figure 12a, even though the number of interest objects is much larger than the maximum capacity of the cached NN query results, we can find a remarkable server workload decrease with a higher MH cache capacity. In Figure 11c, however, because of the sparseness of interest objects, the server workload becomes stabilized after a cache capacity of five.

4.2.3 Effect of the MH Movement Velocity

We studied the effect of host movement velocity by changing the MH velocity from 10 miles per hour (mph) to 50 mph and the results are shown in Figure 13 and 14. We observe that the movement velocity has a stronger effect on the server workload in areas with a lower vehicle and interest object density. However, the effect is quite gradual in all cases.

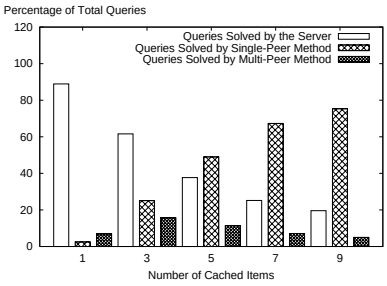

Fig. 11a. Los Angeles County.

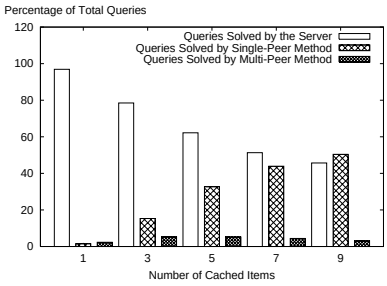

Fig. 11b. Synthetic Suburbia.

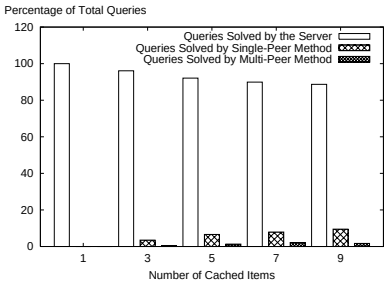

Fig. 11c. Riverside County.

Figure 11: The percentage of queries that are resolved by one peer, multiple peers and the server as a function of the mobile host cache capacity of a 2 miles by 2 miles area.

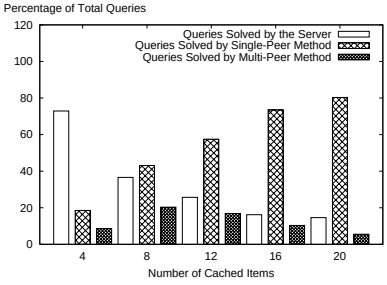

Fig. 12a. Los Angeles County.

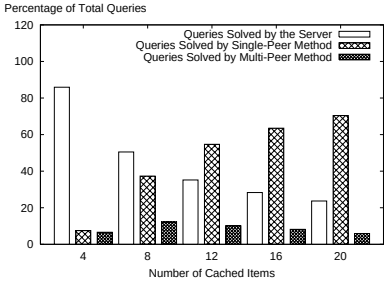

Fig. 12b. Synthetic Suburbia.

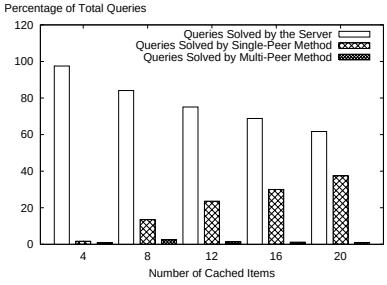

Fig. 12c. Riverside County.

Figure 12: The percentage of queries that are resolved by one peer, multiple peers and the server as a function of the mobile host cache capacity of a 30 miles by 30 miles area.

4.2.4 Effect of k

We were also interested in the effect that varying the number of requested nearest neighbors, i.e., k, would have on the system performance. In our simulation we chose k randomly for each host and each query in the range from 1 to 9 and 3 to 15 respectively in the two regions. Figure 15 and 16 illustrate the results. The server workload of the Los Angeles County parameter set increases 68% and 29% when we raise the k from 1 to 9 and from 3 to 15 in the two regions. The server workload of the Riverside County parameter set increases by only 11% and 19%, because its starting level is much higher. Not surprisingly result sharing is much more effective for small values of k.

4.3 Experimental Results from the Free Movement Mode

We executed the simulations in *free movement mode* with otherwise the same parameter settings as before with the *free movement mode*. We observe from the experimental results that the server workload with the Los Angeles County parameter set decreases between 5% and 8% (in the 2 miles by 2 miles area) and 2% to 5% (in the 30 miles by 30 miles area) under all conditions. The results of the synthetic and the Riverside parameter sets are very close to their counterparts of the road network mode. Because mobile hosts do not have to follow any underlying road network for their movements, this change decreases the distance between mobile hosts compared with the road network mode and hence slightly increases the performance of our sharing algorithm. This effect is more evident in regions with a high vehicle density such as Los Angeles County.

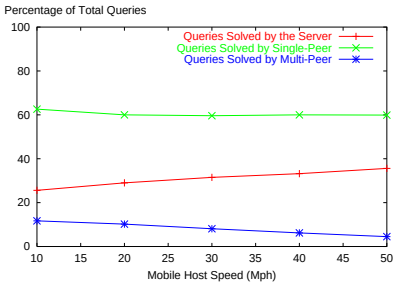

Fig. 13a. Los Angeles County.

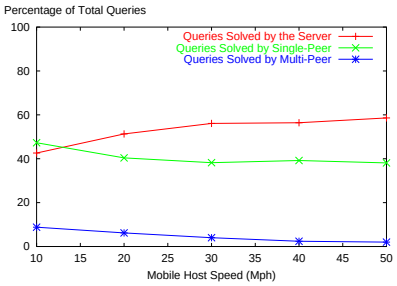

Fig. 13b. Synthetic Suburbia.

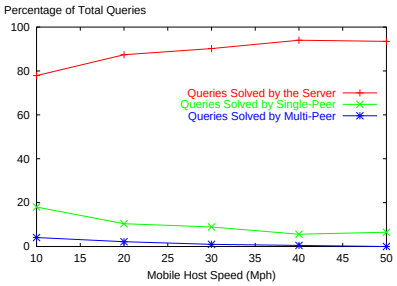

Fig. 13c. Riverside County.

Figure 13: The percentage of queries that are resolved by one peer, multiple peers and the server as a function of the mobile host movement velocity of a 2 miles by 2 miles area.

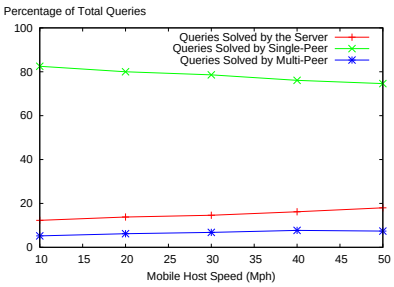

Fig. 14a. Los Angeles County.

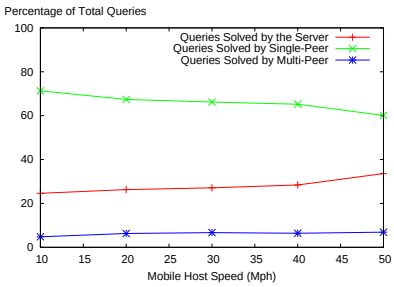

Fig. 14b. Synthetic Suburbia.

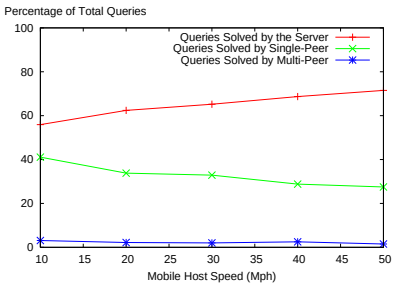

Fig. 14c. Riverside County.

Figure 14: The percentage of queries that are resolved by one peer, multiple peers and the server as a function of the mobile host movement velocity of a 30 miles by 30 miles area.

4.4 Experimental Results from the Spatial Database Server

In order to evaluate the nearest neighbor query pruning bounds of Section 3.3, we extended the R-tree incremental nearest neighbor (INN) algorithm [10] with one more metric, MAXDIST. For each incoming NN query from mobile hosts, the server module executes both the original INN algorithm and our extended INN algorithm with pruning bounds (denoted by EINN) to compare the performance improvement with respect to page accesses. We examined the behavior of the two algorithms as the number of k increases. We utilized the R*-tree for indexing the POI data set (gas station locations) in the server module. The R*-tree has an advantage in query response time over the conventional R-tree algorithm by utilizing more sophisticated insertion and node-splitting methods, which attempt to minimize a combination of overlap between bounding rectangles and the total area. The branching factor of both the index and leaf nodes was set to 30. Because NN queries are generated by randomly selected mobile hosts, query points are uniformly distributed over the simulation area. The experiments are executed sufficiently often to obtain consistent results.

At the end of a spectrum there are two extreme I/O behaviors of the spatial database server: all requested memory pages are found in main memory or every I/O leads to disk activity. In the former case, because of fast main memory access, we cannot discern a significant performance difference between INN and EINN. However, any reasonably large data set will not fit into main memory and more disk I/Os will be performed. Hence, the database I/O behavior is closer to the other end of the spectrum. Since the EINN usually requests fewer R*-tree nodes and objects than INN, we believe that the kNN search algorithm with query pruning bounds (EINN) will have good scalability with large data sets.

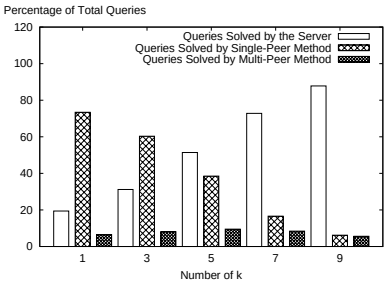

Fig. 15a. Los Angeles County.

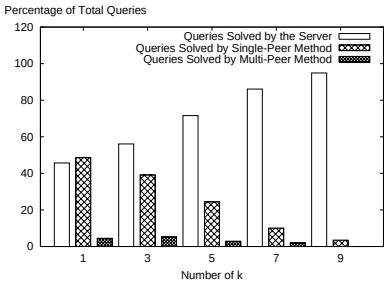

Fig. 15b. Synthetic Suburbia.

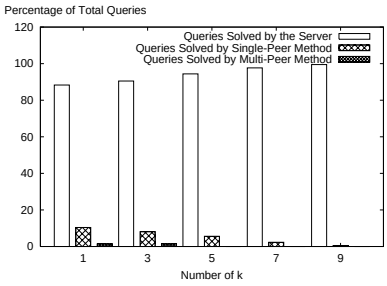

Fig. 15c. Riverside County.

Figure 15: The percentage of queries that are resolved by one peer, multiple peers and the server as a function of k of a 2 miles by 2 miles area.

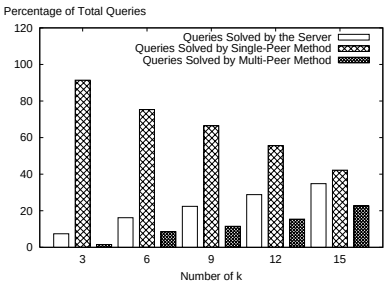

Fig. 16a. Los Angeles County.

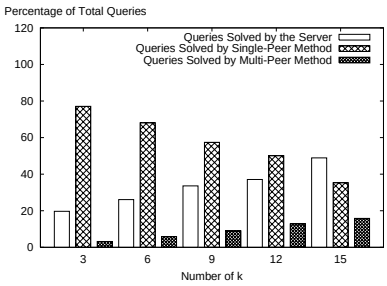

Fig. 16b. Synthetic Suburbia.

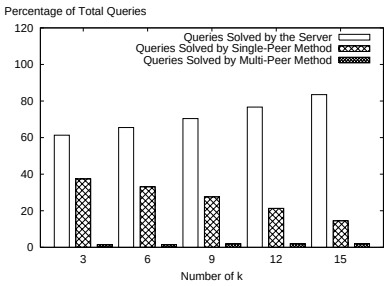

Fig. 16c. Riverside County.

Figure 16: The percentage of queries that are resolved by one peer, multiple peers and the server as a function of k of a 30 miles by 30 miles area.

During the simulation process, the server module counts the number of R*-tree node (index nodes and data nodes) accesses which correspond to both main memory and disk I/Os. According to our observation, the number of node accesses provides a good predication of the actual NN query I/O cost.

Next, we varied the number of k with all the three parameter sets with both the EINN and INN algorithms. The server module recorded the relevant R*-tree page access information (Section 4.2.4). As shown in Figure 17, the EINN algorithm performs consistently better than INN, while the rate of growth is similar for both. We conclude that the pruning bounds can always decrease the number of page accesses. We varied the number of k from 3 to 15 with the three parameter sets and the EINN algorithm accesses 10% to 21% fewer pages than INN.

We conclude from all the performed experiments that the mobile host density has a considerable impact on the *spatial query request rate*. As a result, if more mobile hosts travel in a specific area, each MH has a higher opportunity to fulfill its kNN queries by peers. Furthermore, the nearest neighbor query pruning bounds also have a significant positive effect on the *page access rate* and successfully decrease the server load.

5 Conclusions and Future Work

We have presented a novel approach for answering spatial nearest neighbor search queries by leveraging results from neighboring peers within a mobile environment. Significantly, our method allows a mobile peer to locally verify whether candidate objects received from neighbors are indeed part of its own

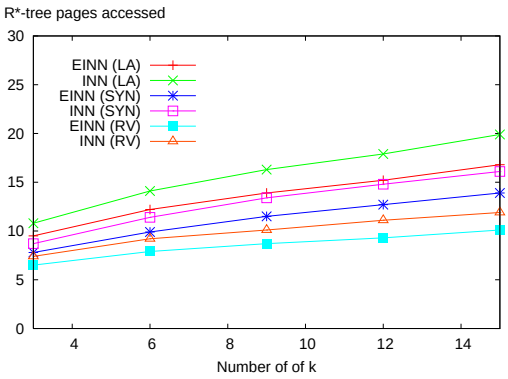

Figure 17: The page access comparison of EINN and INN as a function of k.

nearest neighbor data set. Our simulation results indicate that the technique can reduce the access traffic to remote servers by a significant amount, for example up to 80% in a dense urban area. This is achieved with minimal caching at the peers. By virtue of its peer-to-peer architecture, the method exhibits great scalability: the higher the mobile peer density, the more queries can be answered by peers. Therefore, the load on the remote databases increases sub-linearly with the number of clients. We plan to extend our work to investigate other types of spatial queries, such as range and spatial join searches.

6 Acknowledgments

This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), CMS-0219463 (ITR), and equipment gifts from the Intel Corporation, Hewlett-Packard, Sun Microsystems and Raptor Networks Technology.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[2] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the 4^{th} ACM/IEEE MobiCom*, pages 85–97, 1998.

[3] C.-Y. Chow, H. V. Leong, and A. Chan. Peer-to-Peer Cooperative Caching in Mobile Environment. In *Proceedings of the 24^{th} International Conference on Distributed Computing Systems Workshops*, pages 528–533, 2004.

[4] C.-Y. Chow, H. V. Leong, and A. T. S. Chan. Group-based Cooperative Cache Management for Mobile Clients in a Mobile Environment. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, Montreal, Quebec, Canada, August 15-18, 2004.

[5] M. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Procedings of the 1^{st} USENIX OSDI*, pages 267–280, November 1994.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications (2nd Edition)*. Springer, 2000.

[7] E. W. Dijkstra. *A Note on Two Problems in Connection with Graphics*, volume 3. Numeriche Mathematik, 1959.

[8] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 419–429, 1994.

[9] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, Massachusetts, June 18-21, 1984.

[10] G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.

[11] H. Hu, J. Xu, W. S. Wong, B. Zheng, D. L. Lee, and W.-C. Lee. Proactive Caching for Spatial Queries in Mobile Environments. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, Tokyo, Japan, April 5-8, 2005.

[12] M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *Proceedings of the 30th International Conference on Very Large Databases (VLDB)*, pages 840–851, Toronto, Canada, August 31 - September 3, 2004.

[13] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query Processing in Spatial Network Databases. In *Proceedings of the International Conference on Very Large Databases*, pages 790–801, 2003.

[14] Q. Ren, M. H. Dunham, and V. Kumar. Semantic Caching and Query Processing. *Transactions on Knowledge and Data Engineering (TKDE)*, 15(1):192–210, January/February 2003.

[15] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, San Jose, CA, May 22-25, 1995.

[16] T. Seidl and H.-P. Kriegel. Optimal Multi-step k-Nearest Neighbor Search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 154–165, 1998.

[17] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh. A Road Network Embedding Technique for k-Nearest Neighbor Search in Moving Object Databases. In *Proceedings of the Tenth ACM International Symposium on Advances in Geographic Information Systems*, pages 94–100, McLean, Virginia, November 2002.

[18] Z. Song and N. Roussopoulos. *k*-Nearest Neighbor Search for Moving Query Point. In *Proceedings of Advances in Spatial and Temporal Databases, 7th International Symposium (SSTD)*, pages 79–96, Redondo Beach, CA, July 12-15, 2001.

[19] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB)*, pages 287–298, Hong Kong, China, August 20-23, 2002.

[20] D. Wessels and K. Claffy. ICP and the Squid Web Cache. *IEEE Journal on Selected Areas in Communications (JSAC)*, pages 345–357, March 1998.

[21] L. Yin and G. Cao. Supporting Cooperative Caching in Ad Hoc Networks. In *IEEE INFOCOM 2004*, Hong Kong, China, March 7-11, 2004.

[22] B. Zheng, W.-C. Lee, and D. L. Lee. On Semantic Caching and Query Scheduling for Mobile Nearest-Neighbor Search. *Wireless Networks*, 10(6):653–664, November 2004.