Wireless Networks 10, 653–664, 2004

On Semantic Caching and Query Scheduling for Mobile Nearest-Neighbor Search

BAIHUA ZHENG *
Singapore Management University, 469 Bukit Timah Road, Singapore 259756

WANG-CHIEN LEE
The Penn State University, University Park, PA 16802, USA

DIK LUN LEE
Singapore Management University, 469 Bukit Timah Road, Singapore 259756

Abstract. Location-based services have received increasing attention in recent years. In this paper, we address the performance issues of *mobile nearest-neighbor search*, in which the mobile user issues a query to retrieve stationary service objects nearest to him/her. An index based on *Voronoi Diagram* is used in the server to support such a search, while a semantic cache is proposed to enhance the access efficiency of the service. Cache replacement policies tailored for the proposed semantic cache are examined. Moreover, several query scheduling policies are proposed to address the *inter-cell roaming* issues in multi-cell environments. Simulations are conducted to evaluate the proposed methods. The result shows that the system performance, in terms of cache hit ratio, query response time, cell-cross number and cell-recross number, is improved significantly.

Keywords: location-based services, nearest-neighbor search, Voronoi Diagram, indexing technique, semantic caching, query scheduling, roaming

1. Introduction

Owing to increasing demands from mobile users, *Location-Based Services* (LBSs) have received a lot of attention in recent years. Examples of queries for location-based services include "find the nearest gas station from my current location", "find all the cinemas within 1 km radius", "which buses will pass by me in the next 10 minutes?", and so on. While data objects in the first two examples are stationary, those in the last example are mobile. In this paper, we focus on queries issued by mobile users on relatively static data objects, because they are the most common kind of queries in LBSs. The movement of mobile clients presents many new research problems for location-dependent query processing [3,11].

There are several technical issues involved with the implementation of an LBS, which include locating the position of a mobile user, tracking and predicting movements, processing queries efficiently, and bounding location errors. In this paper, we focus on the efficient processing of location-dependent queries and, in particular, a sub-class of queries called *mobile nearest-neighbor* (*NN*) *search*. A mobile *NN* search is issued by a mobile client to retrieve stationary service objects nearest to its user.[1] It is an important function for LBSs, but the implementation is difficult since the clients are mobile and queries must be answered based on the clients' current locations. If a client keeps moving after it issued a query, the query result would continue to change in accordance with the client's movement. As such, it is difficult to obtain results which are accurate with respect to the position at which the user receives them.

Despite the fact that LBSs open up new research opportunities, most of the on-going research work still concentrates on traditional queries which return answers independent to the locations of the query issuers. In other words, each data object has only one set of attribute values in the server. If a client caches a local copy of the data to improve performance, the cached data become invalid only when the corresponding copy in the server is updated. As for location-dependent queries, a data object usually has multiple sets of attribute values, each of which is valid only when the client is located within a specific region. While mobile data caching and invalidation for location-independent queries has been actively pursued in the mobile computing research community, very few work had been done on indexing and query processing techniques for location-dependent queries.

In this paper, we propose an elegant indexing mechanism to support mobile nearest-neighbor search. The index is based on the *Voronoi Diagram* (VD) [7]. To the best of the authors' knowledge, this is the first time that VD is used as the basis for developing indexes for LBSs in mobile environments. In addition, to enhance access efficiency of the system, we propose a semantic caching scheme, which stores along with a data object the valid spatial scope of the data object. Since

* Corresponding author.
E-mail: bhzheng@smu.edu.sg

[1] A mobile client refers to the device used by a mobile user. When it does not cause confusion, we sometimes use mobile client to refer to a mobile user.

the cached data could become invalid due to user mobility, we propose three cache replacement policies which estimate the potential utilization of the cached data items based on their spatial scopes and other spatial properties such as velocities. Moreover, we extend our study to a multi-cell environment, where a client has the freedom to roam across different cells served by different base stations. By observing the fact that some clients may leave the cell before they receive the answers for the queries they issued in the cell, we propose several query scheduling schemes for the servers to provide fair service opportunities to all users. Finally, we conduct a simulation to evaluate the performance of the proposed schemes with respect to cache hit ratio, query response time, cell-cross number and cell-recross number. The simulation result demonstrates that the proposed methods improve system performance significantly.

In addition to identifying and exploring technical problems facing the development of an LBS, the contribution of this study is four-fold:

- An index based on Voronoi Diagram to support mobile nearest-neighbor search.
- A semantic caching scheme to address issues of access efficiency and user mobility.
- Three cache replacement policies tailored for location-dependent semantic caching.
- Four query scheduling schemes for roaming problem in multiple cell environments.

The rest of this paper is organized as follows. Section 2 provides a brief review of the related work. A Voronoi Diagram based index is introduced in section 3. The semantic caching support for LBSs, along with cache replacement policies for semantic cache, is described in section 4. The roaming problem in multi-cell environments and four query scheduling schemes are discussed in section 5. We present the simulation model for performance evaluation and the result in section 6 and section 7, respectively. Finally, section 8 concludes our work and points out future research directions.

2. Related work

It is clear that many research issues explored in this paper have been pursued in different contexts. A brief summary of related work is given as follows.

2.1. Voronoi Diagram

Voronoi Diagram (VD) is a traditional data structure in computational geometry [7]. While it has been employed in similarity search in multimedia databases recently [5,6], it is useful for many spatial applications [7], which include:

- *Nearest-neighbor queries*. Given a point set P and a query point q, determine the closest point in P to q.
- *Facility location*. Suppose that a new grocery store is to be opened in an area with several existing, competing grocery stores. One natural method to ensure the new store's business is to locate the new store as far away from the old ones as possible.
- *Path planning*. Imagine a cluttered environment through which a robot must plan a path. In order to minimize the risk of collision, the robot may like to stay as far away from all obstacles as possible.

The VD for n objects on a plane can be constructed at $O(n \log n)$ complexity using a simple sweep algorithm. However, the maintenance cost is high, especially for high-dimensional space, thus hindering the application of the VD structure for complex and dynamic datasets. In this paper, we focus on the application of the VD technique to nearest-neighbor search in LBSs. These applications have low-dimensional space and mostly static service objects, making VD a promising indexing method for supporting nearest-neighbor search.

2.2. Caching techniques

The client cache stores frequently used information so that queries can be answered without connecting to a server. In addition to answering queries promptly, the cache may still be able to answer some queries when a connection can not be established.

- *Traditional caching.* A cache stores frequently accessed data in the mobile client in order to save wireless bandwidth and improve access efficiency. Some early work on mobile client caching can be found in [1,4]. For location-dependent information, such as local traffic information, cached data need to be validated when the client changes location. Xu et al. proposed a bit-vector approach to identifying the valid scope of the data and investigated a couple of advanced methods for caches invalidation [19].
- *Semantic caching.* A semantic cache stores data and a semantic description of the data in the mobile client [12]. The semantic description enables the cache to provide partial answers to queries which do not match the cache data exactly. As such, wireless traffic can be reduced and queries may be answered in a disconnected mode. This characteristic makes a semantic cache an ideal scheme for location-dependent queries. A cache method was proposed in [13]. A tuple $S = \langle S_R, S_A, S_P, S_L, S_C \rangle$ was used to record data in the local client. S_R and S_A are, respectively, the relationships and the attributes in S; S_P is the selection conditions that data in S satisfy; S_L is the bound of the location; and S_C represents the actual content of S. When a query is received by the client, it is trimmed into two disjointed parts: a *probe query* that can be answered by some cached data in the client, and a *remainder query* that has to be transmitted to the server for evaluation.

2.3. Continuous queries

A location-dependent query becomes difficult to answer when it is submitted as a continuous query. For example, a client in a moving car may submit the query: "Tell me the room

rates of all the hotels within a 500 meter radius from me" and would like to receive updated information continuously in order to find a cheap hotel. Since the client is moving, the query result becomes time-sensitive in that each result corresponds to one particular position and has a valid duration because of location dependency. The representation of this duration and how to transmit it to the client are the major focuses of *Continuous Queries* (CQs). Sistla et al. employed a tuple $\langle S, begin, end\rangle$ to bound the valid time duration of the query result [16,17]. Based on this method, they also developed two approaches to transmitting the results to the client: an *immediate approach* and a *delayed approach*. The former transmits the results immediately after they are computed. Thus, some later updates may cause changes to the results. The latter transmits S only at time *begin*, so the results will be returned to the client periodically, thus increasing the wireless network traffic. To alleviate limitations of the above two approaches, Periodic Transmission, Adaptive Periodic Transmission, and Mixed Transmission were proposed [9,10].

2.4. Roaming

Roaming is a very important property of mobile computing systems. To allow mobile users roam from cell to cell without interrupting on-going communication and services, hand-off must proceed transparently. Research on supporting hand-off in wireless communication networks has been studied extensively [2,8,18]. However, handoff methods at the query processing level have not been addressed.

For location-based services, a client may have to resubmit an unfinished query when it roams into a new cell, because the answer returned may become invalid in the new cell. However, some mobile clients may happen to stay near the cell boundaries and frequently roam across cell boundaries, while some other clients may cross several cells in a short period of time. These clients may have to wait for a long time for the requested information if the submitted queries have to be resubmitted again every time they enter new cells. Thus, the starvation problem associated with roaming clients has to be addressed when implementing a location-based service. To the best of our knowledge, no prior work had considered this problem.

3. VD-based index for mobile nearest-neighbor search

In this section, we present the concept of VD-based index and describe the processing of mobile *NN* search. We assume that a mobile client knows its position through, say, GPS. Thus, when a client issues a query, its current position and velocity can be submitted along with a timestamp.

3.1. Voronoi Diagram

A Voronoi Diagram records information about the closest regions corresponding to a set of geometric points. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane (or in any

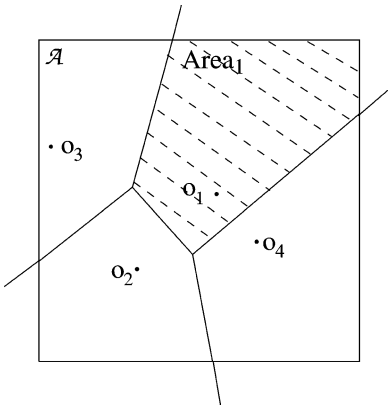


Figure 1. Voronoi Diagrams.

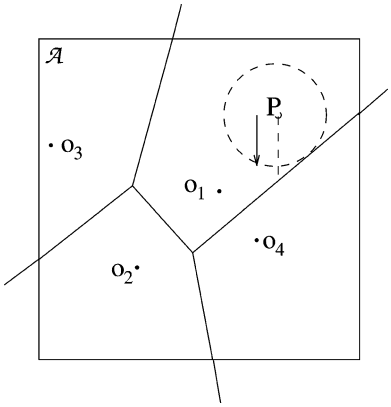


Figure 2. Semantic cache in Voronoi Diagrams.

n-dimensional space). Each of the points is called a *site*. $\mathcal{V}(p_i)$, the *Voronoi cell* for p_i, is defined as the set of points q in the plane such that $dist(q, p_i) < dist(q, p_j)$ where $i \neq j$. That is, the Voronoi cell for p_i consists of the set of points for which p_i is the unique nearest site:

$$\mathcal{V}(p_i) = \{q \mid dist(q, p_i) < dist(q, p_j),\ \forall j \neq i\}. \quad (1)$$

As shown in figure 1, all the points in the shadowed region, $Area_1$, have the same nearest fixed point, namely, O_1. In the context of mobile *NN* search, a Voronoi cell represents the spatial area within which the corresponding Voronoi site is the valid answer to any *NN* search issued within the cell. The readers should also note that the enclosing square in the figure represents the geographical region monitored by a base station. It is referred to as a *cell* or a *wireless cell*.

Since Voronoi Diagram has been studied for a long time in computational geometry, data structures for storing a Voronoi Diagram and efficient point location methods for locating a point in a region are available [7]. We use the *trapezoidal mapping* algorithm to map a given point to a Voronoi cell. Figure 3(a) shows the final trapezoidal map after decomposition, and figure 3(b) depicts the corresponding index. There are three kinds of nodes in the index: x-nodes (the circles) recording the x-coordinate of a vertex, y-nodes (the hexagons) recording a line segment, and leaf nodes (the rectangles) pointing to the trapezoids. Given a query point p, the

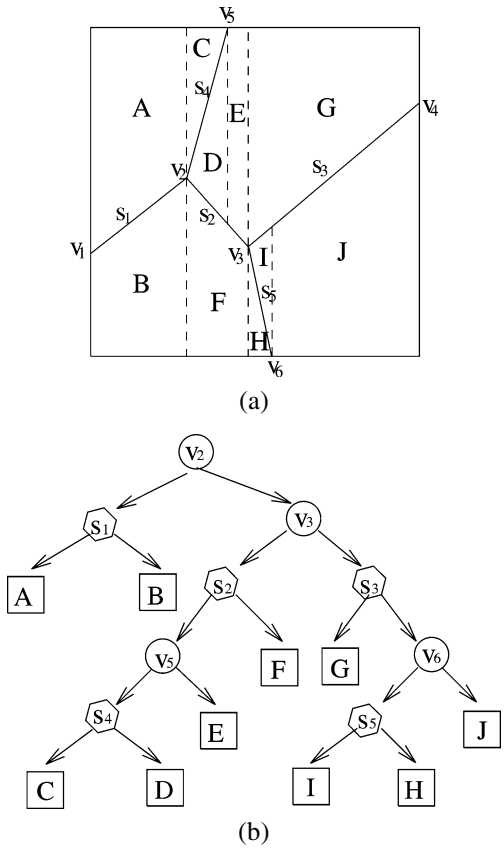


Figure 3. Index construction using trapezoidal map.

search process begins at the root node and terminates when a leaf node is reached. At an x-node, we determine if p lies to the left or to the right of the vertical line using the x-coordinate stored. At an *y-node*, we determine if p lies above or below the line segment stored. For a point lying in trapezoid E, the search path is: v_2, v_3, s_2, v_5, E.

3.2. Basic data structures

A location-based service can construct a Voronoi Diagram for a particular type of service facility (e.g., restaurants). Given the position of a mobile client, an *NN* search can be answered by first finding the Voronoi cell in which the mobile client is located and then returning the corresponding Voronoi site (i.e., the restaurant) as the closest service facility to the mobile client. To facilitate the search, a VD-based index structure has to be maintained at the server.

Efficient disk-resident indexing methods for point location have been proposed [20]. In this paper, we use three basic data structures to record the constructed Voronoi Diagram. The first one is *edge*, denoted by $\langle id, x_1, y_1, x_2, y_2\rangle$, which is used to record the edge *id* and the endpoints of an edge. The second one is *service object*, which records the position of a Voronoi site and its bounding edges. It is represented by a tuple $\langle id, x, y, number, list\rangle$, where x and y are the coordinates of the site, *number* is the number of edges bounding this site, and *list* is the list of *id*s of all the edges. The last one is *edge_service*, which records the information between the service objects and the edges using a tuple $\langle edge\_id, serv\_object\_id_1, serv\_object\_id_2\rangle$. It provides predictive information to the moving client regarding the time when it would reach the next nearest service. id_1 and id_2 are the sites above and below this edge, respectively.

3.3. Discussion

The construction and maintenance cost of VD is high, especially when the dimension of the space is high or the number of the objects is large. However, it is not an issue for LBSs, because they return service facilities based on their physical locations. As such, the search space is only two-dimensional. Furthermore, each base station only maintains the VD for the service facilities under its coverage, the number of objects indexed by a VD is small. Coupled with the fact that service facilities are not updated frequently, VD is an attractive indexing method that provides fast retrieval for a popular class of queries, namely, *NN* search.

4. Semantic caching

In this section, we introduce a semantic cache technique tailored for location-dependent information. By providing the spatial scopes of the data objects, mobile *NN* search can be answered efficiently using the semantic cache. In addition, if the velocity of mobile user is known, the duration within which the returned answer is valid can also be estimated.

4.1. Semantic circles

Given a Voronoi Diagram and the mobile client's location p, we can identify the Voronoi cell containing p and obtain a maximal circle, centered at p, within the cell. We call the circle a *semantic circle* because it represents the valid scope of an answer to a mobile *NN* search. In other words, the returned nearest service information remains valid as long as the mobile user falls within the associated semantic circle. An example of semantic circle is shown in figure 2. When the client later submits the same query and its location falls within one of the semantic circles associated with the data object, the cached value can be returned as the answer.

There are clearly more than one way of representing the valid spatial scope of a cached item. One possibility is to represent the exact shape of the Voronoi cell corresponding to the cached item. Another possibility is to use the maximal inscribe circle of the Voronoi cell. The tradeoff is between the storage overhead and the accuracy of the representation, and the cache performance. For instance, the exact representation obviously requires more computational time, cache space, and wireless bandwidth for its transmission. On the other hand, the maximal inscribe circle will produce cache misses for queries issued outside the circle. A separate study on the various representations had been presented elsewhere [21] and thus is not repeated here. We choose the semantic circle in this paper because it is compact and is able to predict the next nearest service based on the user's movement.

With the assumption that the client moves in the original velocity, the time that it departs the current cell can be approximated and the next cell that it will reach can be detected. For example, the dashed line in figure 2 represents the mobile user's expected trajectory obtained from the velocity of the user, which is denoted using the line with an arrow. The distance of this trajectory divided by the speed is the time that the mobile user leaves her current Voronoi cell. After that duration, the correct nearest service facility should be O_4, rather than O_1.

To support semantic caching, the server returns an answer to a mobile *NN* search, along with the semantic circle (i.e., radius) of the returned data object, its predicted valid time duration, and the next nearest service facility. For example, an answer of $\langle O_1, r_1, t_1, O_2 \rangle$ means that the current nearest service facility is O_1, the radius of its semantic circle is r_1, this answer is supposed to be valid in the next t_1 seconds, and after that duration the next nearest service facility is O_2. There is no need to transmit the center of a semantic circle, where the query was issued. In the case where a client changes its velocity after it submitted a query, the prediction (i.e., valid duration and next nearest service facility) may not be accurate. In order to avoid a false prediction, the client's maximum speed can be used. Within the time duration obtained by dividing the radius of the semantic circle by the maximum speed, the returned answer is guaranteed to be valid. With this scheme, the client can determine the valid duration conservatively. However, accurate prediction of the next nearest service facility is not guaranteed, since the direction of the client's movement is unknown.

In our proposed semantic cache, we store the tuple $\langle P.x, P.y, radius, O_1 \rangle$ in the client cache, where O_1 is a data object, $P.x$ and $P.y$ denote the center of the semantic circle associated with O_1, and *radius* is the radius of the circle. The readers should note that the cache may contain several semantic circles corresponding to a data object.

Considering one type of service facilities, e.g., restaurants, if a mobile client issues a query from a location within one of the semantic circles it caches, the nearest service facility within the circle is returned locally, without connecting to the server. In other cases, the query is submitted to the server and a new tuple is cached after the result is obtained.

In a wireless communication environment, the available bandwidth is limited compared to the wired networks. A dominant factor to access latency is the network latency over the wireless link. Thus, an approach to improving access latency is to reduce the number of transmissions over the wireless link. Caching techniques have been shown to be an effective way to reduce network transmissions. In simulation results shown later, it will be observed that the hit ratio of semantic caching scheme proposed in this paper is much higher than that of other existing schemes. A lot of queries can be answered at a client's cache without making connections to the base station. Consequently, the average length of service queue can be decreased, which, in turn, improves the average access latency of the clients waiting in the queue.

4.2. Query processing

Taking the semantic caching concept introduced earlier, the following summarizes the steps taken by both a client and the server to process a mobile *NN* search:

1. When a query is issued by a mobile client, the local cache is checked to see if semantic circles corresponding to the current location of the client and the requested service facility type can be found. If not, proceed to step 3; otherwise, go to step 2.
2. If there is a matching service facility data object with a valid semantic circle, the data object is retrieved locally. Go to step 5.
3. The current location of a client and its velocity are submitted along with the query to the server. The server will first locate this client in the Voronoi Diagram index, find the data object of the nearest service facility based on the client's query, and compute a semantic circle to be associated with the data object. Proceed to the next step.
4. Based on the velocity of the client and radius of the semantic circle obtained in the previous step, the time interval for the client to reach another Voronoi cell is obtained. Proceed to the next step.
5. A result is returned to the client.
6. After the client receives the result from the server, a new tuple is inserted into the client cache.

4.3. Cache replacement policies

Although there are various reasons for the cached copy of a data object to become invalid, such as data update at the server, expiration of some time-sensitive data, and so on, we only consider invalidity caused by client movements. An interesting aspect of location-based cache invalidation is that when the user has moved out of a semantic circle, it does not mean that the semantic circle has to be removed from the cache. This is because the user may re-enter the region covered by the semantic circle and thus the cached item is valid again.

However, with the limited cache memory in mobile clients, cache replacement is a major issue. When the cache is full, policies have to be employed to select a cache tuple for replacement. We observe that the size of a semantic circle may correlate to the utilization of the corresponding cached data. Thus, we develop new cache replacement policies which take the areas of the semantic circles into consideration. To the best of our knowledge, this is the first attempt that considers area as a major factor in the design of a cache replacement policy.

Furthest Away Replacement (*FAR*) is a well-known cache replacement policy for location dependent semantic cache [13]. It introduced the concept of *moving direction* as a factor for cache replacement. According to the client's velocity, all the data records in the cache can be divided into

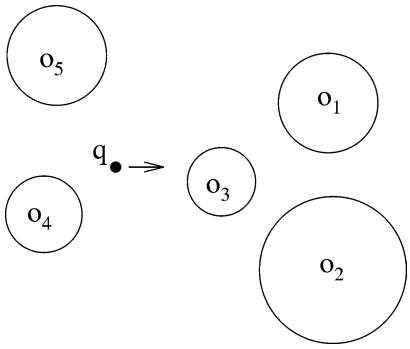


Figure 4. An example of *FAR*.

two sets. One contains all the objects that are in the direction of the client's movement, named "in" set. The other contains the rest of objects, named "out" set. The *FAR* strategy first replaces the victims from the "out" set according to their distances from the query client. Figure 4 depicts an example. Circles mean semantic circles stored in the client's cache, q means the query point, and the arched line means the moving direction of the client. Although o_4 is close to the client, it is in the reverse to the client's moving direction and is grouped into the "out" set with o_5. All the others are in the same direction of the client's movement and are grouped into "in" set. According to the distance metric, o_5 is the final victim since it is farther away from the client than o_4.

In this paper, we propose three new cache replacement policies, namely *Area*, *Dist*, and *AID*, and later compare them with *LRU* and *FAR*. In the following, *lost* refers to the replacement metric used to choose a victim.

Area. This policy considers the areas of the semantic circles. When the cache is full, the data object with the smallest semantic circle is replaced. The reasoning behind this policy is that the data object with a small valid area is less likely to be accessed than the other data objects. Therefore, its replacement is expected to have small impact on the cache hit ratio. Here, $lost_i = area_i$.

Dist. This policy takes *distance* between the center of a semantic circle and the client's current position into consideration. A data object with longest distance from the client is least likely to be accessed for *NN* search. Thus, it is the best candidate to be replaced. Here, $lost_i = 1/dist_i$.

Area-Inverse-Dist (AID). Based on both area and distance, this policy is a hybrid of *area* and *dist* policies. Here, $lost_i = area_i/dist_i$.

In the above policies, the cached data object that has the smallest *lost* score is to be replaced. With the assumption that all the data objects have the same size, we only consider one-by-one replacement.

5. Query scheduling for cross-cell roaming

Roaming is a very important property of mobile computing systems. For location-independent queries, the server may continue to process the query even when the client who issued the query has left the cell, because the answer obtained is still valid in the new cell and can be forwarded to the client there. For mobile *NN* search, however, the forwarded answer is usually invalid due to location change. Consequently, a client needs to resubmit the query to the new base station in order to obtain the correct answer. In this scenario, mobile clients who move around the borders of the cells or go across cell boundaries frequently may have to wait for a long time to finish a query (assuming that our clients are all persistent and finish a query only after the needed data is provided). In order to avoid the starvation problem, we propose the following query scheduling schemes.

Naive. This scheme is used to serve as the bottomline for performance comparison. The server answers the query based on *FCFS*. When a client needs some information, it submits a query to its base station and then waits for the answer. Once the client detects that it has arrived a new cell before obtaining the requested information, it resubmits the same query to the new base station. This procedure is repeated until its query is answered properly. This scheme is simple but the *handoff clients*[2] sometimes have to wait a long time for the requested data. Thus, starvation cannot be avoided.

Priority. In order to handle the starvation problems of the naive scheme, this scheme gives a higher priority to handoff clients. There are two query queues maintained in the base station, with one having a higher priority than the other. The queries submitted by handoff clients are put into the higher priority queue, and the queries issued by normal clients are put into the other queue. As long as the higher priority queue is not empty, the server will answer the query from this queue in *FCFS* order. With this scheduling scheme, starvation may happen for the normal clients when there are many handoff clients. Therefore, a control parameter, which equals to the probability of the server answering a query from the higher priority queue, can be used. Although handoff clients can obtain service quickly in a new cell, the tradeoff for this scheme is to prolong average response time of non-handoff clients.

Intelligent. This scheme gives priorities to clients who are very likely to leave the cell soon so that their queries can be answered before they handoff. When a client submits a query, the server can estimate its departure time based on its current position and velocity. If the time is shorter than the predefined threshold, this client is expected to handoff soon. Therefore, it is better for the server to service this client first. Two different priority queues are maintained in the server. Queries issued by clients expected to leave the cell are put into the high-priority queue, while other queries are put into the normal queue. When the high-priority queue is not empty, the server answers the query from this queue first with some predefined probability. This is the same as in the *Priority* method. Here, no priority has been given to the handoff clients. The effort is to reduce the number of handoff clients. The average query

[2] Handoff clients refer to those mobile clients who just enter new cells.

response time of the normal clients will be prolonged. Handoff clients do not receive any benefit.

Hybrid. This scheme combines *Priority* and *Intelligent* together by giving priorities to both handoff clients and clients to be handoff soon. The expense of this scheme is put upon the average query response time of normal clients.

The various schemes introduced above put different emphasis on different aspects of the handoff problem. Therefore, each one has its own pros and cons, as the performance evaluation presented in the next section shows.

6. Simulation model

We conduct simulations to examine the performance of our proposals. In this section, we describe a simulation model for performance evaluation. *CSIM* [14], a process-oriented, discrete-event simulation package, is used for implementation of our simulations.

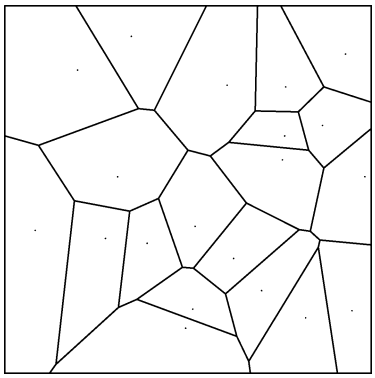
6.1. System model

For the sake of generality, we simulate a multi-cell environment even though some of the experiments can be performed in a single-cell environment. The system has nine cells organized as a 3×3 grid.[3] Each cell is represented by a square with a side length of *SideLen*. The clients' movements follow a "wrapped-around" moving pattern, i.e., when a client leaves the square from an edge, it will enter the square from the opposite edge with the same velocity. In each cell, the average number of clients is denoted as *ClientNum*.

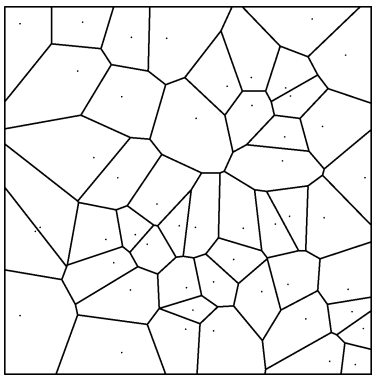
The data server maintains *ServNum* types of services (e.g., restaurants, theaters, and gas stations). Each type of services has different *AreaNum* values. Therefore, *ServNum* $\times$ *AreaNum* approximates the whole database size. To simplify the simulation, we assume that the distributions of data values for all the service types are the same. A wireless cell (i.e., the geographical region monitored by a base station) is evenly divided into *AreaNum* parts. Then, a service facility is randomly produced in each part. Given n locations corresponding to the service facility of the same type, a Voronoi Diagram is constructed using the *Triangle* algorithm in $O(n \log n)$ time [15]. Figure 5 shows two Voronoi Diagrams produced in our simulation that serve as area distributions in later experiments. All this work is done in the data preprocessing stage of a location-based service.

We assume point-to-point communication between the server and clients. Thus, an uplink channel and a downlink channel, with bandwidth *UplinkBand* and *DownlinkBand*, respectively, are created between a client and the server. Table 1 summarizes the configuration parameters of the system model.

[3] We find that the size of grid does not influence the simulation result. A small 3×3 grid is chosen, because a square cell has 8 neighbors.



(a) Area distribution 1: *AreaNum* = 20.



(b) Area distribution 2: *AreaNum* = 50.

Figure 5. Two VDs for performance evaluation.

Table 1
Configuration parameters of the system model.

Parameter	Description
SideLen	Side length of the square service area
ServNum	Number of service types
AreaNum	Number of different facility instances for each service type
DataSize	Size of a data value
UplinkBand	Bandwidth of the uplink channel
DownlinkBand	Bandwidth of the downlink channel
ClientNum	Average number of clients within a cell

6.2. Client model

Each client is modeled by an independent process. It continues to issue queries for *NN* search over various service types. A Zipf distribution, with preset θ value to control skewness, is used to model user data access. Before a query is submitted, a client first checks its local cache for the availability of the requested data. Only when a cache miss happens would the client submit the query to the server; otherwise, the query is answered locally. The think time between two successive queries issued by a client is assumed to be exponentially distributed with a mean value of *MeanThinkTime*. Here, we assume that the client can detect the current base station. When it finds a new base station and it has an unfinished query, this client will resubmit the same query to the new base station.

The client is assumed to have a cache of *CacheSize* size, which is a *CacheSizeRatio* ratio of the database size. We ignore the overhead of the semantic description since it is very

small compared to the size of a data item used in the simulation. A detailed analysis of the relationship between cache usage and the embedded semantic information is available in [21]. Finally, to simplify our evaluation, we assume that the velocity of a mobile user does not change, since this problem is out of scope of this study. In our experiments, we specify a range of clients' speed with *MaxSpeed* and *MinSpeed*. The direction of motion is chosen randomly.

6.3. Server Model

The server is modeled by a single process that serves requests from clients. In the first part of the simulation, we compare the performance of traditional cache and semantic cache, along with various cache replacement policies. In this part, only one infinite queue is maintained in the server. The server answers the queries in *FCFS* order. To answer an *NN* search for certain service type, the server first employs the trapezoidal mapping algorithm to locate the client in a Voronoi cell. Then, the service facility located in that Voronoi cell is returned as the answer. Generally speaking, the algorithm can locate the Voronoi cell of the client in $O(\log m)$, where m is the total number of the edges in this Voronoi Diagram. We assume that the system is heavily loaded and major congestion occurs in wireless transmission. Thus, the query processing time is omitted. Given the size of a data object *DataSize*, the transmission time *ServTime* equals to $(DataSize + Overhead)/DownlinkBand$. *Overhead* here refers to the spatial scope information of semantic circles. In this simulation, each service type is treated independently and the clients can only query one type of services at a time.

In the second part of simulation, various query scheduling schemes are evaluated. Two infinite queues are maintained in the server; one has a higher priority than the other.[4] *UrgReq* is defined as the time threshold to determine whether a client is to handoff soon or not. *HigherFirstProb* is the probability that the server will answer the query from the higher priority queue first when it is not empty. Table 2 summarizes the configuration parameters of the client and server models.

7. Performance evaluation

Based on the simulation model described earlier, experiments have been conducted to evaluate the access performance of mobile *NN* search. We first compare the performance of systems with semantic cache, traditional cache, and no cache. Then, we compare several cache replacement policies designed for the semantic cache used in our study. Finally, the impacts of various query scheduling schemes, which address the roaming problem, are examined. Table 3 shows the parameter settings for our experiments. All the experiments use these settings unless noted explicitly.

[4] For naive scheduling scheme, only one queue is used.

Table 2
Configuration parameters of the client and server models.

Parameter	Description
MeanThinkTime	Average time interval between two consecutive queries issued by a client
MinSpeed	Minimum moving speed of a client
MaxSpeed	Maximum moving speed of a client
CacheSizeRatio	Ratio of the cache size to the database size
CacheSize	Cache size of the client
θ	Skewness parameter for the *Zipf* access distribution
UrgReq	The time threshold used to detect the clients that will be handoff soon
HigherFirstProb	The probability that a server answers the query from a higher priority queue first

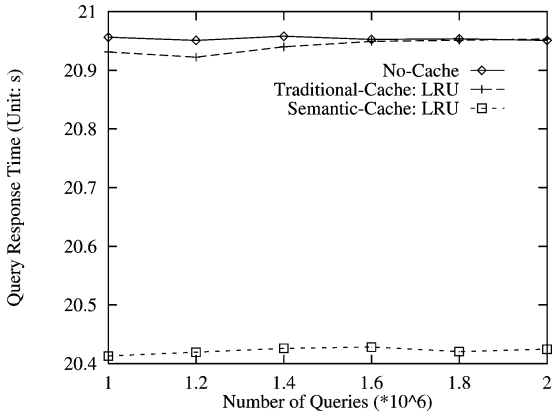
Table 3
Parameter settings for performance evaluation.

Parameter	Setting
SideLen	1000 meters
ServNum	20
AreaNum	20, 50
DataSize	512 byte
DownlinkBand	$1.25 \cdot 10^5$ byte/second
ClientNum	1000
MeanThinkTime	20.0 seconds
MinSpeed	10 meters/second
MaxSpeed	20 meters/second
CacheSizeRatio	10%
UrgReq	$100 \cdot$ *ServTime* seconds
HigherFirstProb	70%

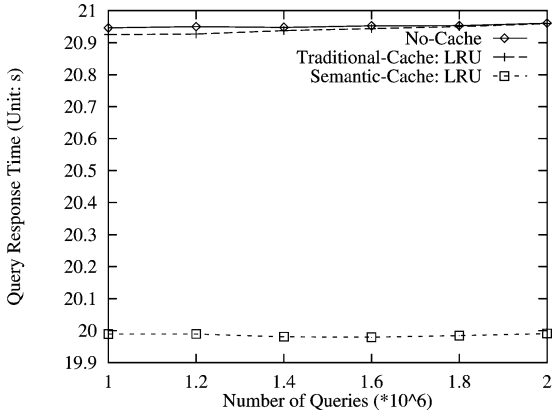
7.1. Experiment #1: Caching techniques

In this section, we compare the average query response time of mobile clients with (1) no cache, (2) traditional cache, and (3) semantic cache. For a mobile client without cache, it has to submit its query to the server and waits for the response. For a mobile client with a traditional cache, it issues a query to the server if the answer is not available in cache. The returned answer is cached for future use. Since we only consider mobile *NN* search in this study, the cached answers can only be reused when the client submits the same query at the same location. For a mobile client with a semantic cache, it issues a query to the server just like a client with a traditional cache, if the answer is not available in cache. The returned data object, along with a semantic circle, is cached to answer the same query in the future as long as the client is located within the semantic circle. To provide a fair comparison between traditional caching and semantic caching techniques, we choose LRU as the cache replacement policy for this experiment.

Figures 6 and 7 show the simulation result in terms of average query response time and cache hit ratio, respectively. By increasing the simulation time, represented by the number of queries processed, we observe that the response time reaches a steady state. The simulation result shows that the traditional data cache does not improve the performance much over no-cache approach. This is due to the location dependency of mobile *NN* search, i.e., cached data can only be reused when the client submits the same query at the same location. On the other hand, cache hit ratio of traditional cache is close to zero,

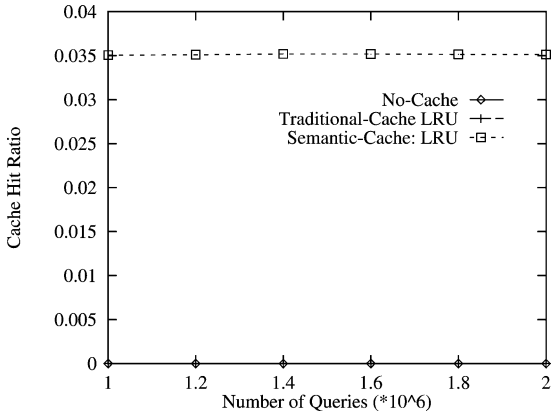


(a) Area distribution 1: *AreaNum* = 20.

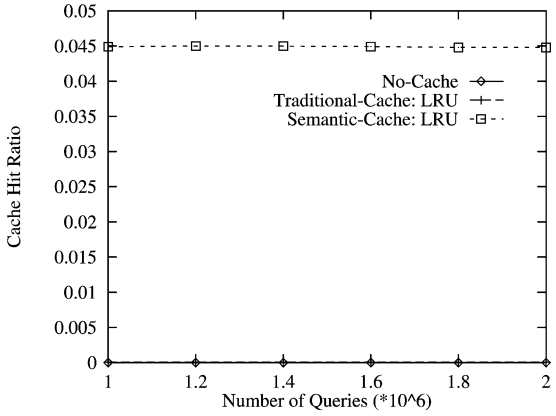


(b) Area distribution 2: *AreaNum* = 50.

Figure 6. Average query response time vs. caching techniques.



(a) Area distribution 1: *AreaNum* = 20.



(b) Area distribution 2: *AreaNum* = 50.

Figure 7. Average cache hit ratio vs. caching techniques.

which shows the ineffectiveness of the traditional caching technique on location-dependent queries. The semantic caching technique simply outperforms the other two methods. In terms of query response time, it improves about 2.61% over traditional cache for the first distribution and 4.84% for the second distribution, respectively. In terms of cache hit ratio, the superiority of semantic cache is overwhelming.

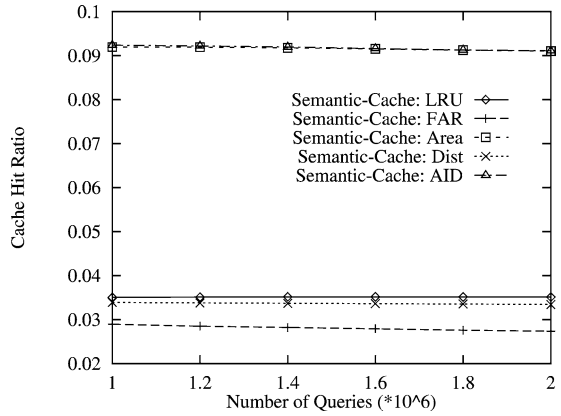
7.2. Experiment #2: Cache replacement policies

This experiment is to compare the impact of different cache replacement policies on semantic cache. A good cache replacement policy will maintain a high cache hit ratio and thus improve the system performance. In this experiment, we examine the semantic caching technique with *Area*, *Dist*, *AID*, *FAR*, and *LRU* cache replacement policies.

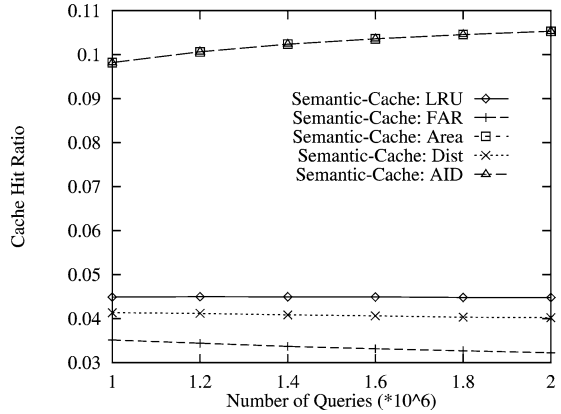
As shown in figure 8, cache replacement policies do have a significant impact on cache performance. We found that *Area* and *AID* outperform others significantly. From the experiments, it is observed that area is a major factor that affects the cache hit ratio for location-dependent queries. While *AID* performs better than *Area*, the improvement is not distinguishable, only about 0.19% for distribution 1 and nearly zero for distribution 2. Surprisingly, the *LRU* performs slightly better than the *Dist* and *FAR*. One reason for this is that, in this experiment, we do not consider the impact caused by the clients' moving patterns and assume the clients move in a constant velocity. *FAR* actually outperforms *LRU* when the queries issued have high locality. Our later study shows the impact of clients' movement on the cache replacement strategies in detail [21].

Another observation is that the performance of semantic cache improves gradually with the increase of total query numbers, which is particularly obvious in distribution 2. This is also impacted by the area factor. *Area* and *AID* take area as the major replacement metric. As a result, a data object with small area is more likely to be replaced than a data object with large area. Consequently, the total area covered by all the cached semantic circles improves along with the increase of simulation time. Cache hit ratio, which is expected to be equal to the proportion between cached area over the area of the Voronoi cell, becomes higher and the query response time is reduced. However, the cache hit ratio will not continue to increase, because the maximal semantic circle of an area is less than or equal to its inscribe circle.

In addition to the above experiment, we evaluate the cache replacement policies by varying the ratio of cache size to database size, i.e., *CacheSizeRatio*, from 5% to 20%. The objective of this experiment is to examine the impact of cache size on hit ratio of cache replacement policies. As illustrated in figure 9, while the cache size increases, hit ratio is improved as we expected.

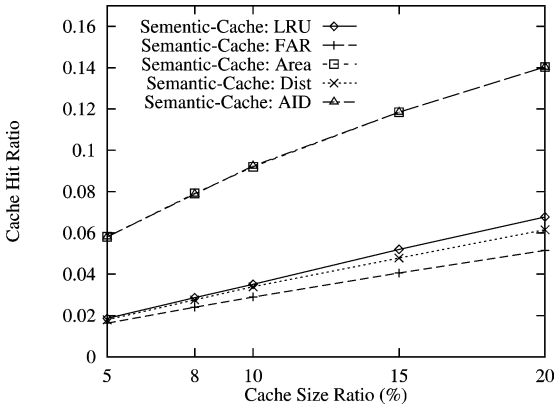


(a) Area distribution 1: $AreaNum = 20$.

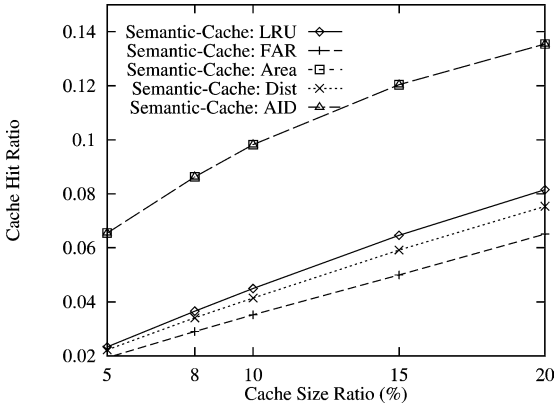


(b) Area distribution 2: $AreaNum = 50$.

Figure 8. Cache hit ratio vs. cache replacement policies.



(a) Area distribution 1: $AreaNum = 20$.



(b) Area distribution 2: $AreaNum = 50$.

Figure 9. Cache hit ratio vs. cache sizes.

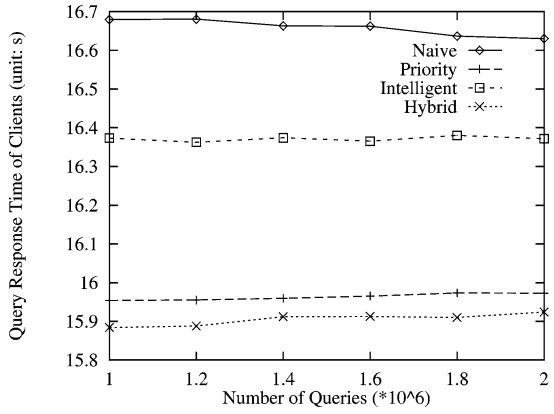
7.3. Experiment #3: Query scheduling schemes

The main objective of this experiment is to evaluate the performance of different query scheduling schemes which addresses the query resubmission problem caused by roaming. In addition to the performance metrics used in previous experiments, we introduce two more performance metrics: *cross number*, denoting the number of handoff clients, and *recross number*, denoting the number of clients that have been handoff several times before receiving answers to their queries. Each client has a local semantic cache employing *Area* as the cache replacement scheme since it provides better performance in most cases as illustrated in our previous experiments. Default values of the parameters related to clients' cache are used. In this experiment, we keep track of the average query response time of normal clients who have not been given any priority but received the requested data in the cell where they issue the queries.

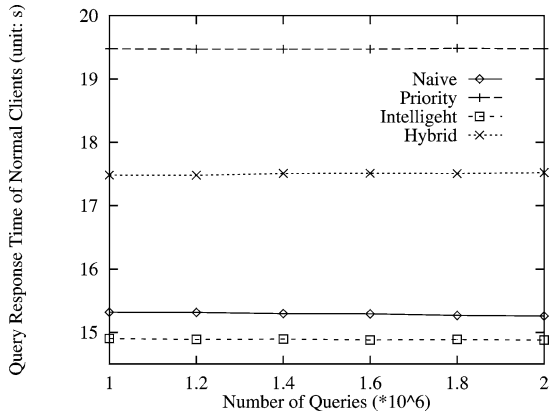
Figure 10 shows the mobile clients' query response time corresponding to various query scheduling schemes. Only the result based on distribution 1 is shown here. The other distribution produces a similar result so we omitted it to save space.

As illustrated in figure 10(a), *hybrid* has the best performance in query response time of all mobile clients, which includes both the handoff clients and normal clients. As far as query response time is concerned, *hybrid* outperforms *naive* by about 4.74%. Compared with *naive*, the improvement is about 4.36% and 1.76% for *priority* and *intelligent*, respectively. Figure 10(b) shows the average response time of the queries issued by the normal clients, which are not given any priority. Since *intelligent* detects the clients who will be handoff soon and tries to answer requests before they leave the cell, those clients receive responses quickly. These clients are also considered as normal clients, consequently, the average query response time of normal clients of *intelligent* is shortened compared with *naive*. For the other two scheduling schemes, i.e., *priority* and *hybrid*, the query response time for normal clients is increased.

Figures 11 and 12 show the experiment results in terms of *cross* and *recross* numbers. These two metrics gauge the severity of query resubmission due to roaming. Therefore, a goal of scheduling schemes is to reduce both the cross and recross numbers so as to maximize the number of mobile clients which will receive their responses before they leave their current cells. As illustrated, *priority* has a higher *cross* number than *naive*, because it provides service to handoff clients first. Therefore, the query response time of normal clients is inevitably increased and the *cross* number is increased. However, this scheme does reduce the *recross* number. As shown in figure 12, its *recross* number is nearly zero. In terms of *recross* number, all the proposed schemes outperform *naive* and the improvement is very significant.



(a) Query response time of all clients.



(b) Query response time of normal clients.

Figure 10. Query response time vs. query scheduling schemes.

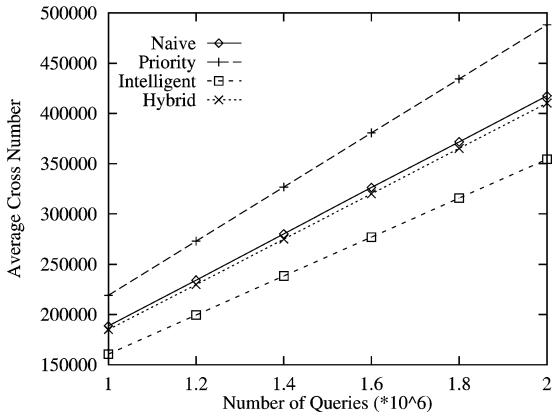


Figure 11. Average cross number.

In the experiments shown above, all the parameters use default values. Actually, some parameters may impact the final performance. As for the *priority* scheduling scheme, *HigherFirstProb* decides the probability that a higher priority is assigned to a handoff client. With the increase of *HigherFirstProb*, from 0.3 to 1.0, only *recross* decreases monotonously. The average response time reaches the optimal value when *HigherFirstProb* is set around 0.9, while the response time of normal clients reaches the peak value when *HigherFirstProb* is near 0.8. Due to space constraint, the details are omitted in this paper.

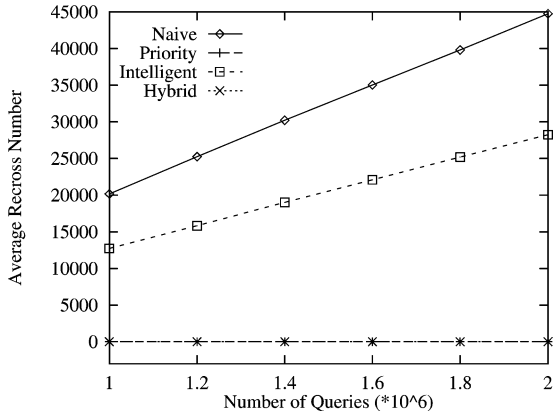


Figure 12. Average recross number.

8. Conclusion

Mobile nearest-neighbor (*NN*) search is a basic but important function for LBSs. A Voronoi Diagram-based index has been introduced in this paper to answer queries for finding the nearest service facilities based on mobile clients' locations. Due to the limited resources in mobile devices and realtime requirement of location-based services, we propose a semantic cache to address the access efficiency of mobile *NN* search. Several cache replacement policies, tailored for the proposed semantic cache, are proposed and examined. Moreover, we extend our study from a single-cell environment to a multi-cell environment. Considering the fact that mobile clients may roam across cells, it's likely for a mobile client to leave a cell without receiving the response for the query it issued there. This requires the resubmission of the query in the new cell and thus results in performance degradation. In order to reduce the number of query resubmissions, four different query scheduling schemes have been proposed.

A series of experiments have been conducted to evaluate the performance of our proposals in this paper. The result shows that the VD-based index is an efficient solution for *NN* search in mobile environments. Enhanced with semantic cache, query processing efficiency is greatly improved. We compared several cache replacement policies for semantic cache and found that the areas of the semantic circles associated with the cached data are the major factor that impacts cache performance in terms of query response time and hit ratio. By taking roaming issues into consideration, our experiments compare four different query scheduling schemes which aim at reducing the cell-cross number and cell-recross number. It is shown that no single scheduling scheme outperforms all the others in all aspects. The appropriate scheduling scheme must be chosen based on the application's requirements.

This study represents the first step into a very important field of location-dependent query processing. In this paper, we have explored some of the research issues in the field, but further studies are needed. Two directions will be pursued in our follow-up studies. First, location dependent queries other than the basic mobile *NN* search will be examined. Secondly, different communication mechanisms between the base sta-

tion and mobile clients will be studied. While the point-to-point communication is employed in this paper, we will investigate corresponding problems in broadcast-based systems.

Acknowledgements

We thank the special issue editors and the anonymous reviewers for their valuable comments and suggestions, which have improved the quality of this article. The Research Grant Council, Hong Kong SAR, China supported this research under grant number HKUST6079/01E. Baihua Zheng was supported by Wharton-SMU Research Center, Singapore Management University. Wang-Chien Lee was supported in part by US National Science Foundation grant IIS-0328881.

References

[1] S. Acharya, R. Alonso, M. Franklin and S. Zdonik, Broadcast disks: Data management for asymmetric communications environments, in: *Proceedings of ACM SIGMOD Conference on Management of Data*, San Jose, CA (May 1995) pp. 199–210.

[2] A. Bakre and B.R. Badrinath, Handoff and systems support for indirect TCP/IP, in: *Proceedings of 2nd Usenix Symposium on Mobile and Location-Independent Computing* (April 1995).

[3] D. Barbara, Mobile computing and databases – a survey, IEEE Transactions on Knowledge and Data Engineering 11(1) (1999) 108–117.

[4] D. Barbara and T. Imielinski, Sleepers and workaholics: Caching strategies for mobile environments, in: *Proceedings of ACM SIGMOD Conference on Management of Data*, Minneapolis, MN (May 1994) pp. 1–12.

[5] S. Berchtold, B. Ertl, D.A. Keim, H.P. Kriegel and T. Seidl, Fast nearest neighbor search in high-dimensional space, in: *Proceedings of the 14th International Conference on Data Engineering (ICDE'98)* (February 1998) pp. 209–218.

[6] S. Berchtold, D.A. Keim, H.P. Kriegel and T. Seidl, Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space, IEEE Transactions on Knowledge and Data Engineering 12(1) (2000) 45–57.

[7] M. Berg, M. Kreveld, M. Overmars and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Chapter 7 (Springer, New York, NY, 1996).

[8] T. Camp, J.C. Lusth and J. Matocha, Reduced cell switching in a mobile computing environment, in: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)* (August 2000) pp. 143–154.

[9] H.G. Gök, Processing of continuous queries from moving objects in mobile computing systems, Master's Thesis, Bilkent University (1999).

[10] H.G. Gök and Ö. Ulusoy, Transmission of continuous query results in mobile computing systems, Information Sciences 125(1–4) (2000) 37–63.

[11] D.L. Lee, W.-C. Lee, J. Xu and B. Zheng, Data management in location-dependent information services, IEEE Pervasive Computing 1(3) (2002) 65–72.

[12] Q. Ren and M.H. Dunham, Semantic caching and query processing, Technical Report 98-CSE-04, Southern Methodist University (May 1998).

[13] Q. Ren and M.H. Dunham, Using semantic caching to manage location dependent data in mobile computing, in: *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'2000)*, Boston, MA (August 2000) pp. 210–221.

[14] H. Schwetman, CSIM user's guide (version 18), Mesquite Software, Inc. (1998), `http://www.mesquite.com`

[15] J.R. Shewchuk, Triangle: Engineering a 2*d* quality mesh generator and delaunay triangulator, in: *Proceedings of the 1st Workshop on Applied Computational Geometry* (May 1996) pp. 124–133.

[16] A.P. Sistla, O. Wolfson, S. Chamberlain and S. Dao, Modeling and querying moving objects, in: *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*, Birmingham, UK (April 1997) pp. 422–432.

[17] P. Sistla, O. Wolfson, S. Chamberlain and S. Dao, *Querying the Uncertain Position of Moving Objects* (Springer, Berlin, 1998) pp. 310–337.

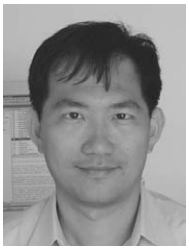
[18] S. Tekinay and B. Jabbari, Handover and channel assignment in mobile cellular networks, IEEE Communications Magazine (November 1991) 42–46.

[19] J. Xu, X. Tang, D.L. Lee and Q.L. Hu, Cache coherency in location-dependent information services for mobile environments, in: *Proceedings of the 1st International Conference on Mobile Data Access (MDA'99)*, Hong Kong (December 1999) pp. 182–193.

[20] J. Xu, B. Zheng, W.-C. Lee and D.L. Lee, Energy efficient index for querying location-dependent data in mobile broadcast environments, in: *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE'03)*, Bangalore, India (March 2003).

[21] B. Zheng, J. Xu and D.L. Lee, Cache invalidation and replacement strategies for location-dependent data in mobile environments, IEEE Transactions on Computers, Special Issue on Database Management and Mobile Computing 51(10) (2002) 1141–1153.

Baihua Zheng received a Ph.D. in computer science from Hong Kong University of Science and Technology. She is a member of the IEEE and the ACM. Currently, she is an Assistant Professor in the School of Information Systems at Singapore Management University. Her research interests include mobile and pervasive computing, and spatial databases.
E-mail: bhzheng@smu.edu.sg



Wang-Chien Lee received a Ph.D. in computer and information science from the Ohio State University in 1996. He is a member of the IEEE and the ACM. Currently, he is an Associate Professor in the Computer Science and Engineering Department at Pennsylvania State University. His research interests include mobile and pervasive computing, data management, and Internet technologies.
E-mail: wlee@cse.psu.edu

Dik Lun Lee received the M.S. and Ph.D. degrees in computer science from the University of Toronto in 1981 and 1985, respectively. He is a Professor in the Department of Computer Science at the Hong Kong University of Science and Technology, and was an Associate Professor in the Department of Computer and Information Science at the Ohio State University, Columbus, OH. He has served as a guest editor for several special issues on database-related topics, and as a program committee member and chair for numerous international conferences. He was the founding conference chair for the International Conference on Mobile Data Management. His research interests include document retrieval and management, discovery, management and integration of information resources on Internet, and mobile and pervasive computing. He was the Chairman of the ACM Hong Kong Chapter.
E-mail: dlee@cs.ust.hk