A Proxy-Based Approach to Continuous Location-Based Spatial Queries in Mobile Environments

Jiun-Long Huang, *Member*, *IEEE*, and Chen-Che Huang

Abstract—Caching valid regions of spatial queries at mobile clients is effective in reducing the number of queries submitted by mobile clients and query load on the server. However, mobile clients suffer from longer waiting time for the server to compute valid regions. We propose in this paper a proxy-based approach to continuous nearest-neighbor (NN) and window queries. The proxy creates estimated valid regions (EVRs) for mobile clients by exploiting spatial and temporal locality of spatial queries. For NN queries, we devise two new algorithms to accelerate EVR growth, leading the proxy to build effective EVRs even when the cache size is small. On the other hand, we propose to represent the EVRs of window queries in the form of vectors, called estimated window vectors (EWVs), to achieve larger estimated valid regions. This novel representation and the associated creation algorithm result in more effective EVRs of window queries. In addition, due to the distinct characteristics, we use separate index structures, namely EVR-tree and grid index, for NN queries and window queries, respectively. To further increase efficiency, we develop algorithms to exploit the results of NN queries to aid grid index growth, benefiting EWV creation of window queries. Similarly, the grid index is utilized to support NN query answering and EVR updating. We conduct several experiments for performance evaluation. The experimental results show that the proposed approach significantly outperforms the existing proxy-based approaches.

Index Terms—Nearest neighbor query, window query, spatial query processing, location-based service, mobile computing

1 Introduction

Location-based services (LBSs), also known as location-dependent information services (LDISs), have been recognized as an important context-aware application in pervasive computing environments [1]. Spatial queries are one of the most important LBSs. According to spatial constraints, spatial queries can be divided into several categories including nearest neighbor (NN) queries and window queries. An NN query is to find the nearest data object with respect to the location at which the query is issued (referred to as the *query location* of the NN query). For example, a user may launch an NN query like "show the nearest coffee shop with respect to my current location." On the other hand, a window query is to find all the objects within a specific window frame. An example window query is "show all restaurants in my car navigation window."

In general, a mobile client continuously launches spatial queries until the client obtains a satisfactory answer. For example, a query "show me the rate of the nearest hotel with respect to my current location" is continuously submitted in a moving car so as to find a desired hotel. The naive method answering continuous spatial queries is to submit a new query whenever the query location changes. The naive method is able to provide correct results, but it poses the following problems:

- *High power consumption.* The power consumption of a mobile device is high since the mobile device keeps submitting queries to the LBS server.
- *Heavy server load.* A continuous query usually consists of a number of queries to the LBS server, thereby increasing the load on the LBS server.

Fortunately, in the real world, the queries of a continuous query usually exhibit spatial locality. Thus, caching the query result and the corresponding valid region (VR) in the client-side cache was proposed in [2], [3] to mitigate the above problems. The *valid region*, also known as the valid scope, of a query is the region where the answer of the query remains valid. Subsequent queries can be avoided as long as the client is in the valid region. Note that a VR is associated with a query by definition. However, as pointed out in [4], by associating a VR with the corresponding object, the VR of an object p can be used to resolve all the queries whose answer object is p.

Although VRs are useful, an LBS server may not provide VRs in practice since the server may simply provide query results only or would not compute VRs under heavy load. In these situations, mobile service providers (e.g., Verizon Wireless and AT&T) or smartphone makers (e.g., Apple and RIM) can utilize a proxy architecture where the proxy provides *estimated valid regions* (EVRs), which are the subregions of the corresponding VRs, for the clients. With the proxy architecture, the clients still can enjoy the benefits of EVRs even if the LBS server does not provide VRs and thus the mobile service providers and smartphone makers can attract more clients. In other words, the proxy-based approach is an alternative solution to serve the function of

- *The authors are with the Department of Computer Science, National Chiao Tung University, No. 1001, University Road, Hsinchu City 30010, Taiwan, R.O.C. E-mail: jlhuang@cs.nctu.edu.tw, cchuang.cs95g@nctu.edu.tw.*

Manuscript received 22 Nov. 2010; revised 4 Aug. 2011; accepted 28 Aug. 2011; published online 23 Sept. 2011.
Recommended for acceptance by T. Sellis.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-11-0621.
Digital Object Identifier no. 10.1109/TKDE.2011.203.

VRs in case that the LBS server could not provide VRs. In [5], [6], the authors proposed to deploy a proxy between mobile clients and the LBS server to build EVRs by exploiting the queries interested in the same objects. EVR provisioning is inspired by that, in addition to spatial locality, spatial queries also exhibit temporal locality, resulting from that a number of queries with close query locations are likely to be launched during a short interval. For instance, a large number of NN queries about nearest hotels will be launched by passengers after a train arrives. Hence, a proxy may be able to answer subsequent queries interested in the same objects by caching EVRs created based on previous queries. Despite the success of proxy deployment and EVR provisioning in [5], [6], they have the following drawbacks:

- *Slow growth of EVRs of NN queries.* The algorithms proposed in [5] suffer from slow EVR growth, which causes a lower cache hit ratio. The effect of an EVR in [5] is also limited by the number and locations of the queries interested in the same data object.
- *Slow growth of EVRs of window queries.* The algorithm proposed in [6] has the same drawbacks as [5]. Besides, since VRs of window queries are likely to be concave polygons, the EVRs are built in a pessimistic manner, causing the EVRs to be extremely small.
- *Lack of mutual support.* The algorithms of [5] and [6] are executed independently. That is, the result of an NN query is useless for [6] to create the EVR of a window query, and vise versa.

In view of this, we propose in this paper a proxy architecture as well as several companion algorithms to provide EVRs of NN and window queries on static data objects for mobile clients. We aim to reduce the number of queries submitted by mobile clients, the time of obtaining query results and corresponding EVRs, and load on the LBS server. The contributions of this paper are summarized as follows:

- For NN queries, we devise new algorithms to efficiently create new and extend existing EVRs. The devised algorithms not only enable mobile clients to obtain effective EVRs immediately but also lead the proxy to build effective EVRs even when the proxy cache size is small.
- For window queries, different from [6], we propose to index the positions of data objects, instead of EVRs, by a grid index. Since VRs of window queries may not be convex polygons, creating effective polygonal EVRs by previous queries is inherently difficult. Thus, we propose to represent the EVRs of window queries in the form of vectors, called *estimated window vectors* (EWVs), to achieve larger estimated valid regions. Such novel representation and indexing lead the proxy to efficiently create more effective EVRs of window queries.
- We introduce two index structures, an EVR-tree for NN queries and a grid index for window queries, due to the distinct characteristics of NN and window queries. We also develop algorithms to make these two index structures mutually support each other. Specifically, the grid index can be used to answer NN queries and extend existing EVRs. The answer objects of NN queries are exploited to update the grid index, benefiting the creation of more effective EWVs of window queries.
- We conduct several experiments to compare the proposed approach with the existing proxy-based approaches [5], [6] and the representative server-based approach [7], [8]. The experimental results show that the proposed approach significantly outperforms the existing proxy-based approaches. Moreover, we compare the proposed approach with the representative server-based approach [7], [8] for understanding the benefits of the proxy architecture. The experimental results demonstrate that the performance of the proposed approach is close to that of [7], [8] even though the proposed approach has only partial information of data objects.

The rest of this paper is organized as follows: Section 2 reviews related work and presents the preliminaries. We elaborate NN and window query processing in Sections 3 and 4, respectively. The experimental results are shown in Section 5 to evaluate the performance of the proposed approach. In Section 6, we provide a comparison between proxy-based and server-based approaches. Finally, Section 7 concludes this paper and discusses future work.

2 Preliminaries

2.1 Related Work

In recent years, a significant number of research studies have been proposed for spatial query processing. Most of these studies addressed representative spatial queries, such as NN queries, kNN queries, and window queries. For different scenarios, some studies coped with static data objects while some tackled mobile data objects. The latter studies [9], [10], [11], [12] coped with continuous window query monitoring while [12], [13], [14], [15], [16] addressed continuous kNN query monitoring. Since our work falls into the former category, we focus on reviewing previous studies on static data objects in the following.

R-tree and its variants, such as R^*-tree [17], are one of the most popular methods of static spatial query processing. An LBS server is able to answer spatial queries quickly using R-tree-like index structures. However, in mobile environments, mobile clients usually launch a continuous query consisting of a number of queries with different query locations for obtaining a satisfactory answer. Continuous queries cause the conventional scheme to suffer from wireless medium contention and heavy query load. To address this problem, previous studies proposed that mobile clients could avoid launching unnecessary queries by caching VRs and EVRs. According to the architecture, these studies can be classified into two categories: server-based approaches and proxy-based approaches. Basically, server-based approaches have the complete information of data objects and can utilize the information to create VRs for mobile clients. On the other hand, proxy-based approaches have only partial information of objects and exploit spatial and temporal locality of queries of mobile clients to build EVRs. We first introduce the existing server-based approaches as follows: In [2], Zheng et al. proposed that an LBS server creates the Voronoi diagram [18]

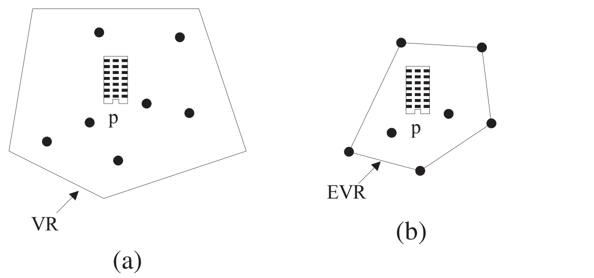


Fig. 1. Example VR and EVR. (a) An example VR. (b) An example EVR.

of data objects in an offline manner and returns the answer object of an NN query as well as the corresponding Voronoi cell to the querying client. The client caches the Voronoi cell to reduce the number of subsequent queries since the Voronoi cell is the VR of the answer object. The main advantage is that the LBS server constructs the Voronoi diagram once and uses it repeatedly. However, searching the corresponding Voronoi cell is time consuming and object updates incur the overhead of partial reconstruction of the Voronoi diagram. Zheng et al. [19] introduced a grid-partition-index to improve search efficiency, but it is still impaired by object updates.

Zhang et al. [20] proposed to compute the VRs by executing a number of time parameterized queries [21] in an online manner that avoids object update cost. The drawback is that, when the number of queries increases, the average waiting time of clients becomes longer since VR computation requires extra I/O and computational cost. Thus, this approach is not suitable for large-scale LBSs. In [7], [8], Lee et al. proposed to retrieve the nearest object and to exploit the objects in the remaining queue to identify the introduced complementary objects. Complementary objects are those objects contributing the bisector as the Voronoi cell perimeter. The VR computation algorithm of [7], [8] only requires one single index lookup. Nutanong et al. [22] presented a technique called V^*-Diagram, which computes VRs based on the data objects, the query location, and the current knowledge of the search space. V^*-Diagram exploits the FRR (fixed-rank region proposed in [23]) of $(k+x)$ maintained objects and the safe region of k nearest objects to create the VR of the kNN query. Both [7], [8] and [22] outperform [20] significantly. As to window queries, in [24], Dar et al. allowed a client to store the answer of a window query as a semantic region. Any window query covered by the semantic region could be directly resolved by the client's local cache. Zhang et al. [20] also introduced an online algorithm to compute the VRs of window queries via a number of time parameterized queries, and later, Lee et al. proposed to identify the complementary objects of a window query by enlarging the query window. If all answer objects are inside the query window and no complementary object is encountered, the mobile client can know that the cached answer objects remain valid. In [25], Ku et al. presented a peer-sharing paradigm, allowing mobile clients to obtain a complete or partial result of the query from other clients in the vicinity. A mobile client only waits for unresolved parts from the LBS server and thus the average waiting time could be reduced.

These server-based approaches suffer from heavy load on the LBS server and relatively longer waiting time for mobile clients. In order to alleviate the problems, Gao and Hurson [5] presented a proxy-based approach which deploys a proxy between an LBS server and mobile clients. Based on spatial and temporal locality of spatial queries mentioned earlier, they proposed to make the proxy create EVRs according to query history. For example, in Fig. 1a, each point represents a previous NN query whose answer is object p. These query points are within the VR of p. Since VR computation cost is high, the proxy can build the EVR of p by connecting these query points. Because each EVR is a subregion of the corresponding VR, caching EVRs helps the proxy resolve some subsequent queries directly. Following [5], Gao et al. [6] employed the proxy to create the EVRs of window queries based on window query history. All query points with the same query result are used to create the corresponding EVR. Since the VR of a window query may not be a convex polygon, the proxy pessimistically assumes that these query points occur at the boundary of the VR, and creates a feasible but small EVR. Besides, EVR construction cannot be applied to the case where a window query has no answer objects. Moreover, the window queries with the same result set but of different sizes cannot be used to create an EVR.

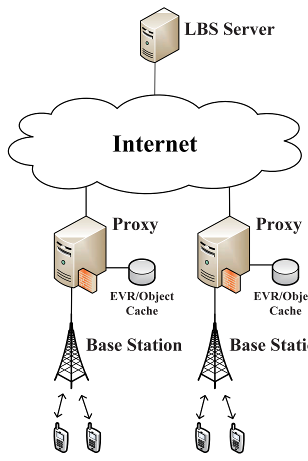


Fig. 2. System architecture.

2.2 System Architecture

Fig. 2 depicts the proposed system architecture for NN and window query processing. The system architecture consists of three parts: 1) an external LBS server, 2) deployed proxies, and 3) the mobile clients. The LBS server is responsible for managing static data objects and answering the queries submitted by the proxies. Note that the LBS server can use any index structure (e.g., R-tree or grid index) to process spatial queries. The LBS server is assumed not to provide VRs. Each of the deployed proxies supervises one service area and provides EVRs of NN queries and EWVs (vector form of EVRs) of window queries for mobile clients in the service area. Each base station serves as an intermediate relay for queries and query results between mobile clients and the associated proxy. Base stations, proxies, and the LBS server are connected by a wired network. A mobile client maintains a cache to store the query results and the corresponding EVRs. When a mobile

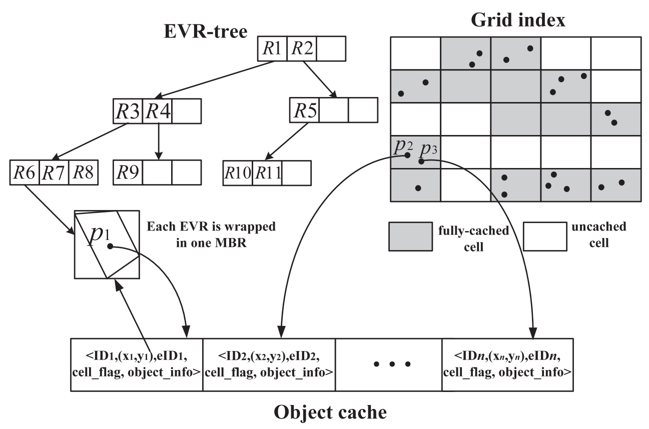

EVR-tree
R1 R2
R3 R4
R5
R6 R7 R8
R9
R10 R11
p1
Each EVR is wrapped in one MBR
Grid index
p2 p3
fully-cached cell
uncached cell
<ID1,(x1,y1),eID1, cell_flag, object_info>
<ID2,(x2,y2),eID2, cell_flag, object_info>
• • •
<IDn,(xn,yn),eIDn, cell_flag, object_info>
Object cache

Fig. 3. Object cache, EVR-tree, and grid index.

client has a spatial query, the mobile device first examines whether the current location is in the EVR of the stored result. If so, the stored result remains valid and the mobile device directly shows it to the client. Otherwise, the mobile device submits the query, which is received and then forwarded by the base station, to the proxy. For the received query, the proxy will return the query result as well as the corresponding EVR to the client.

2.3 Proxy Design

A proxy builds EVRs of NN queries and EWVs of window queries based on NN query history and available data objects, respectively. The proxy maintains an object cache and two index structures: an EVR-tree for NN queries and a grid index for window queries, as illustrated in Fig. 3. The two index structures share the data objects in the object cache. The EVR-tree is an R-tree (or its variants) composed of EVRs where each EVR is wrapped in a minimum bounding box (MBR). An EVR consists of the region vertices with respect to a data object and a pointer to the corresponding object entry in the object cache. When an NN query point q is located in an EVR of the EVR-tree, the proxy retrieves the corresponding object from the object cache to answer the query. On the other hand, the service area is divided into $m \times n$ grid cells managed by the grid index. Grid cells are classified into two categories: *fully cached cells* and *uncached cells*. All grid cells are initialized to uncached. The proxy marks a cell as fully cached when all the objects within the cell are received. The corresponding grid index entry of a fully cached cell caches the object pointers to the associated object entries in the object cache. The purpose of fully cached and uncached cells is to realize the stored object distribution, enabling the proxy to create EWVs of window queries effectively. When receiving a window query, the proxy obtains the result and creates the corresponding EWV by retrieving stored objects in the surrounding fully cached cells. Although the EVR-tree and the grid index are designed for NN and window queries, respectively, these two index structures mutually support each other.

An entry of the object cache is in the form of $\langle ID, (x,y), eID, cell\_flag, object\_info \rangle$. Each data object p is assumed to have a unique ID. (x,y) is the 2D coordinate of p. eID denotes the corresponding EVR ID and $cell\_flag$ indicates whether p is located in a fully cached cell. $object\_info$ contains the basic information of p. For example, the $object\_info$ of a restaurant includes the cuisine type, hours of operation, phone number, and so on. When the EVR of p is inserted into the EVR-tree, the proxy updates the eID of the corresponding object entry. Similarly, once the covering cell where p is located becomes fully cached, the $cell\_flag$ is set to true by the proxy. With the information, when p has to be replaced, the proxy can remove the corresponding EVR from the EVR-tree or revert the fully cached cell to uncached. It is worth noting that, when a fully cached cell is reverted to uncached, the proxy sets the $cell\_flag$ of all relevant object entries to false without removing the objects from the object cache for better performance because the objects are shared with the EVR-tree.

3 Nearest-Neighbor Query Processing

3.1 Overview

When receiving an NN query, a proxy first attempts to answer the query with the EVR-tree and the grid index. If the proxy cannot answer the query, it will submit one or two 2NN queries to the LBS server. The rationale of extending the received NN query into 2NN queries is that the distance between the nearest and second nearest objects is useful for building the EVR of the nearest object based on the theoretical result of [26].[1] By exploiting the objects returned by the LBS server, the proxy extends an existing or creates a new EVR. Besides, the returned objects are utilized to aid grid index growth, which will facilitate window query processing. For each NN query, the proxy provides the mobile client not only the answer object (the nearest object) but also an EVR for helping the client avoid subsequent queries. Note that these 2NN queries initially cause slightly heavier load on the LBS server, but they lead the proxy to provide effective EVRs for mobile clients efficiently. With the EVRs, the number of queries submitted by the clients and the load on the LBS server are reduced greatly. Note that given an object p, the EVR of p is denoted by $EVR(p)$. The processing steps executed by the proxy are as follows:

- *Step 1.* The proxy checks whether the NN query location (x_q, y_q) is in any EVR of the EVR-tree by performing the general R-tree search operation since EVR-tree is an R-tree (or its variant). If so, go to Step 7.
- *Step 2.* If not, the proxy attempts to answer the query with the grid index. If the two nearest objects, say p_1 and p_2, are found in the grid index,[2] the proxy looks up the EVR-tree to see whether p_1 is located in any existing EVR. If so, go to step 6.

1. Specifically, Song and Roussopoulos [26] showed that, given a kNN query with query location q, the server first extends the kNN query into a $(k+1)$NN query with the same query location and then gets the $k+1$ nearest objects with respect to q. Suppose that these $k+1$ objects, $\{p_1, p_2, \ldots, p_{k+1}\}$, are ordered by their distance to q in ascending order. The first k objects (i.e., $\{p_1, p_2, \ldots, p_k\}$) are the answer objects of the original kNN query and the distance between the kth and $(k+1)$th objects (i.e., p_k and p_{k+1}) can be used to define a region so that $\{p_1, p_2, \ldots, p_k\}$ remain the answer objects of the kNN query as long as the new query location is within the region.

2. The details of finding p_1 and p_2 by the grid index are described in Section 3.2.

- *Step 3.* The proxy extends the received NN query into a 2NN query with the same query location (x_q, y_q) and submits the 2NN query to the LBS server. Let p_1 and p_2 be the nearest and the second nearest objects to q, respectively. When receiving p_1 and p_2 from the LBS server, the proxy searches the EVR-tree for $EVR(p_1)$. Meanwhile, the proxy runs algorithm GridIndexUpdate[3] to update the grid index based on q, p_1, and p_2. If $EVR(p_1)$ is found, go to Step 6.
- *Step 4.* The proxy generates another 2NN query with query location (x_1, y_1) where (x_1, y_1) is the location of p_1. Obviously, the nearest object to (x_1, y_1) is p_1. Let the second nearest object to (x_1, y_1) be p_3. The proxy runs algorithm EVR-Creation[4] to create the EVR of p_1 based on p_1, p_2, and p_3. At the same time, the proxy runs algorithm GridIndexUpdate to update the grid index based on p_1 and p_3.
- *Step 5.* The proxy inserts p_1 and $EVR(p_1)$ into the object cache and the EVR-tree, respectively. Go to Step 7.
- *Step 6.* With q, p_1, and p_2, the existing $EVR(p_1)$ is extended using algorithm EVR-Extension.[5] The updated $EVR(p_1)$ is reinserted into the EVR-tree.
- *Step 7.* The proxy returns the answer object p_1 and the corresponding $EVR(p_1)$ to the mobile client.

It is worth mentioning that 1) the grid index for window query processing is helpful for resolving NN queries as well as extending existing EVRs, and 2) the returned answer objects of NN queries benefit grid index growth. The mutual support of these two index structures will be detailed in the following sections.

3.2 Resolving NN Queries by Grid Index

When the proxy cannot answer an NN query by the EVR-tree, it attempts to exploit the grid index and cached objects to resolve the query. With the grid index, the proxy examines whether the query point q lies in a fully cached cell. If so, the proxy retrieves the nearest object p_1 and the second nearest object p_2 to q via the grid index. Besides p_1, the proxy retrieves p_2 since we intend to update $EVR(p_1)$ in case that $EVR(p_1)$ exists. With p_1 and p_2, the proxy calculates $dist(q, p_2)$ and creates a circle C centered at q with $dist(q, p_2)$ as the radius. Note that $dist(p, q)$ denotes the euclidean distance between any two given points p and q. If the circle C does not overlap any uncached cell, p_1 and p_2 can be guaranteed to be the real nearest and second nearest objects to q, as shown in Fig. 4. The proxy proceeds to examine whether p_1 is located in any EVR of the EVR-tree. If $EVR(p_1)$ is found, the proxy updates $EVR(p_1)$ by algorithm EVR-Extension with q, p_1, and p_2. Finally, the proxy returns p_1 together with the updated $EVR(p_1)$ to the query client. Note that, if any uncached cell is involved, the NN query cannot be resolved by the grid index. For example, in Fig. 4, the circle C', centered at q' and with radius as $dist(q', p_2')$, overlaps an uncached cell $Cell_u$. An object may be located in the overlapping region of $Cell_u$ and C' such that p_2' is not the real second nearest object to q'.

3. The details of algorithm GridIndexUpdate are described in Section 3.4.
4. The details of algorithm EVR-Creation are described in Section 3.3.
5. The details of algorithm EVR-Extension are described in Section 3.5.

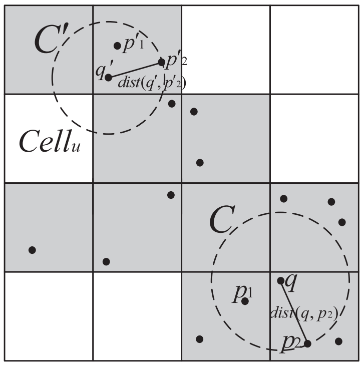


Fig. 4. Resolving an NN query by the grid index.

3.3 EVR Creation

When neither the EVR-tree nor the grid index can be used to answer the NN query with the location (x_q, y_q), the proxy will create a new EVR for the answer object p_1 of the query as follows: First, the proxy extends the NN query into a 2NN query and submits the 2NN query to the LBS server. After obtaining the answer objects p_1 and p_2 of the 2NN query (where $dist(q, p_1) < dist(q, p_2)$), the proxy starts to create the EVR by first building a circle C_1, which is centered at q with r_1 as the radius where $r_1 = (dist(q, p_2) - dist(q, p_1))/2$. Let the location of p_1 be (x_1, y_1). Next, the proxy submits another 2NN query with the query location (x_1, y_1) to the LBS server. Obviously, the nearest object to the location (x_1, y_1) is p_1. Let the second nearest object of this new 2NN query be p_3 with the location (x_3, y_3). Similarly, as shown in Fig. 5b, the proxy builds another circle C_2 with center (x_1, y_1) and radius r_2 where $r_2 = dist(p_1, p_3)/2$. Then, with q as the origin and θ as the angle between q and p_1, the proxy creates seven vertices $v_{new1}(x_{new1}, y_{new1})$, $v_{new2}(x_{new2}, y_{new2})$, $v_{new3}(x_{new3}, y_{new3})$, $v_{new4}(x_{new4}, y_{new4})$, $v_{new5}(x_{new5}, y_{new5})$, $v_{new6}(x_{new6}, y_{new6})$, and $v_{new7}(x_{new7}, y_{new7})$ by the following equations:

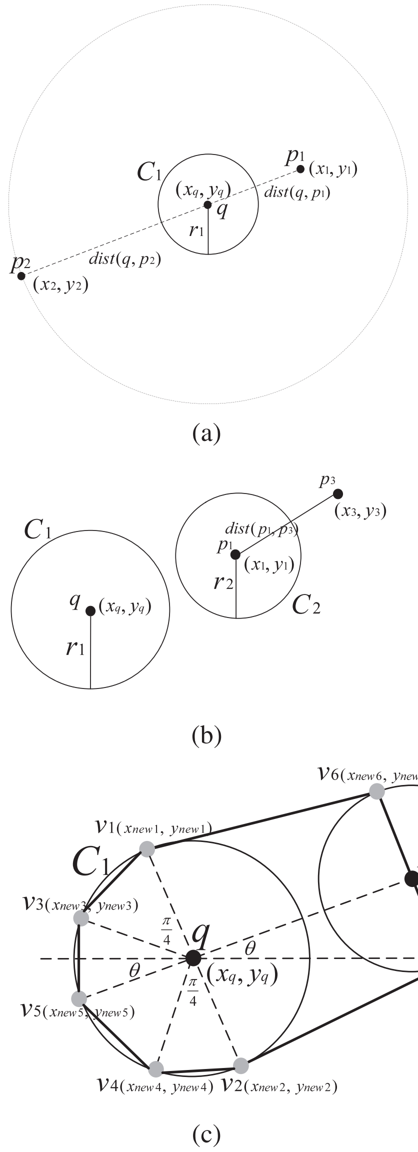
$$\begin{array}{ll}
x_{new1} = x_q + r_1 \cos\left(\theta + \frac{\pi}{2}\right) & y_{new1} = y_q + r_1 \sin\left(\theta + \frac{\pi}{2}\right) \\
x_{new2} = x_q + r_1 \cos\left(\theta - \frac{\pi}{2}\right) & y_{new2} = y_q + r_1 \sin\left(\theta - \frac{\pi}{2}\right) \\
x_{new3} = x_q + r_1 \cos\left(\theta + \frac{3}{4}\pi\right) & y_{new3} = y_q + r_1 \sin\left(\theta + \frac{3}{4}\pi\right) \\
x_{new4} = x_q + r_1 \cos\left(\theta - \frac{3}{4}\pi\right) & y_{new4} = y_q + r_1 \sin\left(\theta - \frac{3}{4}\pi\right) \\
x_{new5} = x_q + r_1 \cos(\theta + \pi) & y_{new5} = y_q + r_1 \sin(\theta + \pi) \\
x_{new6} = x_1 + r_2 \cos\left(\theta - \frac{\pi}{2}\right) & y_{new6} = y_1 + r_2 \sin\left(\theta - \frac{\pi}{2}\right) \\
x_{new7} = x_1 + r_2 \cos\left(\theta + \frac{\pi}{2}\right) & y_{new7} = y_1 + r_2 \sin\left(\theta + \frac{\pi}{2}\right).
\end{array}$$

Finally, the polygon formed by v_{new1}, v_{new2}, v_{new3}, v_{new4}, v_{new5}, v_{new6}, and v_{new7} is the new EVR of p_1, as depicted in Fig. 5c.

In fact, there are numerous combinations of the points on or within C_1 and C_2 to form a feasible EVR. However, considering the computational overhead, transmission cost, and limited storage space of a mobile device, the number of vertices of an EVR has to be limited. Thus, a new EVR is represented by only the seven calculated vertices to achieve effectiveness at low cost. A reader interested in determining the best number of the vertices of an EVR can refer to [2].

3.4 Updating Grid Index by Results of NN Queries

In addition to EVR creation, we exploit the answer objects p_1 and p_2 to speed up grid index growth. We observe that concerning the 2NN query with query point q, there exist only two objects p_1 and p_2 within the circle C centered at q



(a)
(b)
(c)
C_1
p_1 (x_1, y_1)
(x_q, y_q)
$dist(q, p_1)$
q
r_1
p_2
$dist(q, p_2)$
(x_2, y_2)
p_3
(x_3, y_3)
$dist(p_1, p_3)$
r_2
C_2
$v_6(x_{new6}, y_{new6})$
$v_1(x_{new1}, y_{new1})$
$v_3(x_{new3}, y_{new3})$
$v_5(x_{new5}, y_{new5})$
$v_7(x_{new7}, y_{new7})$
$v_4(x_{new4}, y_{new4})$
$v_2(x_{new2}, y_{new2})$
$\frac{\pi}{4}$
θ

Fig. 5. EVR creation. (a) Building C_1. (b) Building C_2. (c) Creating an EVR.

with $dist(q, p_2)$ as the radius. Based on this observation, we attempt to take advantage of the answer objects and algorithm GridIndexUpdate to mark uncached cells as fully cached cells as follows: When the proxy receives p_1 and p_2, it uses the circle C to determine all totally overlapped cells of circle C, as shown in Fig. 6. A cell with respect to the circle C is referred to as a *totally overlapped cell* if the entire cell is within C. If a totally overlapped cell is uncached, this cell becomes fully cached, as the $Cell_i$ in Fig. 6. By doing so, the proxy can accelerate grid index growth, resulting in the creation of more effective EWVs of window queries.

3.5 EVR Extension

As mentioned earlier, when the proxy receives the answer objects p_1 with location (x_1, y_1) and p_2 with location (x_2, y_2) of the submitted 2NN query, it first checks whether $EVR(p_1)$ is cached. If so, the proxy invokes algorithm EVR-Extension to update the EVR by the following steps. First, the proxy retrieves the existing $EVR(p_1)$ from the EVR-tree. Let (x_{old}, y_{old}) stand for the centroid of $EVR(p_1)$

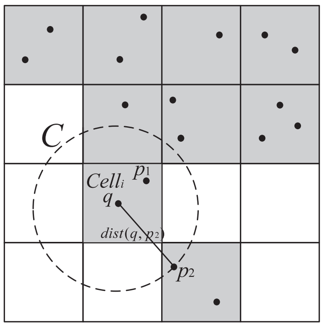


Fig. 6. A totally overlapped uncached cell is marked as a fully cached cell.

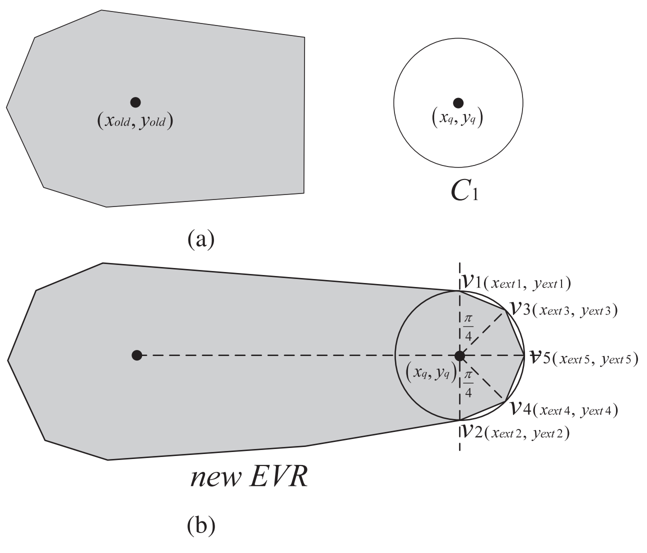
and build the circle C_1 by the method given in Section 3.3, as depicted in Fig. 7a. The proxy proceeds to calculate the coordinates of five new vertices $v_{ext1}(x_{ext1}, y_{ext1})$, $v_{ext2}(x_{ext2}, y_{ext2})$, $v_{ext3}(x_{ext3}, y_{ext3})$, $v_{ext4}(x_{ext4}, y_{ext4})$, and $v_{ext5}(x_{ext5}, y_{ext5})$ with (x_{old}, y_{old}) as the origin and θ as the angle between q and p_1, as shown in Fig. 7b. The coordinates of the five vertices v_{ext1}, v_{ext2}, v_{ext3}, v_{ext4}, and v_{ext5} are determined as below

$$\begin{array}{ll}
x_{ext1} = x_q + r_1 \cos\left(\theta + \frac{\pi}{2}\right) & y_{ext1} = y_q + r_1 \sin\left(\theta + \frac{\pi}{2}\right) \\
x_{ext2} = x_q + r_1 \cos\left(\theta - \frac{\pi}{2}\right) & y_{ext2} = y_q + r_1 \sin\left(\theta - \frac{\pi}{2}\right) \\
x_{ext3} = x_q + r_1 \cos\left(\theta + \frac{3}{4}\pi\right) & y_{ext3} = y_q + r_1 \sin\left(\theta + \frac{3}{4}\pi\right) \\
x_{ext4} = x_q + r_1 \cos\left(\theta - \frac{3}{4}\pi\right) & y_{ext4} = y_q + r_1 \sin\left(\theta - \frac{3}{4}\pi\right) \\
x_{ext5} = x_q + r_1 \cos(\theta + \pi) & y_{ext5} = y_q + r_1 \sin(\theta + \pi).
\end{array}$$

The original $EVR(p_1)$ is then extended with the five vertices v_{ext1}, v_{ext2}, v_{ext3}, v_{ext4}, and v_{ext5}. Since an updated EVR may become a concave polygon, we employ the Melkman's algorithm [27] to compute the convex polygon of the updated EVR to remove the unnecessary vertices and achieve a larger region size. The convex polygon serves as the final updated EVR of p_1. Interested readers can refer to [28] for the proofs of the proposed algorithms.

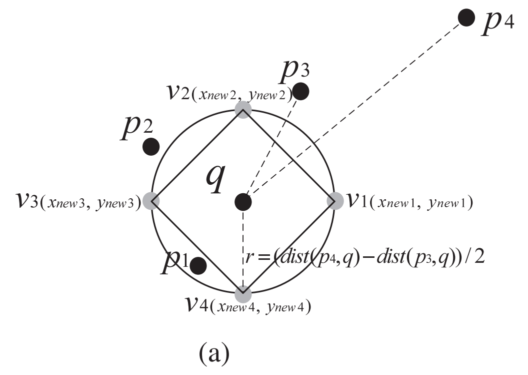
3.6 EVR of kNN Query

Although the proposed algorithms for EVR creation and extension are optimized for NN queries, after minor

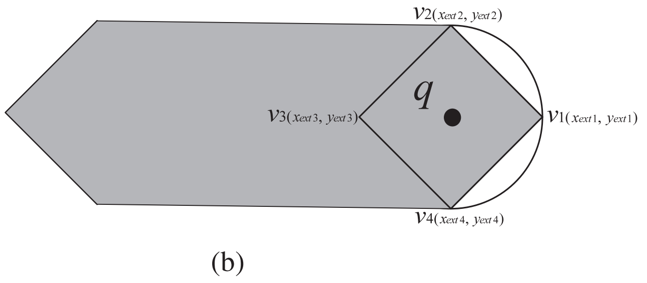


(a)
(b)
(x_{old}, y_{old})
(x_q, y_q)
C_1
$v_1(x_{ext1}, y_{ext1})$
$v_3(x_{ext3}, y_{ext3})$
$v_5(x_{ext5}, y_{ext5})$
$v_4(x_{ext4}, y_{ext4})$
$v_2(x_{ext2}, y_{ext2})$
$\frac{\pi}{4}$
new EVR

Fig. 7. Extending an existing EVR. (a) Before extending. (b) After extending.



p_4
p_3
$v_2(x_{new2}, y_{new2})$
p_2
q
$v_3(x_{new3}, y_{new3})$
$v_1(x_{new1}, y_{new1})$
p_1
$r=(dist(p_4,q)-dist(p_3,q))/2$
$v_4(x_{new4}, y_{new4})$
(a)



$v_2(x_{ext2}, y_{ext2})$
q
$v_3(x_{ext3}, y_{ext3})$
$v_1(x_{ext1}, y_{ext1})$
$v_4(x_{ext4}, y_{ext4})$
(b)

Fig. 8. EVR of a 3NN query. (a) EVR creation. (b) EVR extension.

modifications and disabling some optimization techniques, the algorithms can support kNN queries. Specifically, when receiving a kNN query with query location $q(x_q, y_q)$, the proxy extends the query into a $(k+1)$NN query and submits the $(k+1)$NN query to the LBS server. On receipt of the answer objects $\{p_1, p_2, \ldots, p_{k+1}\}$ from the LBS server, the proxy builds a circle C with center q and radius $r = (dist(q, p_{k+1}) - dist(q, p_k))/2$ where $dist(q, p_k) < dist(q, p_{k+1})$. Next, the proxy creates four vertices $v_{new1}(x_{new1}, y_{new1})$, $v_{new2}(x_{new2}, y_{new2})$, $v_{new3}(x_{new3}, y_{new3})$, and $v_{new4}(x_{new4}, y_{new4})$ by the following equations:

$$x_{new1} = x_q + r \quad y_{new1} = y_q \quad x_{new2} = x_q \quad y_{new2} = y_q + r$$
$$x_{new3} = x_q - r \quad y_{new3} = y_q \quad x_{new4} = x_q \quad y_{new4} = y_q - r.$$

Finally, the polygon formed by v_{new1}, v_{new2}, v_{new3}, and v_{new4} is the new EVR of the kNN query. Fig. 8a is an example of EVR creation for a 3NN query.

If a subsequent kNN query with query location q' has answer objects $\{p_1, \ldots, p_k\}$, the proxy will check whether any existing EVR is associated with exactly the same $\{p_1, \ldots, p_k\}$. If the associated EVR exists, the proxy will extend the EVR by adding four new vertices $v_{ext1}(x_{ext1}, y_{ext1})$, $v_{ext2}(x_{ext2}, y_{ext2})$, $v_{ext3}(x_{ext3}, y_{ext3})$, and $v_{ext4}(x_{ext4}, y_{ext4})$. v_{ext1}, v_{ext2}, v_{ext3}, and v_{ext4} are created by using the above equation with location q' in a similar manner. The proxy proceeds to compute the convex polygon of the updated EVR as the final EVR. An example of EVR extension for a 3NN query is shown in Fig. 8b.

4 Window Query Processing

4.1 Overview

On receiving a window query, the proxy first checks whether all the answer objects of the query are available in the object cache by looking up the grid index. Answer objects are the objects within the query window. According to the grid

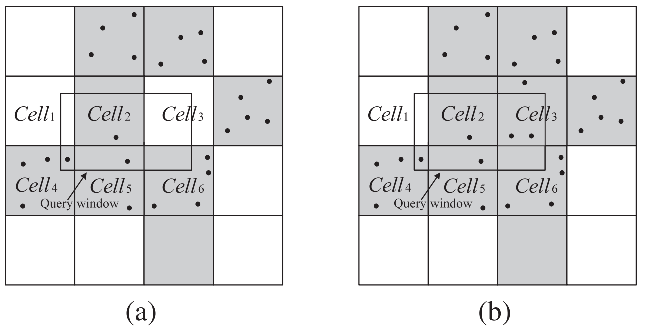


Fig. 9. Grid index growth. (a) Uncached $Cell_3$. (b) Fully cached $Cell_3$.

index, the proxy can know whether the grid cells overlapping the query window are fully cached. If a grid cell overlapping the query window is fully cached (e.g., $Cell_2$, $Cell_4$, $Cell_5$, and $Cell_6$ in Fig. 9a), the answer objects within the overlapping region can be directly retrieved from the object cache. Otherwise, the proxy uses the overlapping region of each uncached cell to generate a new window query (e.g., the overlapping regions of $Cell_1$ and $Cell_3$ in Fig. 9a). Then, the proxy submits the new window query to request the required answer objects from the LBS server. However, to accelerate fully cached cell growth, the proxy utilizes the corresponding entire cell as the new query window instead of the overlapping region if 1) the ratio of the region size to the cell size is above the predefined threshold, or 2) the number of interested queries in this cell exceeds the predefined threshold. For example, in Fig. 9a, the ratio of the overlapping region size to $Cell_3$ size is assumed to exceed the predefined threshold. The proxy sends a new window query with $Cell_3$ as the query window to the LBS server. Next, the proxy marks $Cell_3$ as fully cached when it receives all data objects in $Cell_3$ from the server, as shown in Fig. 9b. After obtaining the objects from the server, the proxy has all the answer objects. Finally, the proxy returns the answer objects together with the corresponding EWV to the query client. Note that these new queries used to accelerate grid index grow increase the load on the LBS server initially, but they lead to effective EWV creation, significantly reducing the number of subsequent queries submitted by clients. The processing steps executed by the proxy are as follows:

- *Step 1.* The proxy identifies the overlapping cells based on the width w, the length l, and the center (x_q, y_q) of the query window.
- *Step 2.* The proxy examines which of the overlapping cells being fully cached, searches the grid index for the object pointers within these overlapping fully cached cells, and then retrieves the answer objects from the object cache.
- *Step 3.* For each unresolved overlapping region, the proxy sends a new query with either the region or the corresponding entire uncached cell as the query window to the LBS server.
- *Step 4.* After receiving the objects from the LBS server, the proxy creates the EWV by algorithm EWV-Creation.[6] Meanwhile, if all the objects in one cell are received, the proxy marks the cell in the grid

6. The details of algorithm EWV-Creation are described in Section 4.2.

index as fully cached and stores the objects into the object cache.

- *Step 5.* The proxy returns the answer objects along with the corresponding EWV to the query client.

4.2 EWV Creation

As mentioned in Section 2.1, since the VRs of window queries may be concave polygons, the proxy in [6] pessimistically builds EVRs in the form of polygons based on previous query points. The polygonal EVRs are extremely small and ineffective. To overcome this problem, we propose to cache data objects indexed by the grid index and represent the EVRs in the form of vectors, called estimated window vectors, to enlarge the estimated valid regions. Due to rectangular windows and cached objects, the EWVs can be set effectively. An EWV consists of estimated valid distance components and is denoted by

$$V = \langle Dx_{ne}, Dy_{ne}, Dx_{nw}, Dy_{nw}, Dx_{se}, Dy_{se}, Dx_{sw}, Dy_{sw} \rangle.$$

Each component is defined as the estimated valid distance with respect to a specific direction. For example, Dx_{ne} and Dy_{ne} are defined as the east and north estimated valid distances with respect to the northeast direction, respectively. Given the moving direction and distances, the corresponding EWV components enable a mobile client easily to determine the validity of the cached query result. Consider a mobile client issued a window query and received an EWV from the proxy at the location (x_q, y_q). When the mobile client moves northeast and wants to launch a new window query at the new location (x_n, y_n), the client calculates the moving vector components $x_n - x_q$ and $y_n - y_q$ and checks whether they are less than Dx_{ne} and Dy_{ne} of the EWV, respectively. If so, the answer objects associated with the EWV remain valid and the mobile device can directly show the cached result to the mobile client without issuing the new query to the proxy. In what follows, we describe the EWV creation in detail.

For EWV creation, two requirements must be met: 1) all answer objects p_i's must remain valid and 2) no outer objects o_i's will be encountered. The data objects outside the query window are called *outer objects*. Since the value of each component is determined in a similar manner, we use Dx_{ne} and Dy_{ne} for the northeast direction to explain the process. To simplify the process of determining Dx_{ne} and Dy_{ne}, the proxy first calculates the estimated valid distances D_e and D_n for the query window moving east and north, respectively. Determining D_e and D_n in advance helps the proxy calculate Dx_{ne} and Dy_{ne} because the query window heading northeast may encounter the o_i's in the east and north directions. As shown in Fig. 10, for D_e, the proxy first calculates the minimum distance between p_i's and the left side L_W of the query window to ensure the validity of p_i's. The minimum distance D_{ie} is set to

$$D_{ie} = \min_{\forall i} \{dist(p_i, L_W)\}. \tag{1}$$

Note that given a point $p(x_p, y_p)$ and a line segment L with end points $(x1_L, y1_L)$ and $(x2_L, y2_L)$, the distance $dist(p, L)$ equals $|x_p - x1_L|$ as L is vertical or $|y_p - y1_L|$ as L is horizontal. Considering p_1 and p_2 in Fig. 10, the proxy sets

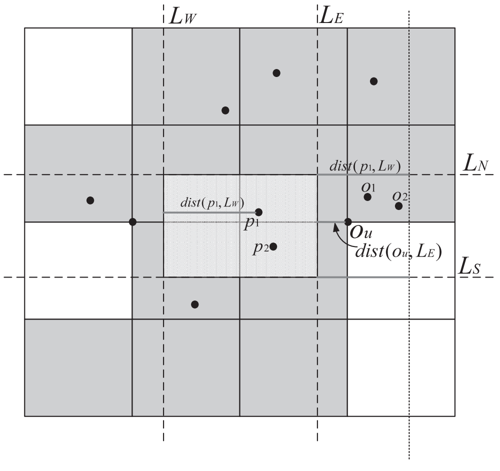


Fig. 10. An example for D_e.

D_{ie} to $dist(p_1, L_W)$. Clearly, p_i's are guaranteed to be valid while the window is moving east at a distance less than D_{ie}.

Next, the proxy retrieves o_i's in the surrounding fully cached cells to calculate the distance that prevents the query window from encountering o_i's. Because the object distributions in the surrounding uncached cells are unknown, the corner points of the uncached cells (e.g., o_u in Fig. 10) are added as outer objects for the worst case. With D_{ie}, the proxy considers only those o_i's with distances to L_E less than D_{ie}. Specifically, the proxy takes into account only those o_i's, which are located right to L_E, having y-coordinates between L_N and L_S and distances to L_E less than D_{ie}. Note that in case of no object within the query window, D_{ie} is set to W, the width of the grid index. Based on the o_i's, D_e is set to the minimum distance between o_i's and L_E. In Fig. 10, D_e is set to $dist(o_v, L_E)$. The similar steps are applied to determine D_n.

With D_e and D_n, the proxy proceeds to tackle the o_i's in the northeast direction of the query window. Based on D_e and D_n, we introduce a virtual object and a bound region to reduce the search region of o_i's. A *virtual object* o_v is an outer object created by the corresponding estimated valid distances D_e and D_n, as depicted in Fig. 11. The region constructed by o_v as well as L_E and L_N is called a *bound region* (e.g., the region specified by gray dashed lines in Fig. 11). Recalling the derivation of D_e and D_n, the query window cannot move out of the bound region and thus only the o_i's in the bound region have to be considered. With the o_i's in the bound region, the proxy aims to determine the point that maximizes the values of Dx_{ne} and Dy_{ne} because the larger values of Dx_{ne} and Dy_{ne} lead to the larger estimated valid region. Since deriving the best point is time consuming and we focus on minimizing query answering time, we present an efficient heuristic scheme to get a feasible point. For each outer object o_i in the bound region, the proxy examines whether its constructed region contains no $o_j, j \neq i$. If so, the proxy calculates and stores the size of the constructed region of the feasible o_i (e.g., o_3 and o_4 in Fig. 11), denoted by $Area(o_i)$. Otherwise, this o_i (e.g., o_v and o_2 in Fig. 11) is ignored. The proxy then uses the object o_m as the desired point where o_m is

$$o_m = \arg\max_i \{Area(o_i)\}. \tag{2}$$

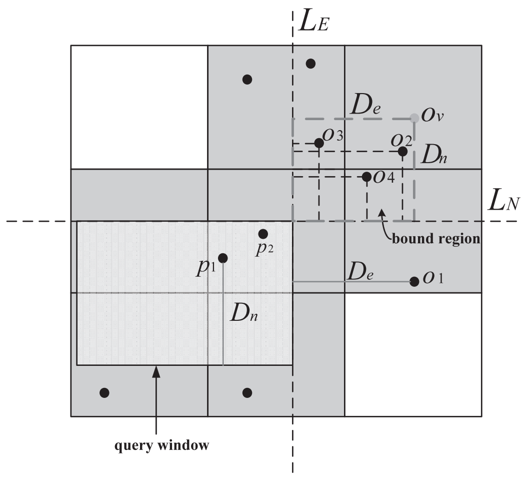


Fig. 11. A virtual object and the associated bound region.

Finally, the proxy defines Dx_{ne} as $dist(o_m, L_E)$ and Dy_{ne} as $dist(o_m, L_N)$. Interested readers can refer to [28] for the proof of the proposed algorithm.

4.3 Extension to Range Queries

Although we focus on window queries in this paper, the proposed approach is able to support range queries whose query region is circular. To resolve a range query, the main idea is to transform the range query to a window query and then address the window query with the window query processing algorithm. We describe the range query processing procedure as follows:

- *Step 1.* When the proxy receives a range query with query location $q(x_q, y_q)$ and radius r, it first transforms the range query into a window query with query window of the center $q(x_q, y_q)$, the width $2r$, and the length $2r$. The proxy executes the window query processing algorithm to resolve the window query and to compute the corresponding $EWV = <D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8>$.
- *Step 2.* Let the answer objects of the window query be $\{p_1, p_2, \ldots, p_n\}$. Without loss of generality, we reorder the answer objects in ascending order based on their distance to q so that $dist(q, p_j) \geq dist(q, p_i)$ when $j > i$.
- *Step 3.* Let $D_{min} = \min_{\forall i}\{dist(p_i, C)\}$ where $dist(p_i, C)$ denotes the shortest distance between p_i and the range query circle C.
- *Step 4.* For $i = 1$ to 8, $D_i = \min(D_i, D_{min})$.
- *Step 5.* The proxy filters out the answer objects $\{p_1, p_2, \ldots, p_n\}$ which are not within the query circle C and then has the final answer objects $\{p_1, p_2, \ldots, p_m\}$.
- *Step 6.* The proxy returns the answer objects $\{p_1, p_2, \ldots, p_m\}$ as well as the final EWV as the EVR of the range query to the query client.

5 Performance Evaluation

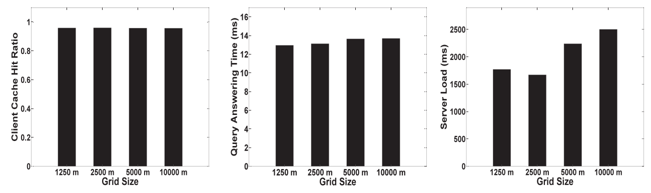
In this section, we evaluate the performance of the proposed approach. We implemented a simulator using Java and conducted several experiments on a Windows platform with 2.33G Intel Core 2 CPU and 3.2 GB RAM. Since [5] and [6] are the only proxy-based approaches in the literature, we compare the proposed approach with [5] for NN queries and [6] for window queries. Besides, although the proposed proxy-based approach serves as an alternative solution, we compared the proposed approach with the representative server-based approach [7], [8] to understand the benefits of the proxy architecture. The main performance metric is the query answering time, which is defined as the time elapsed from the moment a query is submitted to the moment the query result together with the corresponding EVR or VR are received. In addition, we use two performance metrics: client cache hit ratio and server load. The client cache hit ratio indicates the effectiveness of cached EVRs and VRs at mobile clients and energy consumption of mobile devices. The higher the client cache hit ratio is, the smaller number of queries submitted by mobile clients is. Since data transmissions are a source of power consumption of mobile devices, the smaller number of submitted queries leads mobile clients to consume less power. The server load is defined as the total computation time at the LBS server. We use the total computation time rather than the number of queries received by the LBS server since VR construction incurs a higher computational cost than query answer search does.

We employ the real data set that contains schools in California.[7] The data objects stored at the LBS server are indexed by an R-tree with a cache size equal to 5 percent of the R-tree index as the setting in [8]. The sizes of the EVR-tree and the grid index of our proxy are set to about 22 percent and about 2 percent of the proxy cache size, respectively. Considering that a proxy is often deployed in a densely populated area, we limit the service area to Los Angeles and the surrounding areas. The service area size is 120 km $\times$ 80 km. Within the service area, there are 3,461 schools. For mobile clients, we divide the service area into four subareas. Mobile clients are evenly assigned to be located in one subarea and move only within the subarea. The random waypoint model [29] is employed as the mobility model. Mobile clients may move at the speed of 0-2 m/s (0-7.2 km/hr) as pedestrians with probability about 50 percent or 10-20 m/s (36-72 km/hr) as vehicles with probability about 50 percent. The query generation intervals of mobile clients are assumed to be an exponential distribution with mean 6 hours. The duration of a continuous query follows an exponential distribution with mean 30 seconds. During the simulation, a mobile client may issue an NN query or a window query with equal probability. There are four sizes of query windows: $1{,}000\ \text{m} \times 1{,}000\ \text{m}$, $2{,}000\ \text{m} \times 2{,}000\ \text{m}$, $3{,}000\ \text{m} \times 3{,}000\ \text{m}$, and $4{,}000\text{m} \times 4{,}000\ \text{m}$. Each client caches an EVR for NN queries and an EVR (an EWV in our approach) for window queries. The EVR of an NN query consists of 14 points. The EVR of a window query in [5] is composed of four points same as the proposed EWV. The total simulation time is 24 hours. The default simulation parameters are summarized in Table 1 where networking parameters are referred to [30], [31]. The results of [5], [6] are labeled as "Gao" while those of [7], [8] are labeled as

7. http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm.

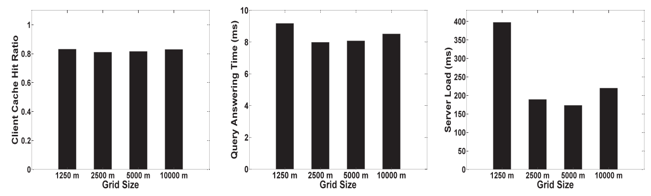
TABLE 1
Default System Parameters

Parameter	Value
Number of clients	40000
Query interval	Exp. dis. with μ = 6 hours
Query duration	Exp. dis. with μ = 30 seconds
Wired network bandwidth	100 Mbps
Wired network latency	0.5 ms
Uplink bandwidth	200 kbps
Downlink bandwidth	400 kbps
Data object size	256 bytes
Simulation time	24 hours



(a) (b) (c)

Fig. 12. Performance of NN queries versus grid size. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.



(a) (b) (c)

Fig. 13. Performance of window queries versus grid size. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.

"Lee." Note that since our approach is optimized for NN queries as well as window queries but not dedicated to kNN queries, we conduct the experiment for kNN queries only in Section 5.5.

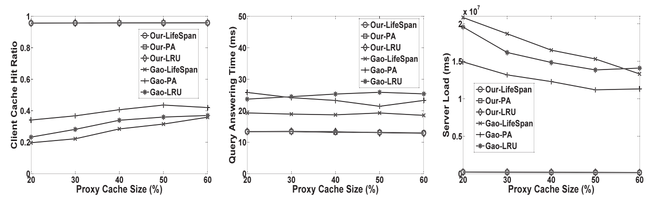
5.1 Effect of Grid Size

Before comparing with the other two approaches, we determine the optimal grid sizes of the grid index of our approach in this experiment. Figs. 12 and 13 show the experimental results of NN and window queries with the grid size varied, respectively. For NN queries, we observe from Fig. 12a that the client cache hit ratio is almost constant for all grid sizes. This is because the client cache hit ratio is primarily determined by EVRs and is almost independent of the grid size. From Figs. 12b and 12c, it can be seen that the query answering time and the server load become shorter and lower, respectively, as the grid size decreases. The reason is that a smaller grid size enables the proxy to resolve more NN queries by cached objects and EVRs. On the other hand, for window queries, Fig. 13a depicts that the client cache hit ratio increases with the grid size getting larger, arising from that a larger grid size results in more object information enabling the proxy to set more effective EWVs. From Fig. 13c, we observe that smaller grid sizes cause heavier server load, arising from that the proxy has to request objects for fully cached cells from the LBS server more frequently. Fig. 13b shows that the query answering time increases as the grid size gets larger. This result seems contrary to the intuition that, the high cache hit ratio yields the smaller number of queries submitted by clients and thus reduces the processing times of the proxy and the server. The underlying reason is as below.

When mobile clients want to launch new spatial queries and find that they are not located in their cached EVRs, it is highly likely that mobile clients are still in the VRs since EVRs are subregions of VRs. If mobile clients are not located in their EVRs but are in the corresponding VRs, the data objects cached at mobile clients remain valid. In this case, it is unnecessary for the proxy to transmit the data objects to query clients. Based on this observation, we introduce a *traffic-reduction* mechanism to reduce the wireless transmission cost as follows: When a mobile client wants to submit a query to the proxy, the client attaches the object ID(s) of the last query result associated with the cached EVR to the query. With the information, the proxy can realize whether the cached object(s) of mobile clients remain valid. If so, the proxy transmits only the new EVR to the client without the data object(s), thereby reducing the wireless transmission cost. With the traffic-reduction mechanism, let N_{valid} denote the number of queries submitted by the clients when mobile clients are outside the EVRs but the cached data objects are still valid. As the cache hit ratio decreases, N_{valid} will increase and thus the proxy will answer more queries with no or fewer objects returned to mobile clients, thereby reducing the query answering time. In the following experiments, we choose to set the grid size to 2,500 m due to better performance.

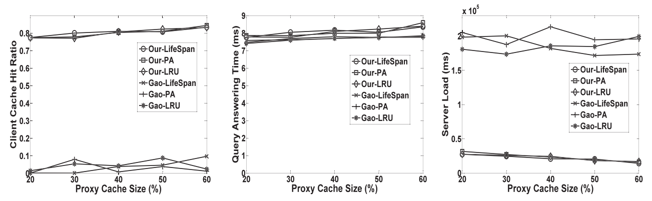
5.2 Effect of Proxy Cache Size

In this experiment, we investigate the effects of proxy cache size and cache replacement policy on the system performance. The cache size ranges from 20 to 60 percent of the storage space required for storing all data objects. Regarding cache replacement, we employ three policies: PA, LRU, and LifeSpan. PA was proposed in [2] and evicts the EVR with the least cost where the cost is defined as the product of the access probability and the EVR size. LifeSpan was suggested in [5], [6] and evicts the EVR that has been accessed more than n times. We set n to 512 in this



(a) (b) (c)

Fig. 14. Performance of NN queries versus proxy cache size. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.



(a) (b) (c)

Fig. 15. Performance of window queries versus proxy cache size. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.

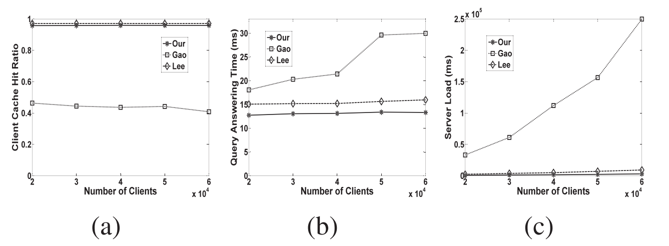


Fig. 16. Performance of NN queries versus number of clients. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.

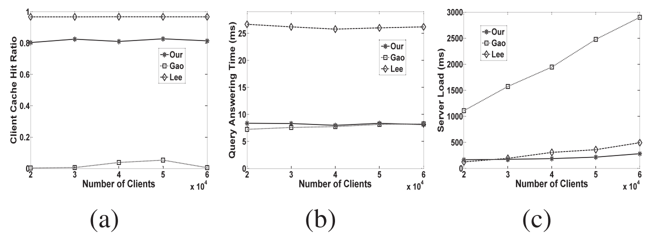


Fig. 17. Performance of window queries versus number of clients. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.

experiment. Figs. 14 and 15 reveal that the client cache hit ratio increases and server load decreases as the cache size increases, agreeing with the intuition. For NN queries, the client cache hit ratio of our approach is about two times higher than that of Gao's approach. Due to the low client cache hit ratio, Gao's approach performs very poorly in terms of server load and query answering time. For window queries, the proposed EWVs increase the client cache hit ratio by an order of magnitude compared with Gao's approach. Although the proxy and the server of Gao's approach have to process a large number of queries submitted by mobile clients, Gao's approach has close query answering time compared with our approach because of the traffic-reduction mechanism. Note that we adopt 50 percent of the storage space for data objects as the proxy cache size in the following experiments.

As to cache replacement policies, from Figs. 14 and 15, we can see that the effect of cache replacement policy on our approach is fairly minor. This is because the created EVRs and EWVs are effective in reducing the number of queries submitted by mobile clients, and such a smaller number of the received queries incurs low replacement cost at the proxy. As a result, our approach is less sensitive to the cache replacement policy. On the other hand, since Gao's approach has to accumulate sufficient query points to enlarge the EVRs, their performance is affected by the adopted cache replacement policy. Compared with LRU and LifeSpan, PA makes Gao's approach have a higher client cache hit ratio since PA takes into account the access probabilities and EVR sizes. It is worth noting that using LifeSpan, Gao's approach has a lower client cache hit ratio but achieves shorter query answering time. The reason is that PA and LRU incur EVR-tree reconfiguration (cache replacement) whenever a new object is received from the server. The high cost of frequent EVR-tree reconfiguration accounts for the long processing time at the proxy when PA or LRU is used. Since PA leads to the higher client cache hit ratio and lower server load, we employ PA as the replacement policy in the following experiments.

5.3 Effect of the Number of Clients

We evaluate the scalability of all the approaches by varying the number of mobile clients from 20,000 to 60,000 in increments of 10,000. As shown in Figs. 16 and 17, Lee's approach outperforms our and Gao's approaches in terms of the client cache hit ratio and server load since Lee's approach has the complete data object information to construct effective VRs. The performance of our approach is close to that of Lee's approach due to the effectiveness of EVRs and EWVs. Gao's approach suffers from poor performance because their EVRs of NN and window queries are small. Figs. 16a and 17a show that the client cache hit ratio is nearly constant because the number of clients has a small effect on VR and EVR construction. For NN queries, the increase in the number of mobile clients results in more submitted queries, causing the performance of Lee's and Gao's approaches to deteriorate in terms of query answering time and server load, as shown in Figs. 16b, 16c and 17b, 17c. Different from Lee's and Gao's approaches, our approach is insensitive to the increase in the number of mobile clients due to the fact that the cached EVRs and objects at the proxy are effective in resolving such queries. In fact, the cached EVRs and objects enable our approach to outperform Lee's approach in terms of server load when the number of mobile clients is large. Specifically, the server load of our approach is only about 30 percent of that of Lee's approach when the number of clients reaches 60,000. Similarly, for window queries, it is observed from Fig. 17c that our approach achieves lower server load than Lee's approach as the number of mobile clients is large. The reason is that as the number of mobile clients increases, the proxy is able to have more fully cached cells, thereby creating more effective EWVs for mobile clients. These results indicate that our approach is more scalable than Lee's approach.

To further understand the query answering time, Fig. 18 lists the distribution of time spent by each entity, including wireless transmission time, proxy processing time, wired transmission time, and server processing time. From Fig. 18, we can see that the query answering time is dominated by the wireless transmission entity. Compared with high-speed CPU, the wireless links are the bottleneck of obtaining query results for mobile clients. With respect to the server processing time, our approach experiences much less time than Lee's approach for two reasons. First, VR computation is more computationally costly than query answer search. Second, heavy VR computation results in longer queuing time on the server. Gao's approach suffers

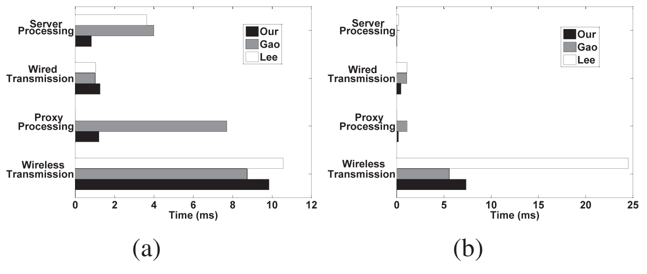


Fig. 18. Distribution of query answering time. (a) NN queries. (b) Window queries.

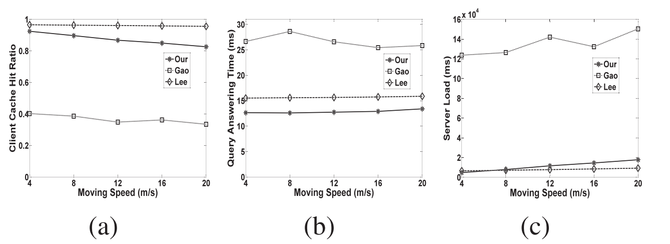


Fig. 19. Performance of NN queries versus moving speed. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.

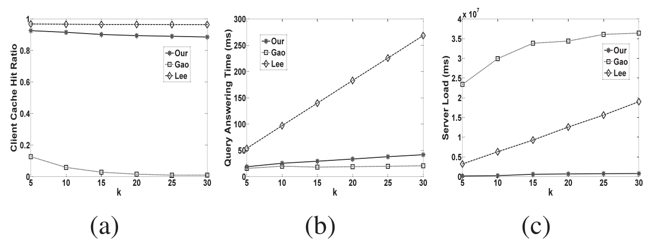


Fig. 21. Performance of kNN queries. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.

from the longest proxy processing time and server processing time due to the large number of the received queries caused by the low client cache hit ratio.

5.4 Effect of Moving Speed

In this experiment, we study the effect of moving speed on the performance. We vary the moving speed from 4 m/s (14.4 km/hr) to 20 m/s (72 km/hr). For NN queries, Fig. 19 shows that the high moving speed degrades the performance of all the approaches. This is because the higher the moving speed is, the more frequently a mobile client moves out of its cached EVR or VR, resulting in more submitted queries. Besides, Fig. 19 shows that the moving speed affects our approach more than the other approaches. Specifically, as the moving speed increases from 4 to 20 m/s, the client cache hit ratio drops from about 95 to 84 percent such that our approach suffers from heavier server load than Lee's approach. The drop mainly results from that the EVRs which are subregions of VRs are less beneficial for mobile clients at high moving speeds. Compared with Gao's approach, however, our approach still enhances the performance significantly at high speeds since the built EVRs of our approach are much larger than those of Gao's approach. On the other hand, for window queries, the performance of all the approaches also degrades as the moving speed increases, as illustrated in Fig. 20. Unlike NN queries, the moving speed has a greater impact on Lee's approach than our approach in terms of server load. The reason is that, with the cached objects, the proxy of our approach can directly answer many received queries, thereby leading to lower server load. Due to the extremely low client cache hit ratio, Gao's approach performs poorly at all moving speeds.

5.5 Effect of k

We vary the value of k from 5 to 30 to study the performance of all the approaches for kNN queries. From Fig. 21a, we can see that the performance of Lee's and our approaches degrades gracefully in terms of client cache hit ratio as the value of k increases. This is because that about half of mobile clients moving slowly as pedestrians rarely move out of the EVRs or VRs for the querying duration. However, the query answering time of Lee's approach is much longer than that of our approach because 1) the query answering time of our approach is greatly reduced by using the traffic-reduction mechanism and 2) Lee's approach suffers from the much longer VR construction time for kNN queries, as shown in Fig. 21c. On the other hand, the large value of k causes the performance of Gao's approach to degrade severely for three reasons. First, a large value of k decreases the probability that the mobile clients have the exactly same answer objects, causing Gao's approach unable to accumulate sufficient query points for EVRs. Second, due to the large number of combinations of answer objects of kNN queries and small cache size, an EVR is usually replaced before it grows sufficiently large. Third, since the cache size is fixed, the more objects stored due to the large value of k cause a smaller number of EVRs at the proxy.

To further explore the effect of the value of k, we vary the moving speed of mobile clients from 4 to 20 m/s and the value of k is fixed at 20. Fig. 22 shows that Lee's approach still achieves high client cache hit ratio due to large VRs. However, the costly VR computation for kNN queries also causes Lee's approach to experience longer query answering time and have heavier server load. As to our approach, the client hit ratio drops from about 75 to 40 percent as the moving speed is varied from 4 to 20 m/s, as shown in Figs. 22a. The reason is that the EVRs are subregions of VRs of kNN queries and thus are much less effective than VRs. However, the traffic-reduction mechanism and cached information at the proxy lead our approach to enjoy shorter query answering time and lower server load, respectively. Finally, since Gao's approach has a very low client cache hit ratio, their performance stays poor at all moving speeds.

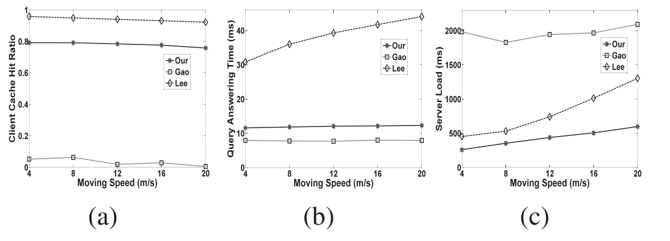


Fig. 20. Performance of window queries versus number of clients. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.

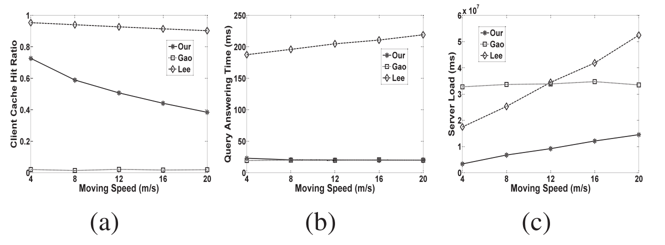


Fig. 22. Performance of 20NN queries versus moving speed. (a) Client cache hit ratio. (b) Query answering time. (c) Server load.

6 Comparison between Proxy-Based and Server-Based Approaches

Here, we discuss the benefits and drawbacks of the proposed proxy-based approach compared with Lee's server-based approach based on the experimental results in Section 5:

- *Client cache hit ratio.*Due to the effectiveness of VRs and EVRs, both of Lee's and our approaches achieve high client cache hit ratios, leading mobile clients to avoid unnecessary query submissions. However, compared with VRs, our EVRs are relatively smaller so that the client cache hit ratio of our approach is lower than that of Lee's approach particularly when the moving speed of mobile clients is high.
- *Query answering time.*Because of the high client cache hit ratio, the number of queries submitted by mobile clients is small for Lee's and our approaches. The fewer received queries together with the efficient processing algorithms and cached information at the proxy lead our approach to experience shorter query answering time. In other words, our approach makes a mobile client wait for only a short period before obtaining the answer object(s) and the corresponding EVR. It is worthwhile to mention that the cached EVRs and objects are more beneficial when the number of mobile clients is large. On the other hand, since VR computation is time consuming and VRs are not reused for resolving new queries, Lee's approach has longer query answering time than our approach, especially when a great deal of mobile clients submit spatial queries.
- *Server Load.*When the number of mobile clients is large, due to the cached EVRs and objects at the proxy, the LBS server of our approach enjoys the lower server load than the server of Lee's approach. On the other hand, if mobile clients move at high speeds, Lee's approach outperforms our approach in terms of server load since their large VRs lead to a higher client cache hit ratio. Because of the high client cache hit ratio, both our and Lee's approach achieve low server load.

To sum up, the performance of our approach is very close to that of Lee's approach even though our approach has only partial information of data objects. Lee's approach is more suitable when mobile clients move at high speeds, whereas our approach is suitable in a densely populated area.

7 Conclusions and Future Work

In this paper, we have proposed a proxy-based approach to continuous NN and window queries in mobile environments. The proxy takes advantage of spatial and temporal locality of spatial queries to create EVRs of NN and window queries. Different from prior work, we devised new EVR creation and extension algorithms for NN queries, enabling the proxy to build effective EVRs efficiently. The devised algorithms make the proxy achieve high performance even when the cache size is small. On the other hand, we propose to represent EVRs of window queries in the form of vectors, called estimated window vectors, to achieve larger estimated valid regions. Moreover, due to distinct characteristics, we introduce an EVR-tree and a grid index to process NN and window queries, respectively. The algorithms for mutual support of the EVR-tree and the grid index are developed to further enhance the system performance. The experimental results show that the proposed approach significantly outperforms the existing proxy-based approaches since our proposed algorithms create much larger EVRs for mobile clients.

Compared with the representative server-based approach, the experimental results indicate that the proposed proxy-based approach achieves similar performance even though the proxy has only partial information of data objects. Besides, the results reveal that the proposed proxy-based approach is suitable in a densely populated area, whereas the server-based approach is suitable when mobile clients move at high speeds. Although offering the above benefits, the inherent problem of data object updates needs further investigation for the proposed proxy-based approach. In future work, we will investigate the impact of data object updates on the proposed approach and extend the proposed approach to efficiently handle frequent object updates.

Acknowledgments

This work was supported in part by Industrial Technology Research Institute of Taiwan, ROC, under contract A301AR2210, and by National Science Council of Taiwan, ROC, under contracts 99-2221-E-009-140-MY2 and 99-2219-E-002-029.

References

[1] D. Lee, B. Zheng, and W.-C. Lee, "Data Management in Location-Dependent Information Services," *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 65-72, July-Sept. 2002.

[2] B. Zheng, J. Xu, and D.L. Lee, "Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments," *IEEE Trans. Computers*, vol. 15, no. 10, pp. 1141-1153, Oct. 2002.

[3] B. Zheng and D.L. Lee, "Processing Location-Dependent Queries in a Multi-Cell Wireless Environment," *Proc. Second ACM Int'l Workshop Data Eng. for Wireless and Mobile Access*, 2001.

[4] B. Zheng, J. Xu, W.-C. Lee, and D.L. Lee, "On Semantic Caching and Query Scheduling for Mobile Nearest-Neighbor Search," *Wireless Networks*, vol. 10, no. 6, pp. 653-664, Dec. 2004.

[5] X. Gao and A. Hurson, "Location Dependent Query Proxy," *Proc. ACM Int'l Symp. Applied Computing*, pp. 1120-1124, 2005.

[6] X. Gao, J. Sustersic, and A.R. Hurson, "Window Query Processing with Proxy Cache," *Proc. Seventh IEEE Int'l Conf. Mobile Data Management*, 2006.

[7] K.C. Lee, J. Schiffman, B. Zheng, and W.-C. Lee, "Valid Scope Computation for Location-Dependent Spatial Query in Mobile Broadcast Environments," *Proc. 17th ACM Conf. Information and Knowledge Management*, pp. 1231-1240, 2008.

[8] K.C.K. Lee, W.-C. Lee, H.V. Leong, B. Unger, and B. Zheng, "Efficient Valid Scope for Location-Dependent Spatial Queries in Mobile Environments," *J. Software*, vol. 5, no. 2, pp. 133-145, Feb. 2010.

[9] S. Prabhakar, Y. Xia, D.V. Kalashnikov, W.G. Aref, and S.E. Hambrusch, "Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects," *IEEE Trans. Computers*, vol. 51, no. 10, pp. 1124-1140, Oct. 2002.

[10] Y. Cai, K.A. Hua, and G. Cao, "Processing Range-Monitoring Queries on Heterogeneous Mobile Objects," *Proc. Fifth IEEE Int'l Conf. Mobile Data Management*, pp. 27-38, 2004.

[11] B. Gedik and L. Liu, "Mobieyes: A Distributed Location Monitoring Service Using Moving Location Queries," *IEEE Trans. Mobile Computing*, vol. 5, no. 6, pp. 1384-1042, Oct. 2006.

[12] H. Hu, J. Xu, and D.L. Lee, "A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 479-490, 2005.

[13] X. Xiong, M.F. Mokbel, and W.G. Aref, "Sea-Cnn: Scalable Processing of Continuous k-Nearest Neighbor Queries in Spatio-Temporal Databases," *Proc. IEEE Int'l Conf. Data Eng.*, pp. 643-654, 2005.

[14] X. Yu, K.Q. Pu, and N. Koudas, "Monitoring k-Nearest Neighbor Queries over Moving Objects," *Proc. 21st Int'l Conf. Data Eng.*, pp. 631-642, 2005.

[15] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao, "A Threshold-Based Algorithm for Continuous Monitoring of k Nearest Neighbors," *IEEE Trans. Knowledge Data Eng.*, vol. 17, no. 10, pp. 1451-1464, Nov. 2005.

[16] M.A. Cheema, Y. Yuan, and X. Lin, "Circulartrip: An Effective Algorithm for Continuous Knn Queries," *Proc. 12th Int'l Conf. Database Systems for Advanced Applications*, pp. 863-869, 2007.

[17] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R^*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 322-331, 1990.

[18] F. Aurenhammer, "Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure," *ACM Computing Surveys*, vol. 23, no. 3, pp. 345-405, Sept. 1991.

[19] B. Zheng, J. Xu, W.-C. Lee, and D.L. Lee, "Grid-Partition Index: A Hybrid Method for Nearest-Neighbor Queries in Wireless Location-Based Services," *The VLDB J.*, vol. 15, no. 1, pp. 21-39, Jan. 2006.

[20] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D.L. Lee, "Location-Based Spatial Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 443-454, 2003.

[21] Y. Tao and D. Papadias, "Time-Parameterized Queries in Spatio-Temporal Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 334-345, 2002.

[22] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik, "The V^*-Diagram: A Query-Dependent Method for Moving kNN Queries," *Proc. VLDB Conf.*, pp. 1095-1106, 2008.

[23] L. Kulik and E. Tanin, "Incremental Rank Updates for Moving Query Points," *Proc. Int'l Conf. Geographic, Information Science*, pp. 251-268, 2006.

[24] S. Dar, M.J. Franklin, B.T. Jónsson, D. Srivastava, and M. Tan, "Semantic Data Caching and Replacement," *Proc. 22th Int'l Conf. Very Large Data Bases*, pp. 330-341, 1996.

[25] W.-S. Ku, R. Zimmermann, and H. Wang, "Location-Based Spatial Query Processing in Wireless Broadcast Environments," *IEEE Trans. Mobile Computing*, vol. 7, no. 6, pp. 778-791, June 2008.

[26] Z. Song and N. Roussopoulos, "K-Nearest Neighbor Search for Moving Query Point," *Proc. Seventh Int'l Symp. Spatial and Temporal Databases*, pp. 79-96, 2001.

[27] A.A. Melkman, "On-Line Construction of the Convex Hull of a Simple Polyline," *Information Processing Letters*, vol. 25, pp. 11-12, 1987.

[28] J.-L. Huang and C.-C. Huang, "A Proxy-Based Approach to Continuous Location-Based Spatial Queries in Mobile Environments," technical report, Nat'l Chiao Tung Univ., 2011.

[29] T. Camp, J. Boleng, and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research," *Wireless Comm. Mobile Computing*, vol. 2, no. 5, pp. 483-502, Sept. 2002.

[30] S.-C. Lo, G. Lee, W.-T. Chen, and J.-C. Liu, "Architecture for Mobility and Qos Support in All-Ip Wireless Networks," *IEEE J. Selected Areas Comm.*, vol. 22, no. 4, pp. 691-705, May 2004.

[31] F.P. Tso, J. Teng, W. Jia, and D. Xuan, "Mobility: A Double-Edged Sword for Hspa Networks: A Large-Scale Test on Hong Kong Mobile Hspa Networks," *Proc. MobiHoc Conf.*, pp. 81-90, 2010.

Jiun-Long Huang received the BS and MS degrees from the Computer Science and Information Engineering Department, National Chiao Tung University in 1997 and 1999, respectively, and the PhD degree from the Electrical Engineering Department, National Taiwan University in 2003. Currently, he is an assistant professor in the Computer Science Department, National Chiao Tung University. His research interests include: mobile computing, wireless networks, and data mining. He is a member of the IEEE.

Chen-Che Huang received the BS degree in mathematics from National Central University in 2004 and the MS degree in computer science and information engineering from National Dong Hwa University in 2006. He is currently working toward the PhD degree in computer science at National Chiao Tung University. His research interests include spatial query processing and wireless sensor networks.