Efficient Continuous Nearest Neighbor Query in Spatial Networks using Euclidean Restriction*

Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi

University of Southern California
Department of Computer Science
Los Angeles, CA 90089-0781
[demiryur, banaeika, shahabi]@usc.edu

Abstract. In this paper, we propose an efficient method to answer continuous k nearest neighbor (CkNN) queries in spatial networks. Assuming a moving query object and a set of data objects that make frequent and arbitrary moves on a spatial network with dynamically changing edge weights, CkNN continuously monitors the nearest (in network distance) neighboring objects to the query. Previous CkNN methods are inefficient and, hence, fail to scale in large networks with numerous data objects because: 1) they heavily rely on Dijkstra-based *blind expansion* for network distance computation that incurs excessively redundant cost particularly in large networks, and 2) they *blindly map* all object location updates to the network disregarding whether the updates are relevant to the CkNN query result. With our method, termed ER-CkNN (short for *Euclidian Restriction* based CkNN), we utilize ER to address both of these shortcomings. Specifically, with ER we enable 1) *guided search* (rather than blind expansion) for efficient network distance calculation, and 2) *localized mapping* (rather than blind mapping) to avoid the intolerable cost of redundant object location mapping. We demonstrate the efficiency of ER-CkNN via extensive experimental evaluations with real world datasets consisting of a variety of large spatial networks with numerous moving objects.

1 Introduction

The latest developments in wireless technologies as well as the widespread use of GPS-enabled mobile devices have led to the recent prevalence of location-based services. Many of the location-based services rely on a family of spatial queries, termed nearest neighbor (NN) queries. In particular, a *Continuous k-NN* query (CkNN for short) continuously monitors the k data objects that are nearest (in network distance) to a given query object, while the data objects and/or the query object arbitrarily move on a spatial network. With CkNN, for example, a driver can use the automotive navigation system of her vehicle to continuously locate the

three nearest restaurants as the vehicle is moving along a path, or a pedestrian can use her GPS-enabled mobile device (cell phone, PDA, etc.) to locate the nearest transportation vehicles (e.g., taxis, buses, trams).

Currently, incremental monitoring (IMA) and its extension group monitoring algorithm (GMA) [9] is the only known method for answering CkNN queries with arbitrarily moving data and query objects. GMA extends IMA with shared execution paradigm by grouping the queries in the same sequence and monitoring them as a group (rather than individually). We refer to these algorithms as IMA/GMA in the rest of the paper. IMA/GMA is based on the incremental network expansion (INE) method [10] (originally proposed to answer one-time/snapshot kNN queries on *static* objects) to support CkNN queries on dynamic/moving objects. However, IMA/GMA is inefficient and, therefore, fails to scale in real-world scenarios where the spatial network is large and the data objects moving on the network are numerous. IMA/GMA is inefficient due to two main reasons. Firstly, to compute the network distance between the query point and the data objects in order to identify the k nearest neighbors, IMA/GMA frequently uses the computationally complex Dijkstra algorithm that relies on *blind network expansion* for network distance computation (costing $O(E + N log N)$ for a network with E edges and N nodes). With network expansion, starting from q all network nodes reachable from q in every direction are visited in order of their proximity to q (hence, a one-to-many search) until all k nearest data objects are located (see Figure 1). The overhead of executing network expansion is prohibitively high particularly in large networks with a sparse (but perhaps large) set of moving data objects, because such a blind search approach has to redundantly visit many network nodes which are away from the shortest paths to the nearest data objects. For example, Figure 1 depicts a real spatial network (namely, the road network of San Joaquin, CA) and illustrates the set of nodes that network expansion would have to visit (marked by the shaded area) to locate the first nearest data object (1-NN) for the query object q. In this case, 47.2% of the entire set of network nodes (8620 nodes out of total 18262) must be visited to find 1-NN.
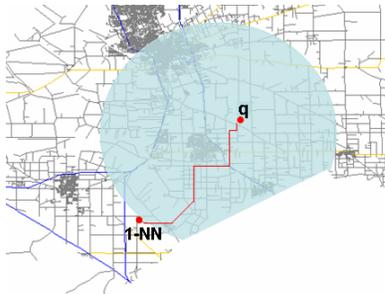

Fig. 1. Blind network expansion

Secondly, with IMA/GMA the cost of mapping the object location updates (i.e., the updates reporting the current coordinates, such as longitude-latitude, of the moving objects) to the network is also prohibitively high. While the location updates are continuously received, they must be mapped to the network to locate the current edge (that is then compared with the expansion-tree of each query)

of the moving object. However, with IMA/GMA *all* location updates are blindly and redundantly mapped to the network when they are received regardless whether they can possibly affect the CkNN query result, whereas most of the updates are irrelevant and can be ignored. Considering the cost of mapping each object location update (i.e., $O(logN)$) as well as high frequency of location updates in large spatial networks with numerous objects, the overhead incurred due to blind object location mapping with IMA/GMA becomes intolerable with real-world applications.

In this paper, we propose ER-CkNN, a *Euclidean Restriction* (ER) based method for efficient CkNN query answering. ER-CkNN addresses the two shortcomings of IMA/GMA by leveraging ER to enable *guided search* and *localized mapping*, respectively. Firstly, to identify the nearest neighbors of the query point q, ER-CkNN uses a filtering mechanism to rapidly find a set of candidate data objects based on their *Euclidean distance* from q (i.e., filtering by ER), which is then refined by computing their *network distance* from q to identify the exact set of nearest neighbors. The benefit of this filter-and-refine approach versus the blind network expansion approach is that once the candidate data objects are identified at the filter step, at the refine step ER-CkNN can use a one-to-one (rather than one-to-many) *guided search* algorithm such as A* [11] to perform the costly network distance computation with minimum redundant network traversal. With ER-CkNN, we use an EBE (Edge Bitmap Encoding)-based A*, with a search complexity proportional to the size of the shortest path. Secondly, to avoid the high cost of blind object location mapping, ER-CkNN only maps a location update to the network if it is relevant to the result of the CkNN query; otherwise, the location update is discarded. To determine whether a location update is relevant to a CkNN query, ER-CkNN uses ER to rapidly (i.e., in $O(1)$) identify whether the location update is within certain Euclidean locality that can potentially change the result of the CkNN query. If the update is within the q locality, ER-CkNN then maps the update to the network (i.e., *localized mapping*) which subsequently initiates the query update.

While ER is previously used for kNN query answering assuming *static* objects (e.g., with the incremental Euclidean restriction (IER) method [10]), to the best of our knowledge ER-CkNN is the first ER-based method proposed to answer CkNN queries on *dynamic/moving* objects. ER-CkNN is fundamentally different from previous ER-based approaches as they unanimously index the objects to apply ER, whereas with moving objects maintenance of such an index is unaffordable. Instead, ER-CkNN indexes the spatial network which is static, and uses a grid file (with $O(1)$ update cost) for efficient access to the objects in order to apply ER (see Section 4). Our experiments with real-world datasets show that ER-CkNN outperforms GMA with at least three times improved response time (see Section 6).

The remainder of this paper is organized as follows. In Section 2, we review the related work about kNN queries on moving objects in spatial networks. In Section 3, we formally define the CkNN query in spatial networks and introduce our system model. We establish the theoretical foundation of our proposed algorithms as well as our data structure and indexing schemes in Section 4. In Section 5, we discuss the factors that affect the performance of ER-CkNN. In Section 6, we present the results of our experiments with a variety of spatial networks with large number of moving objects. Finally, in Section 7 we conclude and discuss our future work.

2 Related Work

The research on kNN query processing can be categorized into two main areas, namely, query processing in Euclidean space and query processing in spatial networks.

2.1 kNN Queries in Euclidean Space

In the past, numerous algorithms [22, 21, 14, 16, 15] have been proposed to solve kNN problem in Euclidean space. Most of these algorithms, assuming the data objects are static, used tree-based (e.g., R-Tree) structures (or their extensions) to enable efficient query processing. Although the tree-based data structures are efficient in handling stationary spatial data, they suffer from the node reconstruction overhead due to frequent location updates with moving objects. Therefore, some researchers have exclusively used the simple but efficient space-based (i.e., grid) structures to index and query the moving objects [3, 20, 19, 7, 8]. All of these approaches are applicable to the spaces where the distance between objects is only a function of their spatial attributes (e.g., Euclidean distance). In real-world scenarios, however, the queries move in spatial networks, where the distance between a pair of data objects is defined as the length of the shortest path connecting them. We proceed to mention early proposals for kNN processing in spatial networks below.

2.2 kNN Queries in Spatial Networks

In [10], Papadias et al. introduced INE (discussed in Section 1) and IER. IER exploits the Euclidean restriction principle in spatial networks for achieving better performance. The data and query objects are assumed to be static in this work. In [13], Shahabi et al. proposed an *embedding technique* to transform a spatial network to a constraint-free high dimensional Euclidean space to fast but approximately retrieve nearest objects by applying traditional Euclidean based algorithms. Kolahdouzan and Shahabi utilized the first degree *network Voronoi diagrams* [4, 5] to partition the spatial network to network Voronoi polygons (NVP), one for each data object. They indexed the NVPs with a spatial access method to reduce the problem to a point location problem in Euclidean space and minimize the on-line network distance computation by precomputing the NVPs. Cho et al. [1] presented a system UNICONS where the main idea is to integrate the precomputed k nearest neighbors into the Dijkstra algorithm. Huang et al. addressed the same problem using *Island* approach [18] where each vertex is associated (and network distance precomputed) to all the data points that are centers of given radius r (so called islands) covering the vertex. With their approach, they utilized a restricted network expansion from the query point while using the precomputed islands. Aside from their specific drawbacks, these algorithms rely on *data object dependent* precomputations (i.e., the distance to the data objects are precomputed) and subdivide the spatial network based on the location of the data objects. Therefore, they assume that data objects are static and/or trajectory of query objects is known. This assumption is undesirable in applications where the query and data objects change their positions frequently.

Recently, Huang et al. [17] and Samet et al. [12] proposed two different algorithms that address the drawbacks of data object dependent precomputation. Huang et al. introduced *S-GRID* where they partition (using grid) the spatial network to disjoint sub-networks and precompute the shortest path for each pair of connected border points. To find the k nearest neighbors, they first perform a network expansion within the sub-networks and then proceed to outer expansion between the border points by utilizing the precomputed information. Samet et al. proposed a method where they associate a label to each edge that represents all nodes to which a shortest path starts with this particular edge. They use these labels to traverse *shortest path quadtrees* that enables geometric pruning to find the network distance between the objects. With these studies, the network edge weights are assumed to be static therefore the precomputations are invalidated with dynamically changing edge weights. This dependence is unrealistic for most of the real-world applications.

Therefore, unlike the previous approaches, we make the fundamental assumption that *both* the query and the data objects make frequent and arbitrary moves on a spatial network with dynamically changing edge weights. Our assumption yields a much more realistic scenario and versatile approach. To the best of our knowledge, the only comprehensive study proposed to this problem is IMA/GMA [9]. We discussed the shortcomings of IMA/GMA in Section 1.

3   Problem Definition

In this section, we formally define CkNN queries in spatial networks. Consider a spatial network (e.g., the Los Angeles road network) with a set of data objects and a query object. We assume the query object and the data objects either reside or move on the network edges. The position of a moving object p at time t is defined as $loc_t(p) = (x_p, y_p)$, where x_p and y_p are the cartesian coordinates of p in the space at time t. We assume all the relevant information about the moving objects and the spatial network is maintained at a central server. Whenever an object moves to a new location and/or the cost of an edge changes, the central server is updated with the new location and weight information, respectively. We formally define the spatial network, network distance, and CkNN queries as follows:

Definition 1. *A spatial network is a directional weighted graph $G(N, E)$, where N is a set of nodes representing intersections and terminal points, and E ($E \subseteq N \times N$) is a set of edges representing the network edges each connecting two nodes. Each edge e is denoted as $e(n_i, n_j)$ where n_i and n_j are starting and ending nodes, respectively.*

Definition 2. *Given a weighted graph $G(N, E)$ and set of edge weights $w : E \to \mathbf{R}$, the network distance d_N between a given source $s \in N$ and a given destination $t \in N$ is the length of the shortest path connecting s and t in G.*

Definition 3. *A Continuous k nearest neighbor (CkNN) query in spatial networks continuously monitors the k data objects that are nearest (in network distance) to a given query object, while the data objects and/or the query object arbitrarily move on network edges. Considering a set of n objects $S = \{p_1, p_2, ...p_n\}$, the k nearest neighbors of a query object q constitute a set $S^{'} \subseteq S$ of k objects such that for any data object $p^{'} \in S^{'}$ and $p \in S - S^{'}$, $d_N(p^{'}, q) \le d_N(p, q)$.*

4   ER-C*k*NN

Arguably, the main challenges with answering C*k*NN queries in spatial networks are 1) efficient network distance computation, and 2) effective maintenance of the query results given frequent changes of the moving object locations. With ER-C*k*NN, we employ a network partitioning approach that allows us to address the above challenges by enabling 1) guided shortest path algorithm that minimizes redundant network traversal, and 2) localized mapping that allows for effective maintenance of the query results.

ER-C*k*NN involves two phases: an off-line grid partitioning phase and an on-line query processing phase. During the off-line phase, the spatial network is partitioned into grid cells and each edge in the network is encoded whether it is a part of a shortest path to any node in a given grid cell (*edge-bitmap-encoding*). In addition, an *edge-cell-mapping* is computed between the edges of the spatial network and the cells of an overlaid grid. These precomputations are used to expedite the on-line query processing. During the on-line phase, a Euclidean Restriction (ER) based filter-and-refine method is adopted to identify the k nearest neighbors at the time of query arrival. At the filter step, ER-C*k*NN performs a grid expansion to rapidly identify a set of candidate nearest neighbors in the Euclidean proximity. The candidate set is then expanded to become a superset of the current k nearest neighbors of the query on the spatial network. At the refine step, the candidate set is refined (if necessary) by fast guided network distance computation exploiting the edge-bitmap-encoding information. However, considering the often large number of moving objects and their frequent location updates, effective maintenance/monitoring of this query result remains the main challenge. To address this challenge, we leverage the edge-cell-mapping information to rapidly identify the relevant location updates (without traversing the spatial network index) and discard the updates that will not affect the query result. Below, we explain the two-phase ER-C*k*NN algorithm.

4.1   Off-line Grid Partitioning

With the off-line phase, we partition the spatial span of the network with regular grid as illustrated in Figure 2(a). Each grid cell is a square of size $\alpha \times \alpha$ (in Section 5.1 we explain how we choose the optimal size) denoted by its row and column indices $c(i, j)$, where the reference cell $c(0, 0)$ is the bottom-left cell of the grid. The resulting grid partitioning, yields following two main advantages.

Firstly, such network partitioning enables ER-C*k*NN to expedite on-line network distance computations using precomputed information. Specifically, ER-C*k*NN, for each edge, maintains a vector $\overrightarrow{v}_{EBE}$ (proposed by Lauther in [6]) that contains encoded values indicating whether the edge is a part of a shortest path to a given grid cell. ER-C*k*NN utilizes $\overrightarrow{v}_{EBE}$ to avoid exploring unnecessary paths (hence pruning the search space) during an on-line shortest path computation. For example, Figure 2(a) illustrates a simple road network (partitioned to nine regions) where the $\overrightarrow{v}_{EBE}$ of edge $e(n_2, n_3)$ only contains three 1 entries which correspond to $c(0, 0)$, $c(1, 0)$, and $c(1, 1)$ cells (marked by the shaded area). This means that edge $e(n_2, n_3)$ can be a part of a shortest path to any node in those regions. Considering a shortest

path search from n_1 with target nodes (e.g., n_{10}) in unmarked cells, the search ignores edge $e(n_2, n_3)$ during the on-line computation.
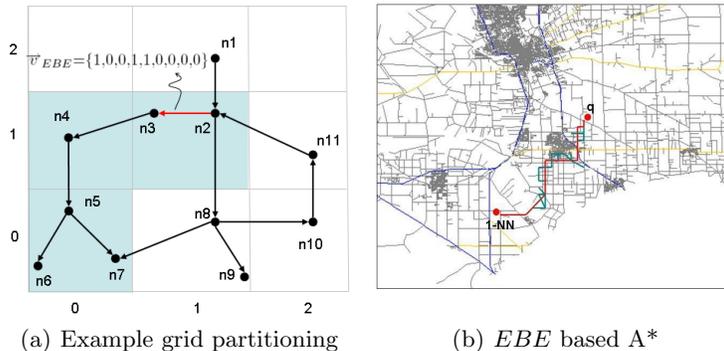

(a) Example grid partitioning   (b) EBE based A*

Fig. 2. EBE based shortest path computation

In order to determine the encoded values (i.e., 1 or 0) contained in $\overrightarrow{v}_{EBE}$ of an edge $e(n_s, n_t)$, we compute a one-to-all shortest path from the head node n_s to all other nodes in the network. If *any* node n_u is reached in a grid cell $c(i, j)$, we set the encoding information to 1 (i.e., true) for the region containing node n_u. We refer to this operation *edge-bitmap-encoding* (EBE) and repeat it for each edge. The integration of $\overrightarrow{v}_{EBE}$ to any shortest path algorithm is very easy. Specifically, any shortest path algorithm can be modified to check the encoded value of the corresponding grid cell (that contains the target node) ever time before traversing an edge (If true, the edge is traversed). With ER-CkNN, we integrate $\overrightarrow{v}_{EBE}$ to A* algorithm (referred as EBE-based A*). This integration further improves the performance of A* algorithm thus minimizing the redundant network traversal. Recall that A* is already much faster than Dijkstra for point-to-point shortest path computations (using Euclidean distance heuristic between the points). We refer readers to [11] for the details of A* algorithm and the comparison of it to Dijkstra. Continuing with our example presented in Figure 1, Figure 2(b) shows the set of edges (highlighted around the actual shortest path) that ER-CkNN would traverse to locate the first nearest data object using EBE-based A*. As shown, EBE-based A* algorithm visits significantly less number of network nodes.

The storage requirement of $\overrightarrow{v}_{EBE}$ is extremely low as its size is bounded by the number of grid cells. The space complexity is $O(RE)$ for a network with R regions and E edges. To imagine, the space required to maintain the EBE information of Los Angeles spatial network (with 304,162 edges) divided to 128X128 grid cells is around 20 mega bytes. Note that only one bit value is stored for each region. In addition, the EBE precomputation is not affected from the dynamic edge weights as it is based on the topology of the network. If the network topology changes (less likely), the $\overrightarrow{v}_{EBE}$ of the edges should be updated.

Secondly, grid partitioning enables ER-CkNN to efficiently manage the object location updates (hence continuous monitoring of the query results). In particular, ER-CkNN maintains an in-memory grid index in which each cell contains a list of the objects currently residing in the cell. Given a moving object location $loc_t(p) =$

(x_p, y_p), $c(\lfloor \frac{x_p}{\alpha} \rfloor, \lfloor \frac{y_p}{\alpha} \rfloor)$ is the grid cell containing the p. In order to relate the grid index with the network edges (indexed by memory based PMR QuadTree [2]) thus enabling spatial network query processing, ER-CkNN associates each network edge with the overlapping grid cells (*edge-cell-mapping*). This information is stored in an hash table. For example, the edge $e(n_8, n_{10})$ (in Figure 2(a)) is mapped to the cells $\{c(0, 1), c(0, 2)\}$. We will describe the use of edge-cell-mapping more in Section 4.4.

4.2 On-line Query Processing

In this section, we first explain how ER-CkNN leverages ER on the grid-partitioned spatial network and implements a filter-and-refine process to generate the initial query results. Next, we discuss how ER-CkNN continuously maintains the query result as new object location updates are received and/or the weights of the network edges change.

4.3 Generating Initial Query Result

ER-CkNN computes the initial result of a query using a filter-and-refine approach. With the filter step, first ER-CkNN performs a grid search to quickly find a set of candidate nearest neighbors based on their Euclidean distance from q. Next, by exploiting the fact that Euclidean distance is a lower-bound for network distance, ER-CkNN uses ER to extend the original candidate to a *super-set* which contains actual nearest neighbors of q. Toward that end, ER-CkNN 1) computes the maximum network distance NDT (short for *Network Distance Threshold*) from q to any of the objects in the original candidate set, and 2) performs a range query with radius NDT to identify all objects (and corresponding edges) that comprise the super set. The super-set contains actual k nearest neighbors for q on the spatial network; hence, no *false misses*. At the refine step, the super set is further refined by removing possible *false hits*, and ER-CkNN returns the top k objects with minimum network distance from q.

4.3.1 Filter Step When a query object initiates a kNN search, the first step is to perform a grid expansion to identify the k nearest neighbors in the Euclidean proximity. Figure 3 illustrates an example where the goal is to find $k = 2$ nearest neighbor for the query object q. Referring to $q.cell$, we first check the grid cell in which the q resides. Since there is not any potential neighbors in this cell (see Figure 3(a)), the search moves to the next level as illustrated in Figure 3(b). Here we find the two nearest neighbor, namely p_1 and p_2 and, hence, the grid search is stopped. Note that with the grid search we only retrieve the object list from each grid cell without traversing the underlying spatial network. Having found the candidate set, next we move on to compute the super set that contains actual nearest neighbors. Toward that end, we first compute the respective network distances (using EBE- based A*) of the objects in the candidate set $d(q, p1) = 10$, $d(q, p2) = 6$, and correspondingly update $NDT{=}10$ (Algorithm 1 Line 3). Next, ER-CkNN performs a range query on the spatial network (using PMR quadtree) with q as the center and NDT as the radius (Line 4-5). With this operation, ER-CkNN retrieves

8

the *active-edges q.activeEdges* (the edges that are within the shaded area in Figure 4(a)) as well as m ($\{p_1, p_2, p_3, p_4\}$) objects that comprise the super set. The crucial observation in this step that If $m = k$ (i.e., there are no false hits), the exact set of k nearest neighbors for q are found (Line 7). Our experiments show that in 68% of the cases $m = k$. This implies that ER-CkNN finds kNN with only a simple grid search and the least number of network distance computations thus incurring fast response time and low computation costs.
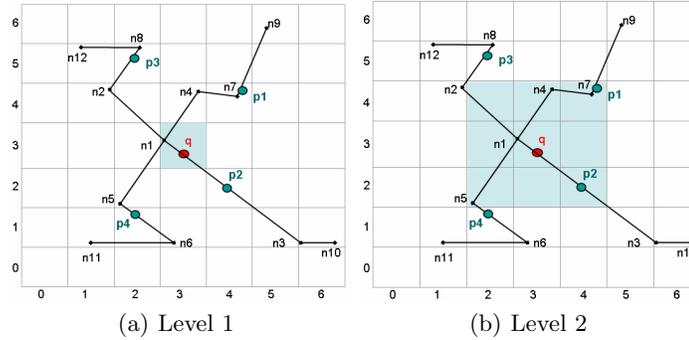

(a) Level 1   (b) Level 2

Fig. 3. Grid search for 2NN query

At this point, it is important to clarify that ER-CkNN, even with the large values of k where the shortest path executed k times and some edges may be traversed more than once, visits less nodes than the network expansion methods (see Section 6.2.3 for experimental results). This is due to the following reasons. First, ER-CkNN performs *EBE*-based A* algorithm that enables extensive pruning (to almost the linear function of the shortest path) hence minimizing the invocation of the costly network data access. Second, ER-CkNN computes the shortest path for only feasible data objects. If the corresponding value of a grid cell that contains one or more data objects is false in the $\overrightarrow{v}_{EBE}$ of the query object's edge, ER-CkNN does not attempt to compute the network distance to those objects at all. Finally, it is possible to reuse some network distance computations as the shortest path from multiple queries to some target data objects might overlap. Consider a query point q looking for data object p which was previously found by q'. If an on-line shortest path computation, from q to p, reaches q' during the scan, then there is no need to continue the computation as ER-CkNN already knows the shortest path from q' to p.

4.3.2 Refine Step Once at the refine step, we have $m > k$. We denote the set of $m - k$ objects that were added to the candidate set set as P' (in Figure 4(a), $P' = \{p_3, p_4\}$). To find the actual nearest neighbors, ER-CkNN applies ER until $m = k$. Specifically, ER-CkNN performs consecutive range queries with $NDT = d_N(p_j, q)$, where p_j is the nearest Euclidean neighbor of q among P' objects, until $m = k$ (Line 11-20). To illustrate, we continue with our running example from Section 4.3.1. Since the active-edges obtained at this step contain $m = 4$ ($m > k$) data objects, ER-CkNN proceeds to refine step, where it computes the network distance to p_4 in P' and compares $d_N(p_4, q) = 8$ with NDT. As $d_N(p_4, q) < NDT$ (if $d_N(p_j, q) > NDT$ the algorithm investigates the remaining objects in P'), the new object p_4 becomes the current k-th nearest neighbor and NDT is updated
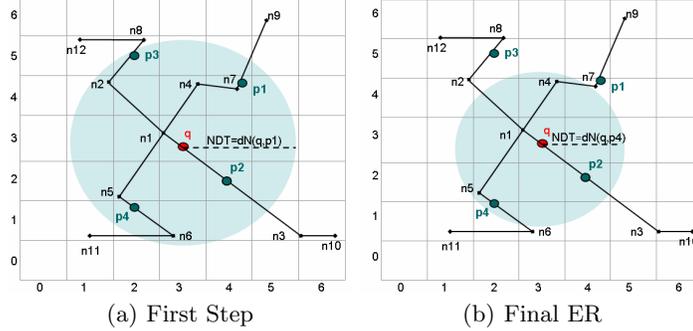
(a) First Step   (b) Final ER

Fig. 4. Using ER (Euclidean Restriction) with ER-CkNN

Algorithm 1 ER-CkNN Algorithm

1: /* k:number of NNs, q:moving query object, p_n: network distance */
2: $[p_k] = searchGrid[q.cell, k]$ /* returns k objects from the grid search*/
3: $q.NDT = max(pnd_1, ..pnd_k)$
4: $edgeList = euclideanRangeQuery(q.loc, q.NDT)$
5: $candidateNNs = retreiveObjects(edgeList)$
6: $m = candidateNNs.length$
7: **if** $(m = k)$ **then** $q.resultSet = [p_k]$; **break;**
8: **else**
9: $j = 1$
10: repeat
11: $p_j = nextEuclideanNN(q)$ /* find closest dataobject to q */
12: **if** $d_N(q, p_j) < NDT$ **then**
13:    $updateResultSet(p_j)$ /*insert p_i and remove kth NN from $q.resultSet$*/
14:    $NDT = d_N(q, p_j)$
15:    $edgeList = euclideanRangeQuery(q.loc, q.NDT)$
16:    $candidateNNs = retreiveObjects(edgeList)$
17:    $m = candidateNNs.length$
18: **end if**
19: $j++$
20: until m=k
21: $updateActiveEdges(edgeList)$
22: **end if**

---

to $d_N(p_4) = 8$. Later, ER-CkNN performs the second range query (with the new NDT) which excludes p_1 and p_3 (see Figure 4(b)). At this point, since there are only two remaining objects in the candidate set (i.e., p_2 and p_4), the refine step terminates by returning p_2 and p_4 as the final result and updating $e.activeEdges$ with the final set of active-edges which are within the shaded area in Figure 4(b) (i.e., $\{e(n_1, n_2), e(n_1, n_3), e(n_1, n_4), e(n_4, n_7), e(n_1, n_5), e(n_5, n_6)\}$). As shown, p_3 is automatically excluded after the second ER. Note that finding objects on the active-edges is in linear time. After finding the active- edges, ER-CkNN first determines the corresponding set of overlapping grid cells using edge-cell-mapping hash table, and then retrieves the objects (in O(1)) from the grid file.

4.4 Continuous Maintenance of Query Result

In this section, we explain how ER-CkNN continuously maintains the query result while data and query objects move and the weights of the network edges change. As noted before, ER-CkNN benefits from the grid partitioning to determine the relevancy of the incoming location updates. Specifically, ER-CkNN identifies the active-cells (that overlaps with the active-edges found in the refine step) around the q. If a location update is mapped to one of these active-cells, it is considered as relevant update and processed; otherwise it is discarded. Below, we first classify the location updates and explain each of the classes in isolation; thereafter, we discuss how ER-CkNN implements all cases simultaneously.

4.4.1 Data and Query Object Location Updates With ER-CkNN, we classify each object location update into following three categories based on the type of the movement: *resident updates*, *inbound updates*, and *outbound updates*. The resident updates are those which indicate an object has been resident among the active-edges of the query since last update, either staying still or moving from one active-edge to another. The inbound updates report movement of the objects that used to reside on an inactive edge but have moved to an active-edge since then. Finally, the outbound updates report movement of the objects that have moved from an active-edge to an inactive edge.

When a data object p sends an update containing its old and new location to the central server, ER-CkNN first updates the grid index with the new location of p by identifying the new grid cell. Next, ER-CkNN checks if the location update falls in to grid cells (*active-cells*) that overlap with the active-edges of a query object. Recall that the active-edges (thus active-cells) of the query objects are already found in the refinement step. The crucial observation here is that only the relevant location updates that are mapped to the active-cells can affect the result set of the queries (*localized mapping*); hence the irrelevant updates are efficiently identified and discarded. The cost to identify whether the location update is relevant or not is in O(1) as ER-CkNN only checks the flag (indicating active or not) of the grid cell that the new location update falls in. For example, assume that p_3 sends a location update which falls into cell $c(6, 5)$ in Figure 5(a). Since neither the new nor the old location of p_3 are mapped to the active-cells (marked cells in the Figure 5(a)) which overlap with the active-edges, the movement of p_3 is ignored.

If relevant updates are received, ER-CkNN updates the results of the queries by considering the update categories. Specifically, ER-CkNN considers two cases. The first case is when the number of outbound updates is less than or equal to the number of inbound updates. In this case there still exist at least k objects on the active-edges. Therefore, ER-CkNN first removes the outbound nearest neighbors from the result set and then merges the objects that are in resident and inbound update categories by returning the k nearest objects among them. Note that if the NDT is decreased after this operation, ER-CkNN updates the NDT and $q.activeEdges$ accordingly. For instance, assume that p_1, p_2, and p_3 send location updates, and p_4 stays still as show in Figure 5(b). In this example, the location updates of p_1, p_2, and p_4 are categorized as inbound, outbound, and resident, respectively (the movement of p_3 is ignored as described before). Since the number of outbound updates is equal to the

number of inbound updates, ER-CkNN removes p_2 from the result set and adds p_1 to it. ER-CkNN avoids network distance computation to p_1 since p_1 resides on the shortest path of q to p_2 that was previously computed. Finally, ER-CkNN returns p_1 and p_4 as the final result set. The second case is the number of outbound updates is more than the number of inbound updates. In this case ER-CkNN needs to expand the search space to retrieve the new results since there are less than k objects on the active-edges. This is achieved by grid expansion as described before until k objects found. To avoid recomputation, the grid expansion, instead of starting from the cell where q resides, starts from the level of the k th data object that is furthest to q in Euclidean distance.

(a) Location mapping   (b) Object location update   (c) Query location update
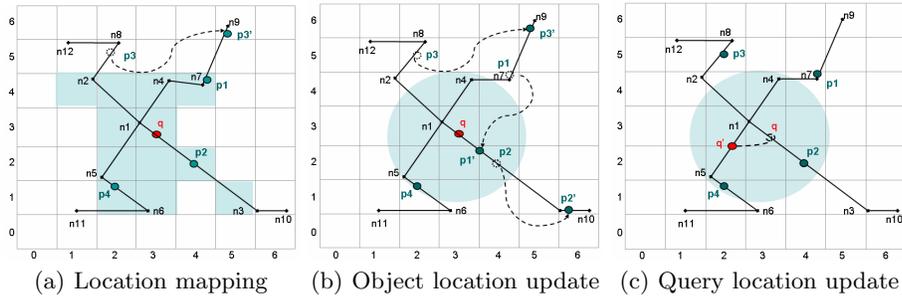
Fig. 5. Data and query object location update

Similar to data objects the query objects move on or out of the active-edges. If q moves on any of the active-edges, we save some computation. To illustrate, consider Figure 5(c) where q moves to a new location q'. Since the shortest path from p_4 to q was originally computed, we can easily compute network distance from q' to p_4. To retrieve the new results of the q, ER-CkNN continues from the Line 11 of the Algorithm 1. In the case q moves to an in-active edge, ER-CkNN computes the new k nearest neighbors and the active-edges from the beginning.

When an edge weight change is received, ER-CkNN first checks if the edge update corresponds to an active-edge. In case the new edge weight has increased, implying that the network distance between q and the data object p_i is increased, there may exist shorter alternative paths to reach p_i in the network. Therefore, ER-CkNN employs the filter step and continues from the Line 12 of the Algorithm 1. If the new edge weight has decreased and the weight change affects the current NDT, ER-CkNN updates the NDT among the data objects in the result set and the $q.activeEdges$.

When ER-CkNN receives object location and edge weight updates simultaneously, it first checks if the q moves to an inactive edge. If so, ER-CkNN recomputes the kNN from the beginning by discarding the data object and edge updates. Otherwise, ER-CkNN first processes the edge updates since handling the query update before considering the edge update may result in retrieving wrong active-edges. After finishing the edge updates, ER-CkNN processes the query movement. Finally, it handles data object updates based on the finalized active-edges from the previous two updates.

5 Discussion

In this section we discuss the two main factors that affect the performance of ER-CkNN, namely the grid granularity and the network topology. We describe the details of these two factors and explain how we optimize the grid granularity.

5.1 Grid Granularity

As grid size grows (having less cells), the total number of cell retrievals decreases. However, large cells result in retrieving more excessive (unneeded) objects. On the other hand, with small grid cells, the cost of grid cell retrieval, and the memory requirements of EBE increases. We derive the optimal value of the grid size with the following theorem.

Lemma 1. *Let the moving objects were represented as points distributed on the unit square $[0, 1] \times [0, 1]$ partitioned by the cell (which contains P objects) size $\alpha \times \alpha$. The number of cells in a region $(L \times L)$ (L is the edge length of the square after grid expansion) is given by $\frac{\lfloor (L+\alpha) \rfloor^2}{\alpha^2}$ and the number of objects in this region is approximately $(\lfloor (L+\alpha) \rfloor^2)P$ under uniform distribution.*

Theorem 1 *For a gird of size $\alpha \times \alpha$ which contains P objects, $\alpha \approx \frac{1}{\sqrt{P}}$ is the optimal grid size to minimize the query time under the assumption of uniform distribution.*

Proof. The overall query processing time T is dominated by the grid expansion (i.e., time required to query several grid cells which contain k objects) and processing the objects located in the grid cells (i.e., time required to compute network distance). Therefore $T = t_g n_g + t_o n_o$ where n_g and n_o represents number of grid cells and number of objects, respectively (t_g and t_o are constants). Computing the region for a kNN query involves finding the circle $C(o; r)$ ($r = d_{NDT}$ of kth object in the worst case) which includes the kth nearest object to the query point. Therefore, using the lemma from above, the number of objects (i.e., k) in this region is $k \approx \pi r^2 * P$ so $r \approx \sqrt{\frac{k}{\pi P}}$ and the number of grid cells contained in the area bounding this circle $n_g = \frac{(2r+\alpha)^2}{\alpha^2}$ and $n_o \approx (2r+\alpha)^2 P$. Replacing n_g and n_o in $T = t_g n_g + t_o n_o$ with the above values and setting $\frac{\partial T}{\partial \alpha} = 0$ gives $\alpha^3 = \frac{t_g r}{t_o P}$ or $\alpha = \sqrt[3]{\frac{t_g}{t_o} \sqrt{\frac{k}{\pi}} \frac{1}{\sqrt{P}}}$. For $k \ll n$, the α can be simplified to $\alpha \approx \frac{1}{\sqrt{P}}$

5.2 Network Topology

In general, the topology of a network may affect the performance of spatial network query processing methods. One topology concern when processing ER based queries in spatial networks is network and Euclidean distance correlation (referred as NEC). Because, ER based methods rely on lower bound restriction that yields better results when NEC is high (i.e., the network distance is close to the Euclidean distance between two points). The experimental evaluations show that the response

time of ER-CkNN is not severely affected from NEC. The average response time loss of ER-CkNN is approximately %12 when the NEC between the query and data objects is low. We used two different datasets (i.e., Los Angeles and San Joaquin networks) to show that the behavior of ER-CkNN does not change significantly with the different network topologies. The NEC is high with Los Angeles network where as it is low with San Joaquin network.

On the other hand, ER-CkNN provides a very effective way to handle directional queries. For example, if a query asks for the nearest gas stations towards North (i.e., the driving direction), ER-CkNN only expands the grid search towards that direction thus pruning the search space to half. Directional queries are commonly used in car navigational systems.

6 Experimental Evaluation

6.1 Experimental Setup

We conducted experiments with different spatial networks and various parameters (see Table 1) to evaluate the performance of ER-CkNN and compare it with GMA [9] (as GMA yields better performance results than IMA). With our experiments, we measured the impact of the data object and query cardinality, data object and query distribution, network size, gird size, k and data object agility and speed. As our dataset, we used Los Angeles (LA) and San Joaquin County (SJ) road network data with 304,162 and 24,123 road segments, respectively. Both of these datasets fit in the memory of a typical machine with 4GB of memory space. We obtained these datasets from TIGER/Line (http://www.census.gov/geo/www/). Since the experimental results with these two networks differ insignificantly (approximately %12) as noted in Section 5.2 and due to space limitations, we only present the results from the LA dataset. We generated the parameters represented in Table 1 using a simple simulator prototype developed in Java. We conducted our experiments on a workstation with 2.7 GHz Pentium Core Duo processor and 8GB RAM memory. We continuously monitored each query for 50 timestamps. For each set of experiments, we only vary one parameter and fix the remaining to the default values in Table 1.

Table 1. Experimental parameters

Parameters	Default	Range
Number of objects	15 (K)	1,5,10,15,20,25 (K)
Number of queries	3 (K)	1,2,3,4,5 (K)
Number of k	20	1,10,20,30,40,50
Object Distribution	Uniform	Uniform, Gaussian
Query Distribution	Uniform	Uniform, Gaussian
Object Agility a	10 %	0, 5, 10, 15, 20,25 (%)
Object Speed v	60 kmph	20, 40, 60, 80, 100 (kmph)

6.2 Results

6.2.1 Impact of Object and Query Cardinality First, we compare the performance of two algorithms by varying the cardinality of the data objects (P) from

1K to 25K while using default settings (see Table 1) for all other parameters. Figure 6(a) illustrates the impact of data object cardinality on response time. The results indicate that the response time linearly increases with the number of data objects in both methods where ER-CkNN outperforms GMA with all numbers of data objects. From P=1K to 5K, the performance gap is more significant where ER-CkNN outperforms GMA by factor of four. Because, when P is less, the data objects are distributed sparsely on the network which causes GMA to incrementally expand the search area by visiting unnecessary edges and nodes. When P is large, GMA, with each location update, traverses the spatial index to identify the edge of the moving objects. However, ER-CkNN needs to identify the edge of the moving object only if the location update of it falls in to localized grid cells. Figure 6(b) shows the impact of the query objects (Q) ranging from 1K to 5K on response time. As shown, ER-CkNN scales better with large number of queries and the performance gap between the approaches increases quadratically as Q grows. Because, in addition to factors mentioned above, GMA maintains an expansion tree (for monitoring) which is invalidated frequently with the continuous location updates of the q thus yielding very high maintenance costs.
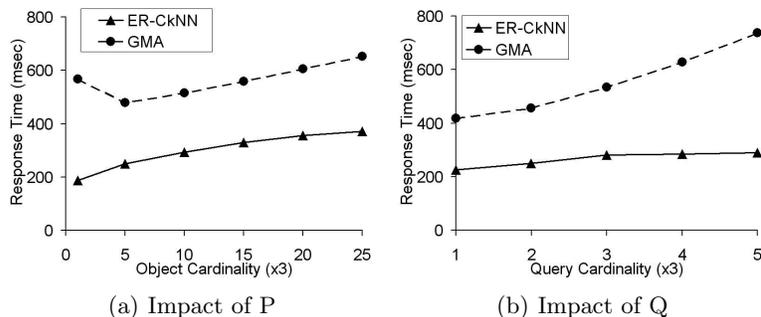

(a) Impact of P   (b) Impact of Q

Fig. 6. Response time versus P and Q

6.2.2   Impact of Object/Query Distribution and Network Size With this experiment set, we study the impact of object, query distribution and network size on ER-CkNN. Figure 7(a) shows the response time of both algorithms where the objects and queries follow either uniform or Gaussian distributions. As illustrated, ER-CkNN outperforms GMA significantly in all cases. ER-CkNN yields better performance for queries with Gaussian distribution. Because, as queries are clustered in the spatial network with Gaussian distribution, their active-cells would overlap hence allowing ER-CkNN to monitor relatively less active-cells. Furthermore, since the shortest paths from clustered queries to some data objects would overlap, ER-CkNN benefits from reusing numerous network distance computations. Apart from evaluating the response time with different distributions, in each set of experiments, we also measured our success rate with finding k nearest neighbors by only performing filter step. In average 68% of the cases ER-CkNN was able to find the k nearest neighbors without executing the refinement step as described in Section 4.3.1.

In order to evaluate the impact of network size, we conducted experiments with the sub-networks of LA dataset ranging from 50K to 250K segments. Figure 7(b)

illustrates the response time of both algorithms with different network sizes. In general, with the default parameters in the Table 1, the response time increases for both algorithms as the network size increases. With this experiment, we observed that the EBE-based shortest path computation yields better speed-up factors with the increasing network size.
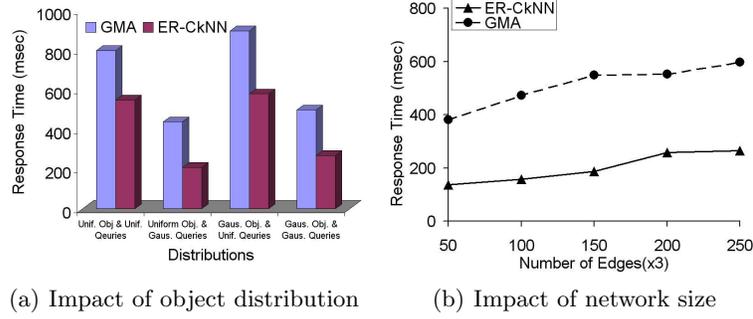

(a) Impact of object distribution   (b) Impact of network size

Fig. 7. Response time versus N and Q

6.2.3 Impact of k With another experiment, we compare the performance of the two algorithms with regard to k. Figure 8(a) plots the average query efficiency versus k ranging from 1 to 50. The results indicate that ER-CkNN outperforms GMA with all values of k and scales better with both small and the large values of k. Because, when k is small ER-CkNN benefits from the directional search. As k increases the search space of GMA incrementally expands in all directions hence incurring redundant node traversal and the expansion tree (see [9] for the expansion tree) grows exponentially by holding more unnecessary network data hence yielding extensive maintenance and access cost. As illustrated, ER-CkNN outperforms GMA by at least a factor of three for $k{\geq}10$. In addition, we compared the average number of node access with both algorithms. As shown in Figure 8(b), the number of nodes accessed by ER-CkNN is less than GMA with all k values. This is because ER-CkNN, rather than expanding the search blindly, utilizes the point-to-point EBE-based A* that minimizes the node access in the network and reuses more network distance computations as the shortest paths to some data objects overlap more.
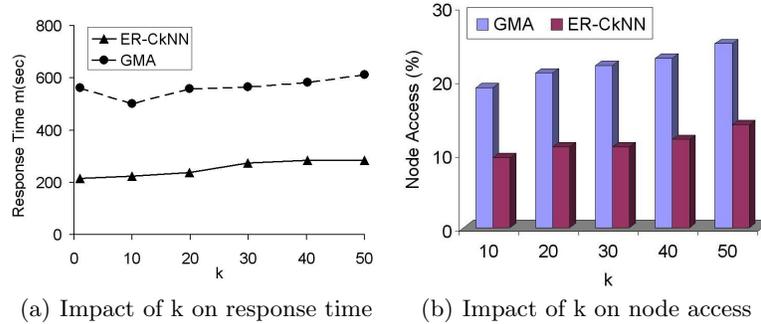

(a) Impact of k on response time   (b) Impact of k on node access

Fig. 8. Response time and node access versus k

16

6.2.4 Impact of Object Agility and Speed With this set of experiments, we evaluate the performance of both algorithms with respect to object movements. We use two parameters to measure the affect of object movements namely object agility a and object speed v. Object agility indicates the percentage of objects that move per timestamp (0% represents static data) while the object speed indicates object's speed measured by kilometer per hour. Figure 9(a) illustrate the impact of object agility ranging from 0% to 25%. As the object agility grows, both algorithms query processing time increases slightly due to the frequent updates in the number of inbound and outbound objects. The superior performance of ER-CkNN approach is due to the usage of localized mapping that avoids extensive invalidations with the expansion tree of GMA and unnecessary node and edge access. As Figure 9(b) indicates, both algorithms are unaffected by the object speed because the focus of both algorithms only concern whether there are object updates (i.e., inbound, outbound) that may invalidate the existing results in the monitoring areas rather than how fast the objects move in or out of the monitoring areas.
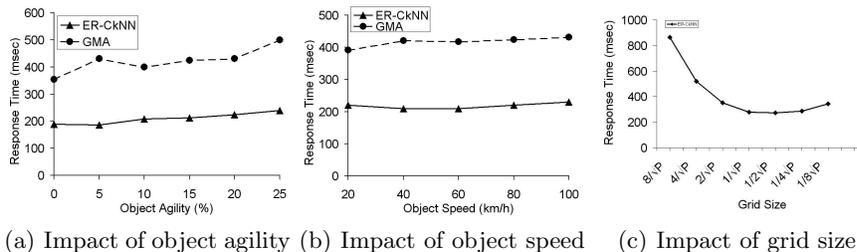

(a) Impact of object agility (b) Impact of object speed (c) Impact of grid size

Fig. 9. Response time versus object agility, speed and grid size

6.2.5 Impact of Grid Size In order to compare the theocratical results from the analysis in Section 5.1 and evaluate the the impact of the grid size on ER-CkNN, we run several experiments with different grid sizes and the default values in Table 1. Figure 9(c) illustrates the response time needed with grid sizes ranging from $\frac{8}{\sqrt{P}}$ to $\frac{1}{8\sqrt{P}}$. As illustrated, decreasing the cell size has the effect of reducing the response time. There is a substantial increase in performance as we move from $\frac{8}{\sqrt{P}}$ to $\frac{1}{\sqrt{P}}$, but later the response time starts increasing again at finer granularity. This validates the analytical results. Note that memory requirements of ER-CkNN is slightly more than IMA/GMA because of the additional grid indexing.

7 Conclusion

With this paper, we proposed an Euclidean restriction based algorithm which avoids the blind network expansion and blind object location mapping shortcomings of the network expansion methods. The key benefit of ER-CkNN is that it enables guided shortest path search that minimizes the redundant node access and localizes the network that facilitates the continuous kNN monitoring. ER-CkNN does not make any simplifying assumption (e.g., static data objects, known trajectories) about the

moving objects and edge weights. Therefore, it can easily be applied to real-world road network kNN applications. As a future work, we intend to extend this study to handle different monitoring queries such as range and reverse kNN in spatial networks.

References

1. H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. In *VLDB*, 2005.
2. E. G. Hoel and H. Samet. Efficient processing of spatial queries in line segment databases. In *SSD*, 1991.
3. D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch. Main memory evaluation of monitoring queries over moving objects. In *DPDB*, 2004.
4. M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, 2004.
5. M. R. Kolahdouzan and C. Shahabi. Continuous k-nearest neighbor queries in spatial network databases. In *STDBM*, 2004.
6. U. Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In *Geoinformation and Mobilitat*, 2004.
7. M. F. Mokbel, X. Xiong, and W. G. Aref. Sina: scalable incremental processing of continuous queries in spatio-temporal databases. In *SIGMOD*, 2004.
8. K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In *SIGMOD*, 2005.
9. K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, 2006.
10. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, 2003.
11. S. J. Russell and P. Norvig. *Artificial intelligence: A modern approach*. Prentice-Hall, Inc., 1995.
12. H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, 2008.
13. C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh. A road network embedding technique for k-nn search in moving object databases. *Geoinformatica*, 2003.
14. Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. In *SSTD*, 2001.
15. Y. Tao and D. Papadias. Time-parameterized queries in spatio-temporal databases. In *SIGMOD*, 2002.
16. Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, 2002.
17. H. X., J. C.S., H. L., and S. Saltenis. S-grid: A versatile approach to efficient query processing in spatial networks. In *SSTD*, 2007.
18. H. X., J. C.S., and S. Saltenis. The island approach to nearest neighbor querying in spatial networks. In *SSTD*, 2005.
19. X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, 2005.
20. X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE*, 2005.
21. J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based spatial queries. In *SIGMOD*, 2003.
22. B. Zheng and D. L. Lee. Semantic caching in location-dependent query processing. In *SSTD*, 2001.