A Novel Dual-Index Design to Efficiently Support Snapshot Location-Based Query Processing in Mobile Environments

Haojun Wang, *Student Member*, *IEEE*, and Roger Zimmermann, *Senior Member*, *IEEE*

Abstract—Location-based services are increasingly popular recently. Many applications aim to support a large number of users in metro area (i.e., dense networks). To cope with this challenge, we present a framework that supports location-based services on MOVing objects in road Networks (MOVNet, for short) [26]. MOVNet's dual-index design utilizes an on-disk R-tree to store the network connectivities and an in-memory grid structure to maintain moving object position updates. In this paper, we extend the functionality of MOVNet to support snapshot range queries as well as snapshot k nearest neighbor queries. Given an arbitrary edge in the space, we analyze the minimum and maximum number of grid cells that are possibly affected. We show that the maximum bound can be used in snapshot range query processing to prune the search space. We demonstrate via theoretical analysis and experimental results that MOVNet yields excellent performance with various networks while scaling to a very large number of moving objects.

Index Terms—Spatial databases, GIS, Location-dependent, sensitive.

1 Introduction

With the widespread use of GPS devices, more and more people are enjoying location-based services. Various applications, such as roadside assistance, highway patrol, and location-aware games, are popular in many urban areas. This has intensified research interests to overcome the inherent challenges in designing scalable and efficient infrastructures to support very large numbers of users concurrently. The mobility made possible by the usage of car-based or handheld GPS devices in metro cities results in two fundamental system requirements: distance computations within a (road) network and processing of moving Points of Interest (POIs).

An increasing number of applications require query processing of moving POIs based on an underlying network. For example, when a pedestrian calls for emergency assistance, the call center may want to locate all police cars within a five-mile distance and dispatch them to the call-originating location. Note that the mentioned examples require snapshot queries, rather than continuous monitoring (which is another class of applications).

Spatial data processing is a very active research field. Some of the early work introduced spatial processing of stationary objects based on euclidean distance metrics. More recent work incorporates POI mobility or network-distance processing, but often not both. Several techniques [4], [14], [21] have aimed at solving location-based (Continuous) k Nearest Neighbor (kNN/C-kNN) queries in spatial networks. These methods assume that the positions of POIs are fixed (e.g., gas stations or bus stops). While the ability to process moving POIs is challenging, it also enables new applications and, in the most general case, the query points and the POIs are interchangeable, i.e., users or vehicles equipped with GPS devices are able to report their positions and, hence, themselves become POIs. Some prior work has focused on providing the functionality for moving POI processing [19], [28], [30]. Specifically, these techniques aim to continuously monitor a set of moving nearest neighbors. However, one of the limitations of these methods is their reliance on euclidean distance measures, which can be imprecise, especially in dense road networks. Two of the main challenges when supporting POI mobility on an underlying road network are to 1) efficiently manage object location updates and 2) provide fast network distance computations. To address these issues, we have designed a novel system to process location-based queries on MOVing objects in road Networks (MOVNet) [26]. The goal is to efficiently execute snapshot range and k nearest neighbor queries over moving POIs within a stationary road network. Although MOVNet is not aimed at continuous query processing in its current form, we believe that a large number of location-based services only require snapshot query processing capabilities. For instance, when a user calls a service center to find a nearby taxi, the query is instantaneous and as soon as a taxi is dispatched to pick up the customer, the transaction is complete (i.e., the ordering phase).

Fig. 1 illustrates MOVNets system infrastructure and components. To handle large networks, MOVNet utilizes an on-disk R*-tree [1] structure to store the necessary connectivity information. Efficient processing of moving object position updates is achieved with an in-memory grid index. An appealing feature of MOVNet is the bidirectional

- *H. Wang is with the Department of Computer Science, University of Southern California, 941 Bloom Walk, Los Angeles, CA 90089. E-mail: haojunwa@usc.edu.*
- *R. Zimmermann is with the School of Computing, Department of Computer Science, National University of Singapore, Computing 1, 13 Computing Drive, Singapore 117417. E-mail: rogerz@comp.nus.edu.sg.*

Manuscript received 16 Nov. 2008; revised 17 Dec. 2009; accepted 28 Feb. 2010; published online 14 Apr. 2010.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2008-11-0459. Digital Object Identifier no. 10.1109/TMC.2010.63.

1536-1233/10/$26.00 © 2010 IEEE Published by the IEEE CS, CASS, ComSoc, IES, & SPS

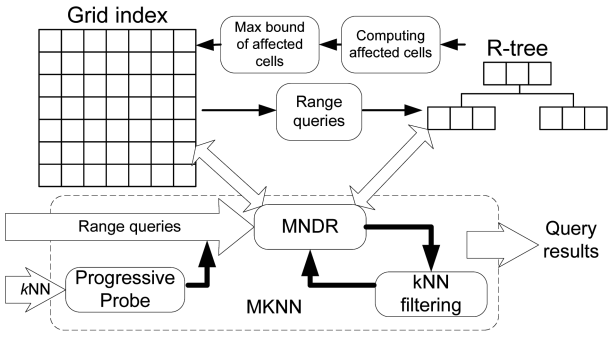

Fig. 1. The index structures and query processing modules of MOVNet.

mapping between the two structures that enables the retrieval of a minimal set of data for query processing. Based on the concept of *affected cells* that form the set of grid cells overlapping with a given edge, we present algorithms to execute range as well as kNN queries. Analytical bounds on the minimum and maximum number of affected cells with an arbitrary network edge enable the pruning of the search space during mobile range query processing. In the mobile kNN query algorithm, we utilize the concept of a *progressive probe* into the grid index to estimate the subspace containing the result set. The performance of our design has been verified vigorously through theoretical analysis and simulations. Our comparison with two state-of-the-art baseline algorithms demonstrate the superior performance of MOVNet.

The remainder of this paper is organized as follows: Section 2 describes the related work. Section 3 discusses our assumptions and the dual-index design. In the following Section 4, we propose our mobile network distance range query and k nearest neighbor query algorithms. We present the theoretical analysis of our design in Section 5. We vigorously verify the performance of MOVNet and demonstrate that the results match our analysis in Section 6. Finally, we conclude with Section 7.

2 Related Work

Processing spatial queries in networks has been intensively studied recently. Papadias et al. [21] first presented an architecture that integrates network and euclidean information in processing network-based queries. Specifically, the idea of *euclidean restriction* utilizes the property that for any two objects in the network, the network distance is at least the same as the euclidean distance. In contrast, the *network expansion* method performs the search directly from the query point by expanding the nearby vertices in the order of their distances from the query point. As an improvement, the VN^3 method [15] was proposed as a Voronoi-based approach to precompute the distances within and across subspaces. The goal was to avoid online distance computations in processing kNN queries. Huang et al. [11] addressed the same problem by proposing the *islands approach* that estimates the overhead of precomputation and the trade-off between query and update performance for kNN queries with various densities of POIs and networks. To cope with C-kNN queries on stationary POIs in a network, Kolahdouzan and Shahabi [14] proposed the Intersection Examination and Upper Bound Algorithm (IE/UBA) to compute the kNN objects of all nodes on the path and the *split points* between adjacent nodes whose nearest neighbors are different. Lately, Cho and Chung [4] solved the same problem by introducing UNICONS which incorporates precomputed kNN lists into Dijkstra's algorithm such that it outperforms the IE/UBA approach in dense networks.

The preceding techniques hold the assumption that the POIs are static (i.e., the POIs do not update their location). However, a large number of the spatial applications require the capability to process moving POIs. This requirement raises the issue of managing a very large number of location updates of moving POIs in an index structure. To overcome this challenge, using predictions (i.e., the trajectory of moving objects) to presume the movement of objects have been used in R-tree-based structures (e.g., the TPR-Tree and its variants [24], [23]) and B-tree-based structures (e.g., the B^x tree [13]). As an alternative, STRIPE [22] introduces the idea of transforming the trajectories of objects in a d-dimensional space into points in a 2D space. However, the assumption of being able to predict the trajectories of moving objects is not always realistic. If the forecast of the object movements fails (e.g., pedestrian strolling in a shopping mall), these approaches are inappropriate. Hence, the Lazy-update R-tree (LUR-tree) [16] and its extension [17] modify the original R-tree design to support frequent updates with no restriction on object movement. Specifically, the periodical sampling of the object location is recorded in these tree structures to represent the current location of POIs.

In general, tree-based indices suffer from node reconstruction overhead when performing location updates. Therefore, grid-based structures have raised intensive interest due to their simplicity and efficiency in indexing moving objects. Specifically, Xiong et al. proposed LUGrid [29], an update-tolerant on-disk grid index, that outperforms the LUR-tree in terms of update and query costs. Based on this fact, much of the recent work leverages either an in-memory grid index [5], [7], [19], [30] or an on-disk grid index [28]. Another notable technique was proposed by Hu et al. [8], who introduced the concept of *distance signature* that specifically focuses on indexing the network distance between objects by partitioning the distances into categories to prune the search space. However, this work assumes a network with long edges, which does not apply in our data sets. Consequently, our design of MOVNet utilizes an in-memory grid index to record the snapshot location of moving POIs.

Backed by a grid index, many methods have been proposed to process location-based services on moving POIs with euclidean distances. For instance, Chon et al. [5] first presented an algorithm based on the trajectory of moving POIs overlapping with the grid cells to solve snapshot range and kNN queries. By assuming computing capabilities on the mobile client side, MobiEyes [7] introduced a distributed infrastructure to process mobile range queries on dynamic POIs. Similarly, Hu et al. [9] proposed a generic framework to handle continuous queries by introducing the concept of *safe region* through which the location updates from the mobile clients can be further reduced. In contrast, SINA [18] and SEA-CNN [28] were introduced as centralized solutions with the idea of *shared execution* to process continuous range and kNN queries on moving POIs. Yu et al. [30] put forward an algorithm (referred to as YPK-CNN) for monitoring

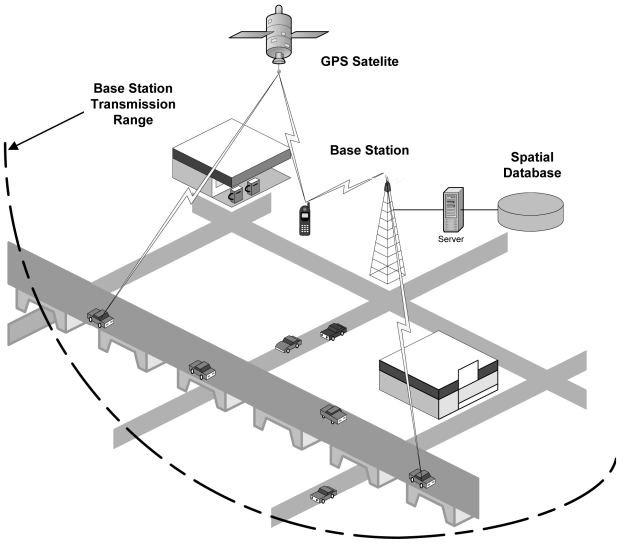

Fig. 2. An example of MOVNet mobile infrastructure.

C-kNN queries on moving objects by defining a search region based on the maximum distance between the query point and the current locations of previous kNNs. As an enhancement, Mouratidis et al. [19] presented a solution (CPM) that defines a *conceptual partitioning* of the space by organizing grid cells into rectangles. Location updates are handled only when objects fall within the vicinity of queries hence improving the system throughput. However, the above techniques do not consider network distance computation, which makes them unsuitable for applications where movement is restricted by a network.

For environments where POIs are dynamic and distances are based on network paths only a few techniques exist. Jensen et al. [12] described an abstract distributed infrastructure for handling location updates of moving POIs in a network in conjunction with a kNN query algorithm. As a centralized alternative, S-GRID [10] was introduced as a means to process kNN queries. A precomputed structure is maintained with regard to the spatial network data such as to improve the efficiency of query processing. Recently, Mouratidis et al. [20] addressed the issue of processing C-kNN queries in road networks by proposing two algorithms (namely, IMA/GMA) that handle arbitrary object and query movement patterns in road networks. This work utilizes an in-memory data structure to store the network connectivity and is, therefore, unsuitable for large-sized networks (e.g., metro cities). In contrast, MOVNet uses an on-disk R-tree structure that has a proven performance record for indexing large-sized, 2D data sets.

3 System Design

Next, we describe our data modeling of the road network, the data structures of indices, and a cell overlapping algorithm that relates the R-tree and the grid index in MOVNet.

3.1 Network Modeling and Assumptions

Fig. 2 illustrates an example of the MOVNet mobile infrastructure. We assume that mobile users are equipped with car-based or handheld GPS devices that provide the coordinates of their current location. Additionally, these devices are also able to communicate with a central spatial data server via cellular-based wireless networks (e.g., 3G and WiMax). Moreover, moving POIs, such as taxis, are also equipped with GPS devices that periodically report their locations to the spatial data server. We assume that given the location of a mobile user or a moving POI, the GPS device is able to snap (geomatch) the location to be on a road segment. Note that GPS devices have some limitations on their location accuracy which can be improved through various techniques [25].

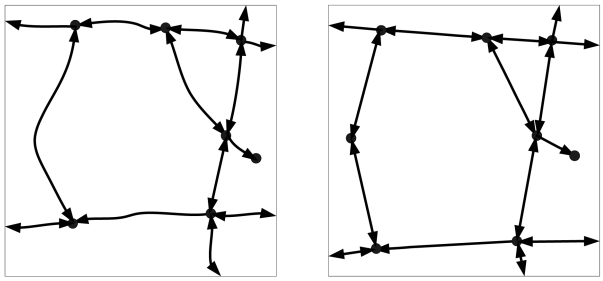

Fig. 3. An example of a road network and its corresponding, linearized modeling graph.

We define a road network (or network for short) as a directional weighted graph G consisting of a set of edges (i.e., road segments) $\mathbb{E}$, and a set of vertices (i.e., intersections and dead ends) $\mathbb{V}$, where $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$. For any network $G(\mathbb{E}, \mathbb{V})$, each edge e is represented as $e(v_1, v_2)$, i.e., it is connected to two vertices v_1, v_2, where v_1 and v_2 are the starting and ending vertex, respectively. Let $v_1 \neq v_2$. Each edge e is associated with a $length$, given by a function $length(e)$: $\mathbb{E} \rightarrow \mathbb{R}^+$, where $\mathbb{R}^+$ is the set of positive real numbers.

The road network is transformed into a *modeling graph* during query processing. Specifically, graph vertices represent the following three cases: 1) the intersections of the network, 2) the dead end of a road segment, and 3) the points where the curvature of a road segment exceeds a certain threshold so that the road segment is split into two pieces to preserve the curvature property. Although polylines can also be used to represent the edges, we use a set of line segments to represent an edge due to the nature of our data set. As a result, the modeling graph is a piecewise linear approximation of the network. For example, Fig. 3a shows a small road network, and Fig. 3b illustrates the corresponding modeling graph.

There are different objects (e.g., cars, taxis, and pedestrians) moving along the road segments in a network. These objects are known as the set of *moving objects* $\mathbb{M}$. A moving object $m \in \mathbb{M}$ is a POI located in the network. For simplicity, we assume that a map-matching procedure is invoked when a POI sends its location to the central server. Hence, a POI is presumed to be located on a network edge in the system. The position of m at time t is defined as $loc_t(m) = (x_m, y_m)$, where x_m and y_m are the x and y coordinates of m at time t, respectively. A query point $q \in \mathbb{M}$ is a moving object issuing a location-based spatial query at different times. Currently, our design is focusing on snapshot range queries (e.g., "find all taxis within a three mile range from my current location").

Note that these queries are processed with network distances. For simplicity, we use the term distance to refer to the network distance in the following sections.

To represent the snapshot location of moving objects, MOVNet assumes that periodic sampling of the moving object positions conveys their locations as a function of time. Specifically, a spatial query submitted by a user at time t_1 is computed based on $loc_{t_0}(\mathbb{M})$. The system maintains the latest snapshot of moving objects t_0 with $t_0 \leq t_1$, $t_1 - t_0 < \Delta t$, and Δt is a fixed time interval; the result is valid until $t_0 + \Delta t$. This method is commonly used (see [28], [30]) to represent the snapshot location of moving objects. Note that slight result inaccuracies may arise because of the discreet sampling of time and communication delays. The result converges to the accurate value as the time interval Δt and the wireless transmission delays approach zero. The sampling rate is also related to the maximum object speed that is expected in the system. The system designer must tune the sampling parameter to achieve a good trade-off between the result precision and the sampling overhead.

We define the distance function of two moving objects m_1 and m_2 at time t as $dist_t(m_1, m_2)$: $loc_t(m_1) \times loc_t(m_2) \rightarrow \mathbb{R}^+$. Here, $dist_t(m_1, m_2)$ denotes the shortest path from m_1 to m_2 in the metric of the network distance at time t. For notational simplicity, we denote $dist(m_1, m_2)$ as the distance function of m_1 and m_2 at the current time. Similarly, the distance function of an edge $e(v_1, v_2)$ and a moving object m at time t is defined as $dist_t(e, m)$: $loc(v_1) \times loc_t(m) \rightarrow \mathbb{R}^+$. Finally, $dist_t(e, m)$ denotes the shortest path from v_1 to m in the metric of the network distance at time t.

The distance between two moving objects depends on the length of edges and the connectivity of vertices as well as the current locations of the objects. We elaborate on our dual-index structure designed to facilitate distance computations in the following section.

3.2 Dual-Indexing Structure Design

To record the connectivity and coordinates of vertices in stationary networks, MOVNet utilizes an on-disk R*-tree, a data structure which has been intensively studied for handling very large 2D spatial data sets. Once the edges are retrieved from disk, a corresponding modeling graph is constructed in memory using the following structure. We use a vertex array to store the coordinates of vertices in the graph. For each vertex, the array maintains a list that records its outgoing edges. To quickly locate a vertex in the array, MOVNet manages a hash table to map the coordinate of a vertex into its index in the vertex array.

A memory-based grid index is used to manage the locations of moving objects [30]. Without loss of generality, we assume that the service space is a square. We can partition the space into a regular grid of cells with a size of $l \times l$. We use $c(column, row)$ to denote a specific cell in the grid index (assuming that the cells are ordered from the lower left corner of the space). At time t, a moving object m is positioned at $loc_t(m) = (x_m, y_m)$, therefore, it overlaps with cell $c(\lfloor \frac{x_m}{l} \rfloor, \lfloor \frac{y_m}{l} \rfloor)$. Each cell maintains an object list containing the identifiers of enclosed objects. The objects' coordinates are stored in an object array, and the object identifier is the index into this array. Fig. 4 shows a part of the network of Fig. 3b that is managed by a grid index of 8×8 cells. An example object on $e(v_2, v_4)$ is enclosed by $c(5, 5)$. Accordingly, the object list of $c(5, 5)$ records the object identifier and, hence, we can retrieve the coordinate of the object from the object array.

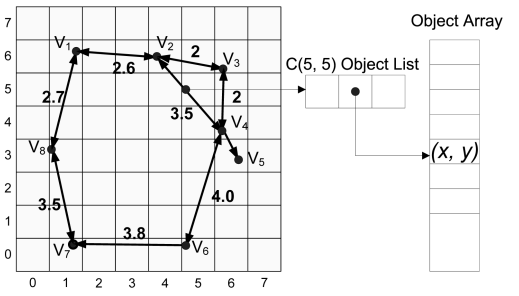

Fig. 4. An example network indexed by the grid index and its data storage structure.

Given a set of grid cells, retrieving the underlying network can be transformed into range queries on the R-tree. It is highly desirable to have an algorithm such that for an arbitrary edge, we are able to find the set of overlapping cells very quickly. Although this is related to the line rasterization algorithm (e.g., Bresenham's Algorithm [2]), it is noteworthy to point out that these existing algorithms only obtain an approximation of the overlapping cells (or pixels, in that case). In contrast, our goal is to compute the complete set of overlapping cells. Therefore, we devise an incremental algorithm.

First, let us assume that the service space is managed by a grid-based index. We define the set of cells $\{c_1, c_2, \ldots, c_n\}$, which are consecutively overlapped from v_1 to v_2 by an edge $e(v_1, v_2)$, as the set of *affected cells* of e. For instance, in Fig. 4, the affected cells of $e(v_1, v_2)$ are $\{c(1, 6), c(2, 6), c(3, 6), c(4, 6)\}$.

We use straight line segments to represent edges in the network. Therefore, any edge $e(v_1, v_2)$ can be described by a first degree polynomial function in the form of $y = m \cdot x + b$ with $x \in [x_{v_1}, x_{v_2}]$. Given an edge $e(v_1, v_2)$, the coordinates of vertex v_1 and v_2 are (x_{v_1}, y_{v_1}) and (x_{v_2}, y_{v_2}), respectively. To compute the set of affected cells of e, we first capture the gradient m and the y-intercept b of e. After that, we compute the cells overlapping with the starting and ending vertex of the edge, respectively. Next, we follow a step-forward approach where in each step, we move one cell on the x-axis from the cell overlapping with the starting vertex and calculate the affected cells along the y-axis. Finally, we terminate once we reach the cell that contains the ending vertex.

Computing the set of overlapping cells with regard to an edge has linear complexity in the length of the edge. Our experimental results show that the CPU time used for computing overlapping cells consumes less than five percent of the query processing time with various settings. This indicates that our method is well suited for online computing. More importantly, MOVNet creates a means to bidirectionally map underlying networks and moving object positions. We present our query design in the following sections showing the flexibility and scalability of this dual-index approach.

4 Query Design

In this section, we first describe our design of a mobile range query algorithm. Next, we present the minimum and maximum bounds on the number of grid cells that can overlap with an arbitrary edge. Then, the maximum bound is used to prune the search space during query processing. Finally, we propose a mobile kNN query algorithm by introducing the concept of a progressive probe and leveraging our range query algorithm.

4.1 Range Query Algorithm

Given a query point q, a value d, a network G, and a set of moving objects $\mathbb{M}$, a location-based network distance range query retrieves all POIs of $\mathbb{M}$ that are within the distance d from q at time t. By using the definitions of Section 3, the query can be represented as $rangeQuery_t(q, d)$: $loc_t(q) \times loc_t(\mathbb{M}) \rightarrow \{m_i, i = 1, \ldots, n\}$, $\forall m_i$, $dist_t(q, m_i) \leq d$.

We propose a Mobile Network Distance Range (MNDR) query algorithm to facilitate the query processing. First, we know from the *euclidean distance restriction* [21] property that the distance $dist(q, m)$ for object m in a network is always larger than or equal to the euclidean distance d of q to m. We observe that only the network data in MOVNet is stored on disk. Therefore, we first perform a euclidean range query with q as the center and d as the radius to retrieve the network from the R-tree and to create the corresponding modeling graph. After that, we are able to perform the later steps efficiently in memory. Second, the starting vertex of an edge $e(v_1, v_2)$ has the property that if $dist(q, v_1) > d$, the affected cells of the edge are not required to be examined during this first pass because any moving object on e has a distance greater than d from q. Hence, for each vertex in the modeling graph, MNDR leverages Dijkstra's algorithm [6] to compute the distance from q. In addition, our algorithm avoids unnecessary processing on any edge with a distance from the query point greater than d. Finally, for each edge whose distance of the starting vertex is within the query range d, MNDR generates the list of affected cells and retrieves the corresponding moving objects from the grid index.

Algorithm 1. Mobile Network Distance Range Query (q, d)

1: /* q is the query object */
2: /* d is the distance */
3: $result = \phi$
4: /* Finding the set of edges $\mathbb{E}'$, AND vertices $\mathbb{V}'$ overlapped by the circle with center point q, and radius d */
5: ($\mathbb{E}'$, $\mathbb{V}'$) = euclidean-range(q, d)
6: G = Create-modeling-graph($\mathbb{E}'$, $\mathbb{V}'$)
7: q = Add-vertex-into-graph(G, q, e)
8: S = Compute-distance(G, q, d)
9: **for** each vertex v in S **do**
10: **for** each edge e outgoing from v **do**
11: $cellSet = cellSet \cup cellOverlapping(e, d - dist(q, v))$
12: **end for**
13: **end for**
14: $result$ = Retrieve-objects($cellSet$, G)
15: **for** each object m in $result$ **do**
16: $dist(q, m) = min(dist(q, v_1) + dist(v_1, m), dist(q, v_2) + dist(v_2, m))$
17: **if** $dist(q, m) > d$ **then**
18: $result = result - m$
19: **end if**
20: **end for**
21: return $result$

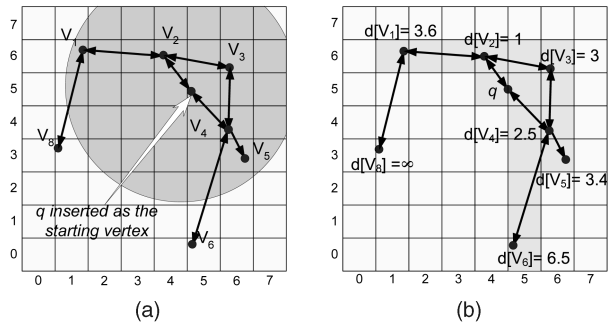

Fig. 5. A mobile network distance range query example.

Algorithm 1 details MNDR. To illustrate the algorithm with an example, let us assume that the system is processing a network as shown in Fig. 4, where the side length of cells is 1.0 unit. A query object q with $dist(q, v_2) = 1.0$ submits a range query with a range $d = 3.5$. MOVNet first invokes a euclidean distance range query with q as the center and d as the radius (Line 5 of Algorithm 1). Consequently, edges overlapping with the shadowed area will be retrieved from the R-tree index and a corresponding modeling graph is built as shown in Fig. 5a (Line 6). Note that q is inserted as the starting vertex into the modeling graph (Line 7). Next, Dijkstra's algorithm is invoked (Line 8). We add a constraint d in the distance computation so that any edge $e(v_1, v_2)$ with $dist(q, e) > d$ will not be processed, which avoids excessive computation on edges that are out of range. When Dijkstra's algorithm finishes, the distance of each vertex from q is shown in Fig. 5b. In addition, $S = \langle (v_2, 1), (v_4, 2.5), (v_3, 3), (v_5, 3.4) \rangle$. Based on this information, MNDR computes $cellSet$ by using our cell overlapping algorithm in Lines 9-13, shown as the dark gray cells in Fig. 5b. After that, the moving objects in $cellSet$ are retrieved from the grid index to constitute the result set. However, several postprocessing steps are required to ensure that the distance of each moving object is within range d. First, some objects may be reachable via more than one path from the query point. MOVNet will only consider the shortest path and examine the path against the range d (Line 16). For example, moving objects on edge $e(v_3, v_4)$ have two paths from q ($q \rightarrow v_2 \rightarrow v_3$, $q \rightarrow v_4$). MNDR will compute the distance of each object via every path, and only use the shortest one. Second, once the distance from q to the object is determined, MNDR confirms that the distance is $\leq d$. For instance, for any object m retrieved from $c(5, 0)$, $dist(q, m) > 3.5$, thus, the algorithm removes these objects in Lines 17-19.

When we compute $cellSet$ in Algorithm 1 (Lines 10-14), some cells can be further pruned before the system retrieves the moving objects from the corresponding grid index. For instance, in the example illustrated above, $e(v_4, v_6)$ overlaps with six cells. Some of the cells can be pruned because their distances from $q > d$. This optimization can be achieved by using the geometric properties as described in the following section.

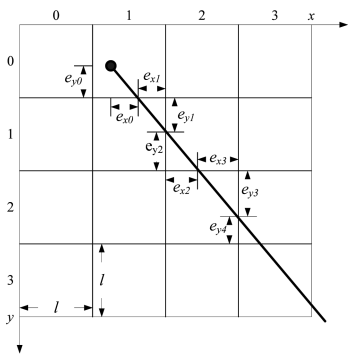

Fig. 6. Computing the length of edges with regard to the number of grid cells.

4.2 The Minimum and Maximum Number of Cells Overlapping with an Edge

We present an important geometric property that relates an arbitrary edge overlapping with the grid cells it overlaps. Since the edge is represented as a straight line segment, the relationship between the length of an edge $e(v_1, v_2)$ and the number of its affected cells can be described as follows:

Lemma 1. *Assume that the service space is managed by a grid-based index with a cell size of $l \times l$. For an edge $e(v_1, v_2)$ with a set of affected cells $\{c_1, c_2, \ldots, c_n\}$, the maximum length of e is $\sqrt{2} \times l \times n$. The minimum length of e is*

$$\begin{cases} 0, & 1 \le n \le 2, \\ \sqrt{\left\lfloor \frac{n-3}{2} \right\rfloor^2 + \left\lfloor \frac{n-2}{2} \right\rfloor^2} \cdot l, & n \ge 3. \end{cases}$$

Proof. Without loss of generality, let us consider an edge e in the service space that overlaps with grid cells as shown in Fig. 6. Assume that the number of affected cells for e is n. Therefore, for $0 \le e_{xi} \le l$, $0 \le e_{yi} \le l$, we have

$$length(e) = \sqrt{\sum_{i=0}^{n-1} e_{xi}^2 + \sum_{i=0}^{n-1} e_{yi}^2}. \quad (1)$$

We observe that, when $e_{xi} = e_{yi} = l$, where $0 \le i \le n-1$, we have the maximum length of e when substituting e_{xi} and e_{yi} in (1)

$$length_{max}(e) = \sqrt{n^2 \cdot l^2 + n^2 \cdot l^2} = \sqrt{2} \cdot l \cdot n.$$

To compute the minimum length of e, we observe from Fig. 6 that $e_{y1} + e_{y2} = e_{x2} + e_{x3} = l$, and so on, which can be summarized as

$$\begin{cases} e_{x(2j)} + e_{x(2j+1)} = l, & 1 \le j \le \left\lfloor \frac{n-3}{2} \right\rfloor, \\ e_{y(2k-1)} + e_{y(2k)} = l, & 1 \le k \le \left\lfloor \frac{n-2}{2} \right\rfloor. \end{cases} \quad (2)$$

For simplicity, we use E_{xj} to refer to $e_{x(2j)} + e_{x(2j+1)}$ and E_{yk} to refer to $e_{y(2k-1)} + e_{y(2k)}$ from here on.

When $n = 1$, the minimum length of

$$e = \sqrt{e_{x0}^2 + e_{y0}^2} = 0,$$

where $e_{x0} = e_{y0} = 0$. Similarly, when $n = 2$, the minimum length of $e = 0$, where $e_{x0} = e_{y0} = e_{x1} = e_{y1} = 0$.

When n is $\ge$3 and odd, we have

$$\begin{cases} \sum_{i=0}^{n-1} e_{xi}^2 = \left(e_{x0} + e_{x1} + \sum_{j=1}^{\lfloor \frac{n-3}{2} \rfloor} E_{xj} + e_{x(n-1)} \right)^2, \\ \sum_{i=0}^{n-1} e_{yi}^2 = \left(e_{y0} + \sum_{k=1}^{\lfloor \frac{n-2}{2} \rfloor} E_{yk} + e_{y(n-2)} + e_{y(n-1)} \right)^2. \end{cases}$$

Using the properties in (2), the above equations can be transformed into

$$\begin{cases} \sum_{i=0}^{n-1} e_{xi}^2 = \left(e_{x0} + e_{x1} + \left(\left\lfloor \frac{n-3}{2} \right\rfloor \right) \cdot l + e_{x(n-1)} \right)^2, \\ \sum_{i=0}^{n-1} e_{yi}^2 = \left(e_{y0} + \left(\left\lfloor \frac{n-2}{2} \right\rfloor \right) \cdot l + e_{y(n-2)} + e_{y(n-1)} \right)^2. \end{cases}$$

Substituting the corresponding parts of (1) with the above equations, we can conclude that, if $e_{x0} = e_{x1} = e_{x(n-1)} = e_{y0} = e_{y(n-2)} = e_{y(n-1)} = 0$, then

$$length_{min}(e) = \sqrt{\left\lfloor \frac{n-3}{2} \right\rfloor^2 + \left\lfloor \frac{n-2}{2} \right\rfloor^2} \cdot l.$$

Similarly, when n is $\ge$3 and even, we have

$$\begin{cases} \sum_{i=0}^{n-1} e_{xi}^2 = \left(e_{x0} + e_{x1} + \sum_{j=1}^{\lfloor \frac{n-3}{2} \rfloor} E_{xj} + e_{x(n-2)} + e_{x(n-1)} \right)^2, \\ \sum_{i=0}^{n-1} e_{yi}^2 = \left(e_{y0} + \sum_{k=1}^{\lfloor \frac{n-2}{2} \rfloor} E_{yk} + e_{y(n-1)} \right)^2. \end{cases}$$

Using the same properties as shown in (2), we can conclude that

$$length_{min}(e) = \sqrt{\left\lfloor \frac{n-3}{2} \right\rfloor^2 + \left\lfloor \frac{n-2}{2} \right\rfloor^2} \cdot l.$$

Therefore, we have proved that when $n \ge 3$, in both even and odd cases,

$$length_{min}(e) = \sqrt{\left\lfloor \frac{n-3}{2} \right\rfloor^2 + \left\lfloor \frac{n-2}{2} \right\rfloor^2} \cdot l.$$

□

Lemma 1 states the minimum and maximum bounds of the length of an edge given a fixed number of cells. We further derive from Lemma 1 the maximum and minimum number of affected cells with regard to an arbitrary edge.

Corollary 1. *Assume that the service space is managed by a grid-based index with a cell size of $l \times l$. For an edge $e(v_1, v_2)$, the maximum and minimum numbers of affected cells are $\lceil \frac{\sqrt{2} \cdot length(e)}{l} \rceil + 3$, and $\lfloor \frac{length(e)}{\sqrt{2} \cdot l} \rfloor$, respectively.*

Proof. We know from Lemma 1 that, given an edge $e(v_1, v_2)$, $length(e) \le \sqrt{2} \cdot l \cdot n$, hence, we can directly deduce that $n \ge \lfloor \frac{length(e)}{\sqrt{2} \cdot l} \rfloor$.

Similarly, since

$$length(e) \ge \sqrt{\left\lfloor \frac{n-3}{2} \right\rfloor^2 + \left\lfloor \frac{n-2}{2} \right\rfloor^2} \cdot l,$$

it follows that

$$length(e) \geq \sqrt{2 \cdot \left\lfloor \frac{n-3}{2} \right\rfloor^2} \cdot l,$$

which leads us to conclude that $n \leq \lceil \frac{\sqrt{2} \cdot length(e)}{l} \rceil + 3$. □

We utilize the property of the maximum number of affected cells in Corollary 1 to prune the search space. Let us assume that MNDR generates the list of cells overlapping with an edge $e(v_1, v_2)$ and there are n_1 affected cells. By using Corollary 1, we conclude that a range $d - dist(q, v_1)$ is only able to overlap with at most n_2 cells, with $n_2 < n_1$. Therefore, MNDR will only record the first n_2 cells on $e(v_1, v_2)$ into $cellSet$. As an example, consult Fig. 5b. We know that $dist(q, v_4) = 2.5$, therefore, we only need to record cells on $e(v_4, v_6)$ within a range of $3.5 - 2.5 = 1.0$ from v_4. Using the maximum bound of the number of affected cells, MNDR records the first five cells on $e(v_4, v_6)$ starting from v_4, even though there are six cells overlapping with $e(v_4, v_6)$.

In summary, Corollary 1 provides a precise range on how edges overlap with grid cells. Our simulation results indicate that this property offers substantial performance improvements when computing the affected cells over long edges (i.e., freeway segments).

4.3 k Nearest Neighbor Query Algorithm

Given a query point q, a value k, a network G, and a set of moving objects $\mathbb{MI}$, the network-distance-based k nearest neighbor query retrieves the k objects of $\mathbb{MI}$ that are closest to q according to the network distance at time t. Formally, a mobile kNN query is represented as $kNNQuery_t(q, k)$: $loc_t(q) \times loc_t(\mathbb{MI}) \rightarrow \{m_i, i = 1, \ldots, k\}$, where $\forall m_j = \mathbb{MI} - m_i$, $dist_t(q, m_j) \geq dist_t(q, m_i)$.

To cope with this type of query, we propose a Mobile k Nearest Neighbor (MKNN) query algorithm leveraging our MNDR algorithm to efficiently compute the kNN POIs from the query point in the network. We observe that the grid index in MOVNet enables fine-grained space partitioning. Additionally, the grid index maintains an object list in each grid cell, which can be quickly accessed to retrieve the number of enclosed objects. Therefore, we begin by searching the surrounding area of the query point in the grid index and continuously enlarging the area until we are able to find a subspace that contains kNN POIs in terms of the euclidean distance. We term this procedure a *progressive probe*. Note that in the progressive probe, we only retrieve the size of the object list from each cell, while the distance of each object from the query point is not computed because we aim to obtain an approximate area enclosing kNN objects within network distance. Our experimental study shows that in 30 to 48 percent of the test cases the actual number of kNN objects is bounded by our progressive probe. More importantly, the complexity of retrieving the object list size from each cell is $O(1)$, which is very efficient especially since our grid index is an in-memory structure.

We define that cells in the grid index are grouped into levels centered at $c(\lfloor \frac{x_q}{l} \rfloor, \lfloor \frac{y_q}{l} \rfloor)$, where q is a moving object submitting a mobile kNN query and l is the side length of a grid cell. The first level L_0 is the single cell $c(\lfloor \frac{x_q}{l} \rfloor, \lfloor \frac{y_q}{l} \rfloor)$ and cells in the next level are the surrounding cells of L_0, and so on. By using the definition above, the progressive probe first retrieves the number of objects in L_0 via the grid index. If there are less than k objects in L_0, it continues to scan the number of objects in the next level of cells, and so on. Fig. 7a illustrates an example of these steps. Assume the system is maintaining a network as shown in Fig. 4 and a query object q in $c(5, 5)$ submits a nearest neighbor query with $k = 10$. The progressive probe first locates q in $c(5, 5)$, which becomes L_0. After that, the number of POIs in $c(5, 5)$ is retrieved from the grid index. If there are less than 10 POIs in L_0, the progressive probe sequentially searches the next levels L_i, where $i \in \{1, 2, \cdots\}$, illustrated in the shadowed areas in Fig. 7a. Assuming that at least 10 POIs have been found after the scan in L_2, the probe stops and results in an estimated space for kNN objects in the network. Because the R-tree is on the secondary storage in MOVNet and the number of disk I/Os should be minimized, MKNN utilizes this estimated area to launch a range query extracting the edges from the R-tree, instead of following a network expansion approach to retrieve a few edges at a time.

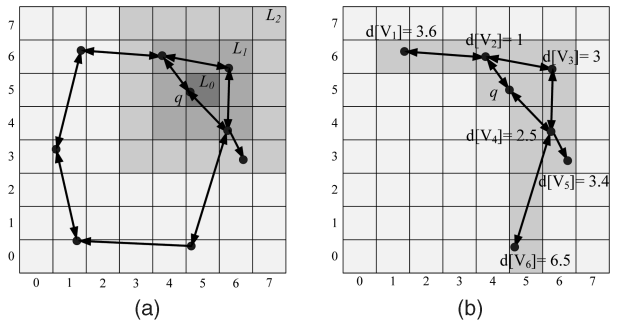

Fig. 7. A mobile network distance k-NN query example.

We also introduce the following data structures: $candidateObjs$ and $unvisitedVertices$. These are minimum priority queues on the value of the distances from the query point. The set of candidate objects is retrieved from the grid index as possible objects in the final result set. The set of unvisited vertices is to be expanded when there are less than k objects found during query processing. Additionally, we manage $resultObjs$ as a maximum priority queue in terms of the distance from the query point with a size of k.

Algorithm 2 elaborates on the MKNN algorithm. MKNN first executes the progressive probe in the grid index so that an approximate query result space is created. After that, MKNN uses this subspace as an initial range to invoke the MNDR module so that the corresponding edges are retrieved from the R-tree and the distance of each vertex from q is computed (Lines 5-8). Given the example of Fig. 7a, Fig. 7b demonstrates the correlated modeling graph and the distance to each vertex.

Algorithm 2. Mobile Network Distance kNN Query (q, k)
1: /* l is the side length of cell */
2: $foundkObjs = false$, $visitedVertices = \phi$
3: $radius =$ Progressive-probe(q, k, l)
4: **while** $foundkObjs = false$ **do**
5: ($\mathbb{E}'$, $\mathbb{V}'$) = euclidean-range(q, $radius$)
6: $G =$ Create-modeling-graph($\mathbb{E}'$, $\mathbb{V}'$)
7: $q =$ Add-vertex-into-graph(G, q, e)
8: $S =$ Compute-distance(G, q)

```
 9:   unvisitedVertices = S − visitedVertices
10:   while unvisitedVertices ! = NULL do
11:     minVertex = De-queue(univisitedVertices)
12:     cellSet = φ
13:     if resultObjs.size = kAND
        minVertex.dist ≥ kth resultObjs.dist then
14:       foundkObjs = true
15:       break
16:     end if
17:     for each edge e outgoing from minVertex do
18:       cellSet = cellSet ∪
          cellOverlapping(e, d − dist(q, v))
19:     end for
20:     candidateObjs = candidateObjs ∪
        Retrieve-objects(cellSet, G)
21:     while resultObjects.size < k do
22:       De-queue(candidateObjs) to resultObjs
23:     end while
24:     while Peak(candidateObjs).dist ≤
        kth resultObjs.dist do
25:       Swith(De-queue(candidateObj),
          De-queue(resultObjs))
26:     end while
27:   end while
28:   radius = minVertex.dist
29: end while
30: return resultObjs
```

Next, a vertex is dequeued from $unvisitedVertices$ (Line 11). For each outgoing edge from the vertex, the set of affected cells is computed and objects are retrieved from the corresponding grid cells and placed into $candidateObjs$ (Lines 17-20). After that, we examine two possible cases: first, if there are less than k objects in $resultObjs$, MKNN dequeues objects from $candidateObjs$ into $resultObjs$ (Lines 21-23). Second, the distance of the first element of $candidateObjs$, the kth result object will be dequeued and inserted into $candidateObjs$. Next, $candidateObjs$ dequeues an object and inserts it into $resultObjs$ (Lines 24-26).

The algorithm terminates when $resultObjs$ contains k POIs and the distance of the kth result object is less than the distance of the minimum vertex in $univsitedVertices$ (Lines 13-16). Otherwise, if the last vertex v in the modeling graph (i.e., the vertex with the longest distance to q) is visited and the distance of the kth result object is greater than $dist(q, v)$, MKNN will use $dist(q, v)$ as the radius to launch a range query in the R-tree as a new iteration of MKNN (Line 32). Although this step causes I/O operations as well as the overhead of creating a modeling graph again, MKNN maintains the set of visited vertices in each iteration to avoid visiting these vertices in future iterations (Line 13). As our simulation results have verified, under various settings, MKNN requires no more than two iterations during query processing in more than 97 percent of the test cases. Therefore, this method significantly reduces the I/O cost and ensures high system throughput.

5 Theoretical Analysis

We present our theoretical analysis of MOVNet in the following sections. We assume that the network and moving objects are uniformly distributed in a one unit square space (i.e., for an object m, $0 \leq x_m < 1$, and $0 \leq y_m < 1$). This is an optimal simplification, which is analogous to previous studies [30], [19]. A grid index with $l \times l$ side length manages the moving object location updates. The total number of edges and moving objects in the network are E and M, respectively.

5.1 Analysis of MNDR

For a MNDR query with a range d, let us assume the query covers an area of $4d^2$. Although the euclidean distance query in MNDR is in actuality performed within an area of πd^2, our assumption does not change the quality of our analysis. During the processing of MNDR, there are $O(d^2E)$ edges retrieved from the on-disk R-tree in the euclidean distance range query. The next step that creates a modeling graph is of complexity $O(d^2E)$ since every edge will be recorded in the graph. Finding the edge, where the query point is located can be achieved during the modeling graph construction. Additionally, inserting the query point into the modeling graph as the starting vertex requires only $O(1)$ operations. The running time of Dijkstra's algorithm to compute the distance of each vertex from the query point is $O(d^2E{\cdot}log(d^2E))$. Next, MNDR calculates the cell set overlapping with the edges based on the distance information. Note that each edge is examined at most once during the course of this step. Therefore, $O(d^2E)$ iterations are needed to calculate the overlapping cells. Moreover, since the length of the edge is bounded by d, the total complexity of this step is $O(d^3E)$. Finally, MNDR retrieves the objects from the grid index and computes the result set. For a range query with a side length of $2d$, the number of overlapping grid cells is $(2d + l)^2/l^2$ [27]. For each cell, we can assume that there are l^2M objects. Hence, the number of moving objects retrieved in the final step is $O((2d + l)^2M)$. To sum up, the cost of MNDR can be represented as $O(d^2Elog(d^2E) + (2d + l)^2M)$.

We observe that the cost of MNDR is linear in the number of POIs. Similarly, the system throughput is proportional to the side length of cells (or inversely proportional to the number of cells). Additionally, both factors are lower bounded by the cost of graph construction, Dijkstra's algorithm, and the overlapping cell computation. Finally, the CPU cost is a quadratic function of d, which means a larger range results in a serious increase in CPU cost.

5.2 Analysis of MKNN

For simplicity, let us assume that the progressive probe results in a subspace containing k nearest neighbor objects. In the case that MKNN needs to expand to a larger space with more iterations, it can be modeled with our cost model times a constant, which does not change the characteristic of our analysis.

Since we assume that POIs are uniformly distributed, the subspace containing kNN objects has a size of $\frac{k}{M}$. Therefore, MKNN needs to scan $\frac{k}{M \cdot l^2}$ cells to find the kth object and return a subspace. The subsequent steps that perform a euclidean distance range query, construct the modeling graph, compute the overlapping cells, and retrieve objects are the same as the ones in MNDR. Hence, the cost in these operations can be summarized as

$$O\left(\frac{kE}{M}\left(2 + log\frac{kE}{M} + \sqrt{\frac{kE}{M}}\right) + \left(\sqrt{\frac{kE}{M}} + l\right)^2 M\right).$$

The final step that filters objects from $candidateObjs$ into $resultObjs$ is bounded by the size of $resultObjs$ (i.e., k). In summary, the cost of MKNN can be simplified as

$$O\left(\frac{k}{Ml^2}+\frac{kE}{M}\sqrt{\frac{kE}{M}}+\left(\sqrt{\frac{kE}{M}}+l\right)^2 M\right).$$

The equation above shows that the CPU cost of MKNN is proportional to k. The explanation is that with an increasing k, MKNN needs to search for a larger space to find the query result. Additionally, the CPU cost is inversely proportional to the number of objects. This is because with more POIs, the search space for finding the kth object becomes smaller, and vice versa. Finally, the system throughput as a function of the cell size is bounded by two factors: the cost from the progressive probe and the cost of retrieving objects from the grid index. A smaller cell size results in more overhead from the progressive probe. In contrast, increasing the size of cells implies that more objects are retrieved from the grid index. We present our simulation results in the following section as an experimental verification of our theoretical analysis.

6 Experimental Evaluation

To evaluate the performance of MOVNet, we performed extensive simulations on a real dense road network. The result indicates that MOVNet achieves good throughput with a wide variety of data settings. In Section 6.1, we start by describing the data sets used in our simulation and our simulator implementation. Experimental results and the corresponding discussion are presented in Section 6.2.

6.1 Simulator Implementation

We are using the TIGER/Line[1] data set for our road network segments and a moving object generator [3] to create the locations of 100,000 moving objects over time. Therefore, the locations we used in our simulation have the same format as that of the TIGER/Line data, which is expressed as Longitude/Latitude. For each road network (e.g., the complete set of the LA county road segments), TIGER/Line defines four bounding coordinates: the west, east, north, and south bounding coordinate. We use these bounding coordinates to define the service space and overlap it with our grid index. The Los Angeles (LA) County data set has 304,162 road segments distributed over an area of 4,752 square miles. The average length of road segments is 0.1066 miles. For simplicity, we assume that each road segment is bidirectional. The network data are indexed with an R*-Tree.

Existing work, such as IMA/GMA, only focuses on C-kNN query processing. This method differs from the functionality of MOVNet which supports both snapshot range and kNN query processing. Therefore, we leveraged the concept of *network expansion* [21] to design baseline algorithms for performance comparisons in our simulations. The baseline algorithm for mobile range queries executes as follows: First, we retrieve the edge where the query point is located. Next, the closest vertex to the query point is expanded and outgoing edges from this vertex are retrieved. The expansion stops once all vertices whose distances from the query point are less than d have been expanded. After that, for each expanded edge, the overlapping cells are computed and POIs in these cells are retrieved to constitute the result set. Based on the same idea, the baseline algorithm for mobile kNN queries has the following steps. First, we locate the road segment on which the query point is moving and compute the affected cells. Next, the corresponding moving objects are retrieved from the affected cells. If there are less than k objects in the result set, or if the distance from the query point to the closest vertex is less than the distance of the kth object from the query point, the closest vertex is expanded and the outgoing edges from this vertex are retrieved. Afterward, the set of affected cells on the outgoing edges is computed and the corresponding objects are retrieved from the grid index. The vertex expansion process stops when there are k objects in the result set and the distance from query point to the kth object is no greater than the distance from the query point to the closest unexpanded vertex.

We also implemented the design of S-GRID [10] in Java to compare it with the CPU cost of MKNN. Specifically, we implemented the *Vertex-Edge* component of S-GRID as an on-disk module. Edges of the network are indexed by an R*-tree. The precomputing component of S-GRID is stored in memory.

We implemented a simulator in Java. The simulation was executed on a workstation with 1 GB memory and a 3.0 GHz Xeon processor. We arranged the road segments of the LA county data set into a R*-tree index file, in which we set the page size to 4 KB. Each road segment is stored in a Minimum Bounding Rectangle (MBR) bounded by its starting and ending coordinates. To achieve a fair comparison, our baseline algorithms also use this R*-tree index structure to speedup the edge retrieval during query processing. For each test case, our simulator creates a service space with the area equal to the LA county size. It then opens the R*-tree index file and uses a buffer for caching the disk pages read by MOVNet with a size of 10 pages. Next, an in-memory grid index is created with the positions of the moving objects. To simplify the map-matching process, we assume that object locations always fall along the road segments. In the next step, the query generator randomly picks a moving object and launches a query from its location. Table 1 summarizes the parameters used. In each experimental setting we varied a single parameter and kept the remaining ones at their default values. The experiments measured the CPU time (in milliseconds) and the number of disk page accesses as the performance metrics of the query processing. For each experimental configuration, the simulator executed 1,000 iterations and reported the average result.

TABLE 1
Simulation Parameters

Parameter	Default	Value Range
Number of POIs	50K	10K - 100K
Number of NNs (k)	50	2 - 128
Radius (mile)	5	2 - 10
Number of cells per axis	1K	200 - 1,400

6.2 Simulation Results

We were first interested in verifying the update costs from POIs in MOVNet. Since we use an in-memory grid index to

1. http://www.census.gov/geo/www/tiger/.

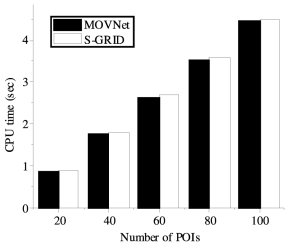

Fig. 8. The CPU time of update cost as a function of POIs (assuming 10 percent of the POIs sends updates).

handle these updates, there is no disk access to the R*-tree index file, which is on secondary storage. We measured the CPU time of the update process. Note that the update and query processing should be finished in one update cycle to ensure the correctness of the query results. We assume that at the beginning of each update period, 10 percent of the POIs submit their new positions. Fig. 8 shows that when there are 10,000 update messages in one period, MOVNet is able to record these changes in about 4.5 seconds. Additionally, the update cost of MOVNet is slightly less than that of S-GRID. This is because both techniques include the map-matching procedure for object updates to record the edge where the object is located. Moreover, S-GRID records the object into a cell if its nearest vertex on edge e belongs to this cell. Therefore, distance computing is performed during object update in S-GRID. In contrast, MOVNet directly insert the object into the cell that encloses it, which simplifies the update procedure.

Next, we verified the performance of MNDR. Fig. 9a illustrates the effect when varying the number of cells with the LA county data set. The results show that MNDR requires less than half of the CPU time compared with the baseline algorithm. Correspondingly, Fig. 9b studies the page accesses of both algorithms. As we can see, the baseline algorithm consumes more than 3,000 page accesses with various cell sizes. By comparison, MNDR requires less than 100 page accesses during query processing. An important observation is that a small number of cells cause the CPU time of MNDR to degrade. On the other hand, the disk access of MNDR is stable with different cell sizes. This can be explained by the fact that a disk access only occurs when we retrieve the road segments from the R*-tree file. Since we use a fixed range in this test, the number of disk accesses is not affected by changing the cell size. However, a larger cell size will result in an increased number of POIs being retrieved from the grid index during query processing. Therefore, the CPU time expended in this portion is larger than with smaller cell sizes. Overall, we conclude that

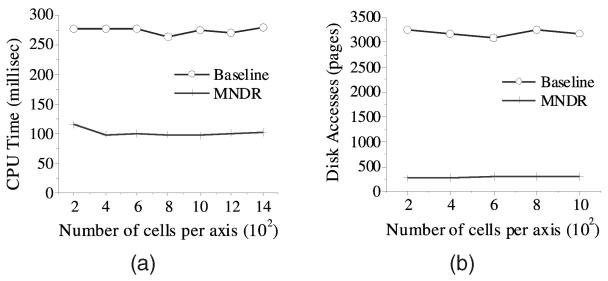

Fig. 9. The performance of MNDR as a function of the number of cells.

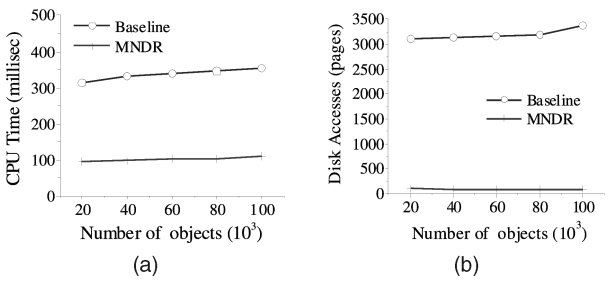

Fig. 10. The performance of MNDR as a function of POIs.

MOVNet scales very well with varying cell numbers. Note that with MNDR, the setting of 1,000 cells per axis achieves a stable and optimal performance, hence, we set the default number of cells per axis to be 1,000 in our other tests.

Next, Fig. 10a illustrates the effect of the number of POIs on the execution time of MNDR. As shown, MNDR outperforms the baseline algorithm with various numbers of POIs. In the case of $20K$ POIs, the CPU time of MNDR is only about 30 percent of that of the baseline algorithm. Additionally, the graph shows that the CPU time increases linearly with the number of POIs, which follows our complexity analysis expectation. The very small gradient of the MNDR output suggests that MOVNet is very scalable to support a very large number of POIs. More importantly, with $100K$ POIs, the processing time for LA county is about 0.1 seconds. This demonstrates how efficiently MOVNet executes. Fig. 10b plots the disk accesses of both algorithms. Similarly to the CPU time outputs, MNDR performs consistently much lower than the baseline algorithm.

Fig. 11a plots the CPU time (with logarithmic scale) versus the query range with the LA county set. The CPU time quadratically increases with a larger range. When the range is 4 miles, MNDR costs 0.076 seconds. Processing a range of 8 miles requires 0.2 seconds by using MNDR compared with 0.65 seconds when using the baseline algorithm. Additionally, MNDR always consumes about 40 percent of the CPU time compared with the baseline algorithm during query processing. Fig. 11b plots the corresponding number of page accesses. The output corresponds to the CPU output, as well as our complexity analysis results. Assuming the road network is uniformly distributed, the number of edges grows quadratically with the increase of the range. Since these edges must be retrieved from the R*-tree file during query processing, the performance of MNDR is deteriorating proportionally.

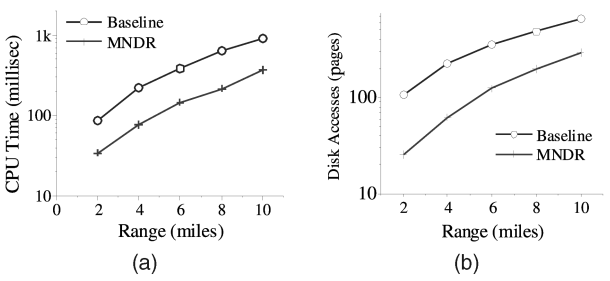

Fig. 11. The performance of MNDR as a function of range.

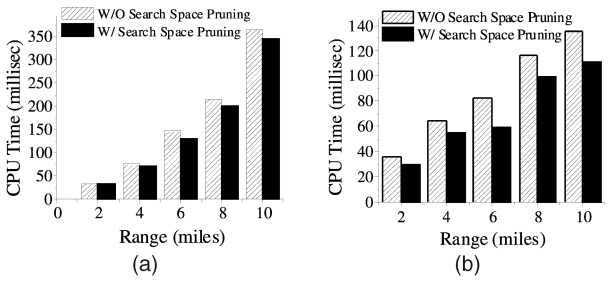

Fig. 12. The CPU time improvement of using Corollary 1. (a) LA County. (b) LA County, Freeways Only.

Next, we are interested in the performance improvement when using Corollary 1 in MNDR. Fig. 12a plots the CPU time when using Corollary 1 to prune the search space in MNDR compared with not using it when handling the LA county data. The performance improvement of using Corollary 1 is about 10 percent when the range is 6.0 miles and less than 5 percent when the range is 2.0 miles. We believe this is largely due to the fact that the TIGER/Line data set consists of many very short road segments (0.1066 miles on average). There are only a few cells that overlap with each edge, which implies that there is little chance to prune some cells during query processing. However, the system improvement by using Corollary 1 is substantial when it is applied to large road segments. To illustrate this fact, we extracted the freeway segments in LA county (the average length of road segments is 2.7127 miles) and performed the simulation on just this network. Fig. 12b shows the results, with query ranges from 2.0 up to 10.0 miles. The results indicate that the improvement of system throughput by applying Corollary 1 is very noticeable. Especially, when the range is 6 miles, the system performance achieves a gain of over 30 percent. Hence, we conclude that for a network with long road segments, it is very appealing to use Corollary 1 to prune the search space.

We also studied the performance of MKNN. Figs. 13a and 13b illustrate the CPU time and disk accesses of MKNN as a function of the number of cells, respectively. An observation is that the throughput of MKNN is relatively stable when the number of cells per axis exceeds 400. This is because when we use a fairly large cell size (e.g., 200 cells per axis), the grid index only provides a very coarse-grained space partition. Hence, the progressive probe results in a space with many unnecessary road segments, which must be retrieved in later steps. Fig. 13a also shows that MKNN performs better than S-GRID with various cell

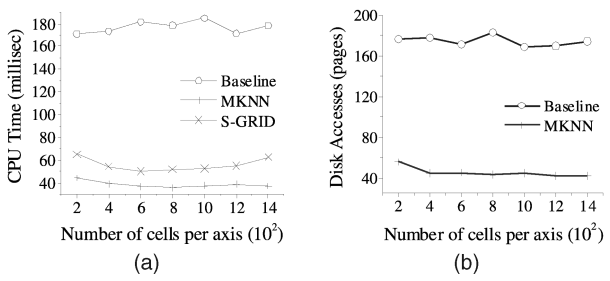

Fig. 13. The performance of MKNN as a function of the number of cells.

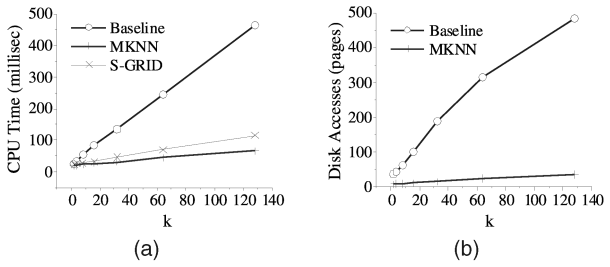

Fig. 14. The performance of MKNN as a function of k.

sizes. When the cell size is relatively large, the performance of S-GRID becomes even worse than MKNN. This is because S-GRID records the distances between each pair of a vertex and a border point in the same cell. When the cell size is large, a cell contains a significant number of vertices, especially for a dense network. Therefore, the search on the *Vertex-Border* component becomes cumbersome without the support of any index structure, which ultimately affects the system performance. Finally, MKNN consistently requires less CPU time and disk accesses than the baseline algorithm.

Fig. 14a plots the performance of MKNN with respect to k. The CPU time grows proportionally with k. More importantly, MKNN outperforms both the baseline algorithm and S-GRID. The growth of the CPU time in MKNN is much slower than that of the baseline algorithm and S-GRID as a function of k. MKNN costs less than 80 percent of the CPU time of the baseline algorithm where $k = 128$. Similary, MKNN requires about 50 percent of the CPU time of S-GRID when $k = 128$. Fig. 14b shows the disk accesses of both algorithms. The gradient of the MKNN curve is very small, which suggests that with the increase of k, the progressive probe in MKNN significantly avoids excessive I/O operations on the R*-tree. Finally, when $k = 128$, the CPU time in LA county is less than 0.5 seconds. This clearly shows that MOVNet can support a very large value of k.

Fig. 15a illustrates the CPU time of MKNN as a function of the number of POIs. The result shows that the CPU time is inversely proportional to the number of POIs, which is what we expect from the theoretical analysis. With a larger number of POIs, the performance of MKNN improves. This characteristic ensures that MOVNet is very applicable for use in metro areas. When there are $100K$ POIs in the service area, processing a kNN query with $k = 50$ requires only 26 milliseconds. Another important observation is that MKNN has better system throughput than both the

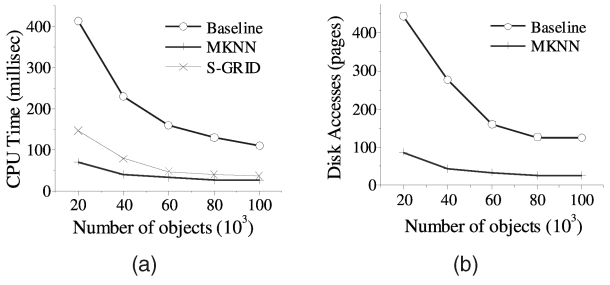

Fig. 15. The performance of MKNN as a function of POIs.

baseline algorithm and S-GRID with varying numbers of POIs. Fig. 15b shows the disk access counts with regard to both MKNN and the baseline algorithm, which correlates with our CPU time measurement result.

Based on our simulation results and analysis, we conclude that the performance of MOVNet scales very well with various settings. In range query processing, the performance difference between MOVNet and the baseline algorithms is much more distinguishable with respect to the disk page accesses, which is due to the usage of the euclidean distance restriction that minimizes the I/O operations. Moreover, our design of the progressive probe results in a good estimation of the result space, hence, the system throughput of MKNN is better than the baseline algorithm and S-GRID.

7 Conclusions

Location-based services have generated growing interest in the research community. This paper presents an infrastructure aimed to process location-based services with moving objects in road networks. We propose a cell overlapping algorithm that quickly relates the underlying network and moving objects in memory. Based on the infrastructure of MOVNet, we present two novel algorithms for processing snapshot range queries and kNN queries, respectively. The experimental evaluation suggests that MOVNet is highly efficient in processing these queries with a real-world, dense road network.

We plan to extend our work in several directions. First, our study currently assumes a static network. However, incorporating some dynamic network updates, such as the real-time traffic information, will be critical for many location-based services, especially those for metro usages. Here, we plan to extend our work to support a dynamic underlying network. Additionally, continuous queries are the most sophisticated query type in location-based services. Although they consume much more computational and memory resources than snapshot queries, they offer an extended view of POI movements and are appealing for monitoring purposes. We are currently extending the functionality of MOVNet to support continuous queries.

Acknowledgments

This research was funded in part by US National Science Foundation grants EEC-9529152 (IMSC ERC), CMS-0219463 (ITR), and IIS-0534761, and NUS AcRF grant WBS R-252-050-280-101/133.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD,* 1990.

[2] J. Bresenham, "Algorithm for Computer Control of a Digital Plotter," *IBM Systems J.,* vol. 4, no. 1, pp. 25-30, 1965.

[3] T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," *GeoInformatica,* vol. 6, no. 2, pp. 153-180, 2002.

[4] H.-J. Cho and C.-W. Chung, "An Efficient and Scalable Approach to CNN Queries in a Road Network," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2005.

[5] H.D. Chon, D. Agrawal, and A.E. Abbadi, "Range and kNN Query Processing for Moving Objects in Grid Model," *Mobile Networks and Applications,* vol. 8, no. 4, pp. 401-412, 2003.

[6] E. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerische Mathematik,* vol. 1, pp. 269-271, 1959.

[7] B. Gedik and L. Liu, "MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System," *Proc. Ninth Int'l Conf. Extending Database Technology (EDBT),* 2004.

[8] H. Hu, D.L. Lee, and V.C.S. Lee, "Distance Indexing on Road Networks," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2006.

[9] H. Hu, J. Xu, and D.L. Lee, "A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects," *Proc. ACM SIGMOD,* 2005.

[10] X. Huang, C.S. Jensen, H. Lu, and S. Saltenis, "S-GRID: A Versatile Approach to Efficient Query Processing in Spatial Networks," *Proc. 10th Int'l Symp. Spatial and Temporal Databases (SSTD),* 2007.

[11] X. Huang, C.S. Jensen, and S. Saltenis, "The Islands Approach to Nearest Neighbor Querying in Spatial Networks," *Proc. Int'l Symp. Spatial and Temporal Databases (SSTD),* 2005.

[12] C.S. Jensen, J. Kolárvr, T.B. Pedersen, and I. Timko, "Nearest Neighbor Queries in Road Networks," *Proc. 11th ACM Int'l Symp. Advances in Geographic Information Systems (GIS),* 2003.

[13] C.S. Jensen, D. Lin, and B.C. Ooi, "Query and Update Efficient B+-Tree Based Indexing of Moving Objects," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2004.

[14] M.R. Kolahdouzan and C. Shahabi, "Continuous K-Nearest Neighbor Queries in Spatial Network Databases," *Proc. Second Workshop Spatio-Temporal Database Management (STDBM),* 2004.

[15] M.R. Kolahdouzan and C. Shahabi, "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2004.

[16] D. Kwon, S. Lee, and S. Lee, "Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree," *Proc. Third Int'l Conf. Mobile Data Management (MDM),* 2002.

[17] M.-L. Lee, W. Hsu, C.S. Jensen, B. Cui, and K.L. Teo, "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2003.

[18] M.F. Mokbel, X. Xiong, and W.G. Aref, "SINA: Scalable Incremental Processing of Continuous Queries in Spatio-Temporal Databases," *Proc. ACM SIGMOD,* 2004.

[19] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias, "Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring," *Proc. ACM SIGMOD,* 2005.

[20] K. Mouratidis, M.L. Yiu, D. Papadias, and N. Mamoulis, "Continuous Nearest Neighbor Monitoring in Road Networks," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2006.

[21] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query Processing in Spatial Network Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2003.

[22] J.M. Patel, Y. Chen, and V.P. Chakka, "STRIPES: An Efficient Index for Predicted Trajectories," *Proc. ACM SIGMOD,* 2004.

[23] S. Saltenis, C.S. Jensen, S.T. Leutenegger, and M.A. Lopez, "Indexing the Positions of Continuously Moving Objects," *Proc. ACM SIGMOD,* 2000.

[24] Y. Tao, D. Papadias, and J. Sun, "The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2003.

[25] G.E. Taylor, G. Blewitt, D. Steup, S. Corbett, and A. Car, "Road Reduction Filtering for GPS-GIS Navigation," *Trans. GIS,* vol. 5, pp. 193-207, 2001.

[26] H. Wang and R. Zimmermann, "Snapshot Location-Based Query Processing on Moving Objects in Road Networks," *Proc. 16th ACM Int'l Conf. Advances in Geographic Information Systems (GIS),* 2008.

[27] H. Wang, R. Zimmermann, and W.-S. Ku, "ASPEN: An Adaptive Spatial Peer-to-Peer Network," *Proc. 13th ACM Int'l Workshop Geographic Information Systems (ACM-GIS),* pp. 230-239, 2005.

[28] X. Xiong, M.F. Mokbel, and W.G. Aref, "SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-Temporal Databases," *Proc. Int'l Conf. Data Eng. (ICDE),* 2005.

[29] X. Xiong, M.F. Mokbel, and W.G. Aref, "LUGrid: Update-Tolerant Grid-Based Indexing for Moving Objects," *Proc. Seventh Int'l Conf. Mobile Data Management (MDM),* 2006.

[30] X. Yu, K.Q. Pu, and N. Koudas, "Monitoring k-nearest Neighbor Queries over Moving Objects," *Proc. Int'l Conf. Data Eng. (ICDE),* 2005.

Haojun Wang received the BS degree in information engineering from Shanghai Jiao Tong University, China, in 1998. He received the MS and PhD degrees in computer science from the University of Southern California, Los Angeles, in 2006 and 2009, respectively. His research interests include spatial data management, peer-to-peer systems, and mobile location-based systems. He is a student member of the IEEE.

Roger Zimmermann received the Informatik Ingenieur HTL degree from the Höhere Technische Lehranstalt in Brugg-Windisch, Aargau, Switzerland, in 1986. He received the MS and PhD degrees in computer science from the University of Southern California, Los Angeles, in 1994 and 1998, respectively. He is currently an associate professor with the Department of Computer Science at the National University of Singapore (NUS), where he is also an investigator with the Interactive and Digital Media Institute (IDMI). Prior to joining NUS, he held the position of research area director with the Integrated Media Systems Center (IMSC) at the University of Southern California (USC). His research interests are in the areas of distributed and peer-to-peer systems, collaborative environments, streaming media architectures, georeferenced video search, and mobile location-based services. He has coauthored a book, four patents, and more than a hundred conference publications, journal articles, and book chapters in the areas of multimedia and information management. He is an associate editor of the *ACM Computers in Entertainment* magazine and the *ACM Transactions on Multimedia Computing, Communications and Applications* journal. He is a member of the ACM and is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.