Snapshot Location-based Query Processing on Moving Objects in Road Networks*

Haojun Wang
Department of Computer Science
University of Southern California
Los Angeles, CA, 90089
haojunwa@usc.edu

Roger Zimmermann
Computer Science Department
National University of Singapore
Singapore 117590
rogerz@comp.nus.edu.sg

ABSTRACT

Location-based services are increasingly popular and it is a key challenge to efficiently support query processing. We present a novel design to process large numbers of location-based snapshot queries on MOVing objects in road Networks (MOVNet, for short). 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. A method to speedily compute the overlapping grid cells in the network relates these two indices. Based on the above features we propose algorithms to support mobile network distance range queries. We demonstrate via experimental results that MOVNet yields excellent performance while scaling to a very large number of moving objects.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Application—*spatial databases and GIS*

Keywords

Location-based Services, Spatial Data Management

1. INTRODUCTION

With the widespread availability of GPS devices more and more people are enjoying location-based services. Various applications, such as road-side assistance and location-aware games, are popular in many urban areas. This trend 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 afforded by car-based or handheld GPS devices 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 example requires a snapshot query, rather than continuous monitoring (which is another class of applications).

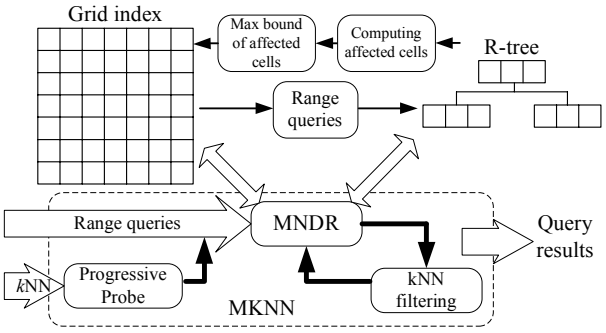

Figure 1: The index structures and query processing modules of MOVNet.

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. Two of the main challenges when supporting POI mobility on an underlying road network are to (a) efficiently manage object location updates and (b) 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). The goal is to efficiently execute snapshot range 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. Figure 1 illustrates MOVNet's 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 bi-directional mapping between the two structures that enables the retrieval of a minimal set of data for query processing. We present an algorithm that speedily form the set of grid cells overlapping with a given edge. The performance of our design has been verified vigorously through simulations, which demonstrates 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

*This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), CMS-0219463 (ITR), IIS-0534761, NUS AcRF grant WBS R-252-050-280-101/133.

ACM GIS '08, November 5-7, 2008. Irvine, CA, USA

and the dual-index design. In the following Section 4 we propose our mobile network distance range query algorithms. A performance evaluation is presented in Section 5. Finally we conclude with Section 6.

2. RELATED WORK

Processing spatial queries in networks has been studied intensely. When POIs are dynamic, the key challenge lies in the large number of location updates that must be managed with an appropriate indexing structure. Movement predictions (i.e., the trajectory of moving objects) have been used with R-tree-based structures (e.g., the *TPR*-Tree* [8]). However, these tree-based indices suffer from excessive node reconstruction costs when performing location updates. Therefore, grid-based structures have raised considerable interest due to their simplicity and efficiency in indexing moving objects. Much of the recent work leverages either an in-memory grid index [3, 5, 11] or an on-disk grid index [4, 10]. Following this trend, our design of MOVNet utilizes an in-memory grid index to manage the location updates of moving POIs.

A number of grid-index based methods have been proposed to process location-based services on moving POIs with Euclidean distances. For instance, Chon et al. [3] first presented an algorithm based on the trajectory of moving POIs overlapping with grid cells to solve snapshot range and kNN queries. SEA-CNN [10] were introduced as centralized solutions with the idea of shared execution to process continuous range and kNN queries on moving POIs. Yu et al. [11] proposed an algorithm (referred to as YPK-CNN) for monitoring 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. [5] presented a solution (CPM) that defines a conceptual partitioning of the space by organizing grid cells into rectangles. However, the above techniques are limited to Euclidean distance computations.

For environments where POIs are dynamic and distances are based on network paths only a few techniques exist. S-GRID [4] was introduced to process kNN queries. A pre-computed structure is maintained with regard to the spatial network data such as to improve the efficiency of query processing. Recently, Mouratidis et al. [6] 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.

3. SYSTEM DESIGN

3.1 Network Modeling and Assumptions

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 $\mathbb{V}$, where $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$. Each edge e is represented as $e(v_1, v_2)$, where v_1 and v_2 are the starting and ending vertex, respectively. 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: (i) the intersections of the network, (ii) the dead end of a road segment, and (iii) 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.

There are different objects (e.g., cars, 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. The location of m is defined as $loc(m)$ = (x_m, y_m), where x_m and y_m are the x and y coordinates of m at current time. A query point $q \in \mathbb{M}$ is a moving object issuing a location-based spatial query at different times. MOVNet assumes that periodic sampling of the moving object positions conveys their locations as a function of time. This method provides a good approximation of the moving object positions.

We define the distance function of two moving objects m_1 and m_2 as $dist(m_1, m_2)$: $loc(m_1) \times loc(m_2) \rightarrow \mathbb{R}^+$. It denotes the shortest path from m_1 to m_2 in the metric of the network distance. 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-Index Structure Design

To record the connectivity and coordinates of vertices in stationary networks, MOVNet utilizes an on-disk R*-tree, in which the edges of the network are stored as MBRs bounded by their vertices. Once the edges are retrieved from disk, a corresponding modeling graph as described in Section 3.1 is constructed in memory for query processing.

A memory-based grid index is used to manage the locations of moving objects [11]. 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 the cells are ordered from the lower left corner of the space). 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. Figure 2 shows an example network (in the form of a modeling graph) that is managed by a grid index of 8 $\times$ 8 cells. An example object on $e(v_2, v_4)$ is enclosed by $c(5, 5)$.

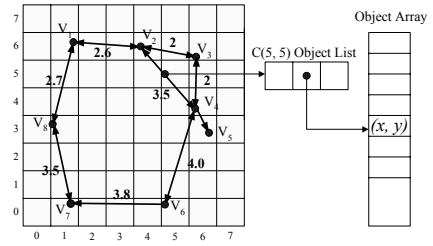

Figure 2: An example network indexed by the grid index and its data storage.

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 so that for an arbitrary edge, we are able to find the set of overlapping cells very quickly. Therefore, we devise an incremental algorithm, which is described in [9] due to space limits. Our experimental results show that our algorithm for computing overlapping cells consumes less than 5% of the query processing time with various settings. This indicates that our method is well suited for online computing. More importantly, by introducing this technique, MOVNet creates a means to bi-directionally map underlying networks and moving object positions.

4. QUERY DESIGN

We propose a Mobile Network Distance Range query algorithm (MNDR) to facilitate the query processing. First, we observe that

only the network data in MOVNet is stored on disk. By leveraging the concept of the *Euclidean distance restriction* [7], 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 overlapping 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. Finally, for each edge whose starting vertex has a distance $\leq d$, MNDR generates the list of overlapping 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 = φ
4: /* Finding the set of edges E', AND vertices V' overlapped by
   the circle with center point q, and radius d */
5: (E', V') = Euclidean-range(q, d)
6: G = Create-modeling-graph(E', V')
7: e = Object-map-matching(q, E')
8: q = Add-vertex-into-graph(G, q, e)
9: S = Compute-distance(G, q, d)
10: for each vertex v in S do
11:    for each edge e outgoing from v do
12:       cellSet = cellSet ∪
          cellOverlapping(e, d − dist(q, v))
13:    end for
14: end for
15: result = Retrieve-objects(cellSet, G)
16: for each object m in result do
17:    e(v1, v2) = Object-map-matching(m, E')
18:    dist(q, m) = min(dist(q, v1) + dist(v1, m),
       dist(q, v2) + dist(v2, m))
19:    if dist(q, m) > d then
20:       result = result - m
21:    end if
22: end for
23: return result
```

Algorithm 1 details MNDR. To illustrate the algorithm with an example, let us assume that the system is processing a network as shown in Figure 2, where the side length of cells is 1.0 unit. A query object q with $dist(q, v2)$ = 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 Figure 3(a) (Line 6). Note that q is inserted as the starting vertex into the modeling graph (Line 8). Next, Dijkstra's algorithm is invoked (Line 9). 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 Figure 3(b). 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 10 - 14, shown as the dark-grey cells in Figure 3(b). After that, the moving objects in $cellSet$ are retrieved from the grid index to constitute the result set. However, several post-processing steps are required to ensure that the distance of each moving object is within range d. First, some cells might overlap with several edges. For instance, $c(6, 6)$ overlaps with $e(v_2, v_3)$ and $e(v_3, v_4)$. Hence for each object in the result set, MNDR determines which edge the object is located on (Line 17) through a map matching process. Second, 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 18). For example, moving objects on edge $e(v_3, v_4)$ have two paths from q ($q \rightarrow v_2 \rightarrow v_3$, and $q \rightarrow v_4$). MNDR will compute the distance of each object via each path, and only use the shortest one. Finally, once the distance from q to the object is determined, MNDR confirms that the distance $\leq d$. For instance, for any object m retrieved from $c(5, 0)$, $dist(q, m) > 3.5$, thus the algorithm removes these objects in Lines 19 - 21.

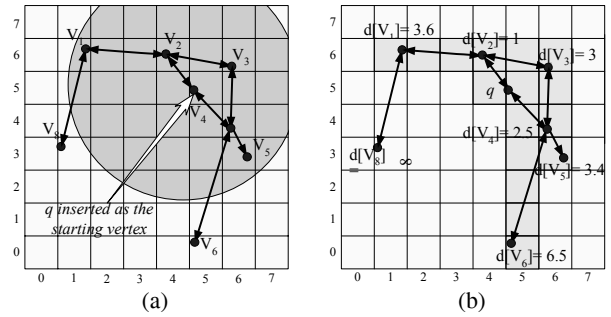

Figure 3: A Mobile Network Distance Range (MNDR) query example.

5. EXPERIMENTAL EVALUATION

5.1 Simulator Implementation

We obtained the Los Angeles County (LA) network data set from TIGER/Line. It has 304,162 road segments distributed over an area of 4,752 square miles. For simplicity, we assume that each road segment is bi-directional. The network data is indexed with an R*-Tree. Each road segment is stored in a MBR bounded by its starting and ending coordinates. Additionally, we used a network simulator [2] to generate the positions of 100,000 moving objects in the road network.

Existing work, such as IMA/GMA, only targets C-kNN query processing. This method differs from the functionality of MOVNet that focuses on snapshot range query processing at its current stage. For performance comparison purposes, we leveraged the concept of $network\ expansion$ [7] to design baseline algorithms in our simulations. We describe the details of our baseline algorithms in [9].

Our Java simulator 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 and we set the page size of the R*-tree to 4KB. 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. After that, the query generator randomly picks a moving object and launches a query from its location. By default, the number of POIs in the network is $50K$; the radius for a range query is 5 miles; and the number of cells per axis is 1,000. 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 metric of the query processing.

5.2 Simulation Results

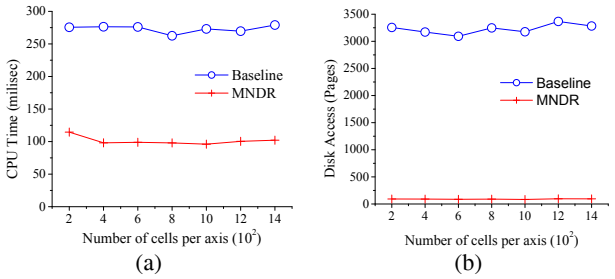

Figure 4: The performance of MNDR as a function of the number of cells

Figure 4(a) illustrates the effect of the number of cells on query processing. The results show that MNDR requires less than half of the CPU time compared with the baseline algorithm. Correspondingly, Figure 4(b) studies the page accesses of both algorithms. As we can see, the baseline algorithm consumes much more page accesses than MNDR. 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 a larger 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.

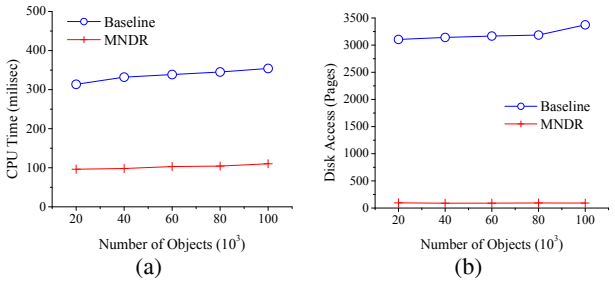

Figure 5: The performance of MNDR as a function of POIs

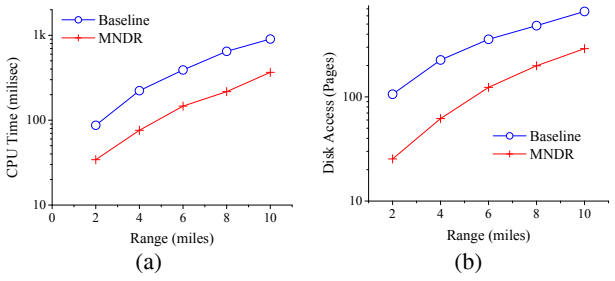

Figure 6: The performance of MNDR as a function of range

Next, Figure 5(a) illustrates the effect of the number of POIs on the execution time of MNDR. As we can see, MNDR outperforms the baseline algorithm with various numbers of POIs. The very small gradient of the MNDR line suggests that MOVNet is very scalable to support a very large number of POIs. Figure 5(b) plots the disk accesses of both algorithms. Similarly to the CPU time results, MNDR performs consistently much lower than the baseline algorithm.

Figure 6(a) plots the CPU time (with logarithmic scale) versus the query range. The CPU time quadratically increases with a larger range. 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% of the CPU time compared with the baseline algorithm. Figure 6(b) plots the corresponding page accesses. The output corresponds to the CPU results.

6. CONCLUSIONS

Location-based services have generated growing interest in the research community. This paper presents an infrastructure aimed at processing 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 a novel algorithm for processing snapshot range queries. The experimental evaluation suggests that MOVNet is highly efficient in processing these queries with various networks.

We are planning to support more query types in MOVNet. An extension to kNN query processing can be found in [9]. More importantly, continuous queries are the most sophisticated query type in location-based services. This functionality is very useful in a number of cases, such as 911 call-centers. We are planning to extend the functionality of MOVNet to support continuous queries.

7. REFERENCES

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

[2] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[3] H. D. Chon, D. Agrawal, and A. E. Abbadi. Range and kNN query processing for moving objects in grid model. *MONET*, 8(4), 2003.

[4] X. Huang, C. S. Jensen, H. Lu, and S. Saltenis. S-GRID: A Versatile Approach to Efficient Query Processing in Spatial Networks. In *SSTD*, 2007.

[5] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *SIGMOD Conference*, 2005.

[6] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, 2006.

[7] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, 2003.

[8] Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In *VLDB*, 2003.

[9] H. Wang and R. Zimmermann. Framework for Snapshot Location-based Query Processing on Moving Objects in Road Networks. Technical Report 08-899, University of Southern California, 2008.

[10] 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.

[11] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE*, 2005.