Materialization-Based Range and k-Nearest Neighbor Query Processing Algorithms[*]

Jae-Woo Chang and Yong-Ki Kim

Dept. of Computer Engineering, Chonbuk National Univ.,
Chonju, Chonbuk 561-756, South Korea
jwchang@chonbuk.ac.kr, ykkim@dblab.chonbuk.ac.kr

Abstract. Recently, the spatial network databases (SNDB) have been studied for emerging applications such as location-based services including mobile search and car navigation. In practice, objects, like cars and people with mobile phones, can usually move on an underlying network (road, railway, sidewalk, river, etc.), where the network distance is determined by the length of the practical shortest path connecting two objects. In this paper, we propose materialization-based query processing algorithms for typical spatial queries in SNDB, such as range search and k nearest neighbors (k-NN) search. By using a materialization-based technique with the shortest network distances of all the nodes on the network, the proposed query processing algorithms can reduce the computation time of the network distance as well as the number of disk I/Os required for accessing nodes. Thus, the proposed query processing algorithms improve the existing efficient k-NN (INE) and range search (RNE) algorithms proposed by Papadias et al. [1], respectively. It is shown that our range query processing algorithm achieves about up to one of magnitude better performance than RNE and our k-NN query processing algorithm achieves about up to 150% performance improvements over INE.

1 Introduction

In general, spatial databases has been well studied in the last two decades, resulting in the development of numerous spatial data models, query processing techniques, and index structures for spatial data [2]. Most of existing work considers Euclidean spaces, where the distance between two objects is determined by the ideal shortest path connecting them. However, in practice, objects, like cars and people with mobile phones, can usually move on an underlying network (road, railway, sidewalk, river, etc.), where the network distance is determined by the length of the practical shortest path connecting two objects on the network. For example, a gas station nearest to a given query q in Euclidean space may be more distant from q in a given network space than any other gas stations. Therefore, the network distance, rather than the Euclidean one, is an importance measure in spatial network databases. Recently, the

[*] This work is financially supported by the Ministry of Education and Human Resources Development (MOE), the Ministry of Commerce, Industry and Energy (MOCIE) and the Ministry of Labor (MOLAB) though the fostering project of the Lab of Excellency.

spatial network databases (SNDB) have been studied for emerging applications such as location-based services including mobile search and car navigation. [3]. Studies on SNDB can also be divided into three research categories, that is, data model, query processing techniques, and index structures. First, Speicys et al. [4] dealt with a computational data model for spatial network. Secondly, Jensen et al. [5] presented k-nearest neighbor (k-NN) query processing algorithms for SNDB. Thirdly, Papadias et al. [1] proposed query processing algorithms for range search, spatial joins, and closest pairs as well as k-NN. Finally, Pfoser and Jensen [6] designed a novel index structure for SNDB. In this paper, we propose materialization-based query processing algorithms for typical spatial queries in SNDB, such as range and k-NN queries. By using a materialization-based technique with the shortest network distances of all the nodes in the spatial network, the proposed query processing algorithms can reduce the computation time of the network distance of two nodes as well as the number of disk I/Os accesses for visiting the nodes. Thus, the proposed query processing algorithms can improve the existing efficient k-NN and range search algorithms proposed by Papadias et al. [1]. This paper is organized as follows. In Section 2, we introduce related work on query processing algorithms for SNDB. In Section 3, we present the architecture of underlying storage and index structures for SNDB. In Section 4 and 5, we propose materialization-based range and k-NN query processing algorithms, respectively. In Section 6, we provide the performance analysis of our k-NN and range query processing algorithms. Finally, we draw our conclusions and suggest future work in Section 7.

2 Related Work

In this section, we overview related work on query processing algorithms for spatial network databases (SNDB). First, Jensen et al. described a general framework for k-NN queries on moving objects in road networks [5]. The framework includes a data model and a set of concrete algorithms needed for dealing with k-NN queries. The data model captures road networks and data points with continuously changing locations. It encompasses two data representations. The detailed two-dimensional representation captures the geographical coordinates of the roads and moving objects. The more abstract graph representation captures the road and moving objects in a form that enables k-NN queries to be answered efficiently by using road distances instead of Euclidean distance. The algorithms for k-NN queries employ a client-server architecture that partitions the NN search. First, a preliminary best-first search for a nearest-neighbor candidate (NNC) set in a graph is performed on the server. Secondly, the maintenance of the query result is done on the client, which re-computes distances between data points in the NNC set and the query point, sorts the distances, and refreshes the NNC set periodically to avoid significant imprecision. Finally, the combination of NNC search with the maintenance of an active result provides the user with an up-to-date query result.

Next, Papadias et al. proposed a flexible architecture for SNDB by separating the network from the entity datasets [1]. That is, they employ a disk-based network representation that preserves connectivity and location, while spatial entities are indexed by respective spatial access methods for supporting Euclidean queries and dynamic updates. Using the architecture, they also developed two frameworks, i.e., Euclidean restriction and network expansion, for each of the most common spatial queries, i.e., nearest neighbors, range search, closest pairs, and distance joins. The proposed algorithms expand conventional query processing techniques by integrating connectivity and location information for efficient pruning of the search space. Specifically, the Euclidean restriction algorithms take advantages of the Euclidean low-bound property to prune the search space while the network expansion algorithms perform query processing directly in the network.

3 Storage and Index Structures for SNDB

Considering a road network, both network junctions and the starting/ending points of a road can be represented as nodes. The connectivity between two nodes can be represented as an edge. Each edge connecting node n_i and n_j includes a network distance $d_N(n_i, n_j)$ which equals the length of the shortest path from n_i to n_j in the network. Most of the existing work on index structures for SNDB focuses on storage structures representing spatial network, especially, storing both the nodes and the edges of the spatial network into a secondary storage. For a fast answer to users' spatial queries, however, it is necessary to efficiently index the spatial network itself as well as objects residing on the spatial network. The objects on the spatial network can be divided into two types according to their mobility, such as points of interest (POIs) and moving objects. To design our storage and index structures for SNDB, we make use of the following ideas. The first one is to differentiate the underlying network from POIs and moving objects. This separation has a couple of advantages. First, dynamic updates in each dataset can be handled independently. Secondly, new/existing datasets can be added to and removed from the system easily. The other one is to make a special treatment on storing and indexing moving objects' trajectories. Because moving objects are continuously moved on the spatial network, their trajectory information is generally large in size. To answer users' spatial query, the support for partial match retrieval on moving objects' trajectories is required. Based on the two main ideas, we design the architecture for storing and indexing spatial network data, point of interests (POIs), and moving objects in SNDB, as shown in Figure 1.

First of all, for the spatial network data, we design a spatial network file organization for maintaining both nodes and edges. For nodes, the node-node matrix file is used to store all the network distance $d_N(ni, nj)$ between node ni and node nj and the node adjacent information file is used to maintain the connectivity between nodes. Both the node ID table and the hash table are used to gain fast accesses to the information of a specific node. For edges, the edge information file is used to store the edge information as well as to maintain POIs residing on an edge. The edge R-tree is used to locate edges rapidly for answering spatial queries. Secondly, we design a POI storage organization for POIs, like restaurants, hotels, and gas stations. The POI information file is used to store the information of POIs and its location in the underlying road network. The POI B+-tree is used to have fast accesses to the information of a specific POI. The edge R-tree is also used to find which edge a specific POI is covered by. Finally, for moving objects, such as cars, persons, motorcycles, etc., we design an object trajectory signature file to have fast accesses to the trajectories of a given moving objects. The architecture supports the following main primitive operations for dealing with SNDB. (i) find_edge(p) outputs a set of edges that covers a point p by performing a point location query on the network R-tree. If multiple edges cover p, the first one found is returned. This function is applied whenever a query is issued, so as to locate an edge which the query point is covered by. (ii) find_points(e) returns a set of POI points covered by the edge e. Specifically, it finds all the candidates points that fall on the MBR of e, and then eliminates the false hits using the edge information file. (iii) compute_ND(p1,p2) returns the network distance d_N(p1, p2) of two arbitrary points p1, p2 in the network. This can be achieved in a fast way by accessing the node-node matrix file incorporated into our architecture via the hash table.

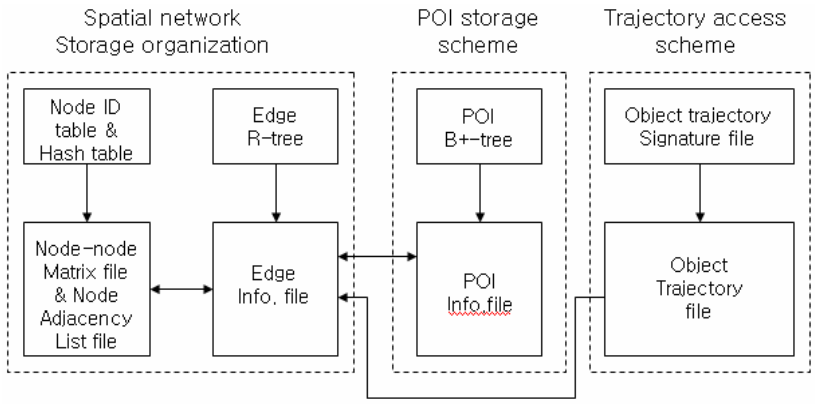
Spatial network Storage organization
POI storage scheme
Trajectory access scheme
Node ID table & Hash table
Edge R-tree
POI B+-tree
Object trajectory Signature file
Node-node Matrix file & Node Adjacency List file
Edge Info. file
POI Info.file
Object Trajectory file

Fig. 1. Storage and index structures for SNDB

4 Materialization-Based Range Query Processing Algorithm

A range query processing algorithm for SNDB is quite different from the conventional ones proposed for the ideal Euclidean space [7] because objects can usually move only on the underlying network. For instance, suppose a query to find gas stations within 10Km from q in Figure 2. The results to satisfy the query in the Euclidean space are p1, p2, p3, and p4 while only p2 can satisfy the query in the network space. To design a range query processing algorithm in SNDB, it is possible to simply apply into the spatial network the conventional algorithms being proposed in Euclidean space [1]. But, the Euclidean restriction algorithm, called RER, generally requires a large number of disk I/O accesses to answer a range query in the underlying network. To remedy this problem, the network expansion algorithm, called RNE, was proposed [1], where it performs network expansion starting from an edge covering a query and determines if objects encountered are within a given range. However, both the RER and the RNE are inefficient where there are lots of roads, being represented as lines, and lots of intersections cross them, being represented as nodes, in spatial networks. This is because they require a lot of the computation time of network distance between a pair of nodes and the number of disk I/Os accesses for visiting nodes.

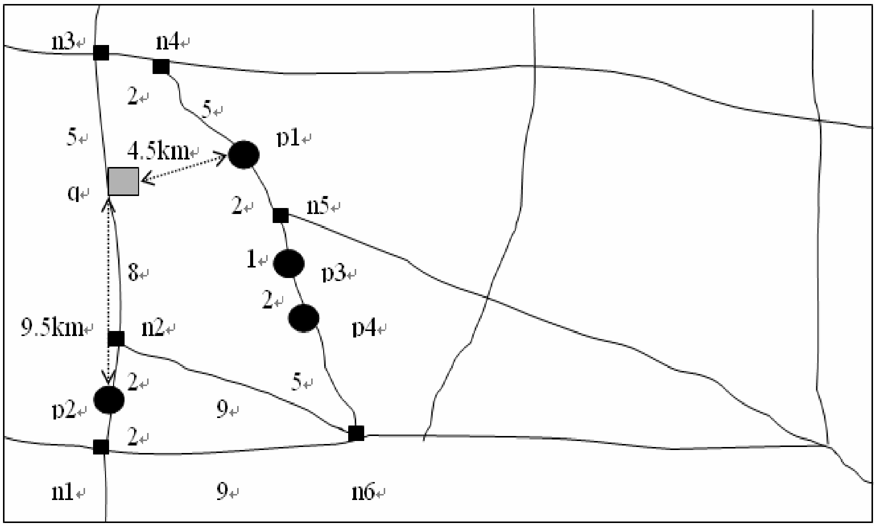

Fig. 2. Range search in Euclidean spaces and spatial networks

```
Algorithm Range(q,r) /* q is the query point and r is the net-
work distance */
1.  Estimate D, the density of POIS in a circle made by r from q
2.  result = ∅
3.  if( D  ≤ threshold_value ) {
4.           PE = Euclidean-range(q,e)
5.           for each point p in PE {
6.              dN(q,p) = compute_network_dist(q, e(ni,nj), p)
7.              if($d_N$(q,p) ≤ r) result = result ∪ p  }
8.  } else {
9.     e(ni,nj) = find_edge(q)
10.    EN = expand-network(e(ni,nj)) // EN is a set of edges in
the expanded network
11.    for each edge e(n,m) in EN {
12.          PS = set of POIs covered by e(n,m)
13.          for each p in PS {
14.             dN(q,p) = compute_network_dist(q, e(ni,nj), p)
15.             if(dN(q,p) ≤ r) result = result ∪ p  }
16.    } //end of for
17. } //end of if else
End Range

Function compute_network_dist(q, e(ni,nj), p) /* q is a query
point, p a target POI, and e(ni,nj) an edge covered by q */
1.  e(nk,nl) = find_edge(p)
2.  return dN(q,p)=min{dN(ni,nk)+dN(ni,q)+dN(nk,p),
                  dN(ni,nl)+dN(ni,q)+dN(nl,p),
                  dN(nj,nk)+dN(nj,q)+dN(nk,p),
                        dN(nj,nl)+dN(nj,q)+dN(nl,p)}
End compute_network_dist
```

Fig. 3. Our materialization-based range query processing algorithm

To remedy it, the network distance computation should be facilitated by the materialization of pre-computed results one time [8], so as to efficiently answer the most common spatial queries in SNDB, such as range and k-NN queries. A critical disadvantage of maintaining all the network distances is to require a huge storage space. For example, we assume that the number of nodes in the network is 200,000 and one network distance is stored with four bytes, we require 160GB (=200K*200K*4) to store all the network distances. Currently, Maxter and Segate Inc. offer the hard disk drivers (HDDs) of 500GB in capacity [9]. Thus, it is possible to maintain all the network distances requiring a huge storage capacity in a disk. A record RMi for a node Ni in the node-node matrix file is RMi = <dist(Ni,N1), … dist(Ni,Nj) … dist(Ni,Nn)> where dist(Ni, Nj) is the shortest network distance between Ni and Nj. Based on our node-node matrix file, we propose a materialization-based range query processing algorithm where the pre-computation of the shortest paths between all the nodes in the network is performed beforehand, as shown in Figure 3.

5 Materialization-Based k-NN Query Processing Algorithm

In SNDB, a k-NN query processing algorithm for SNDN is quite different from the conventional ones which were proposed under the assumption that objects moves on the ideal Euclidean space [10]. As a result, the nearest neighbor of a query in the Euclidean space may be not the case in a spatial network. Figure 2 show an example of the case. That is, the nearest neighbor of q is p1 in the Euclidean space and the distance between q and p1 is 4.5Km. However, the nearest neighbor of q is p2, not p1, in the underlying network and the network distance is 10Km because there is no direct path between q and p1. To design a k-NN query processing algorithm in SNDB, it is possible to simply apply the conventional algorithms proposed for Euclidean space into the spatial network [1]. However, the Euclidean restriction algorithm, called IER, generally searches a large number of Euclidean nearest neighbors to find the network nearest neighbors, thus leading to a large number of disk I/O accesses to answer a k-NN query. To remedy this problem, the network expansion algorithm, called INE, was proposed [1]. The algorithm performs network expansion starting from a query and examines objects in the order that they are encountered until finding k-nearest neighbors. The algorithm computes the network distance of k-nearest neighbors from q and terminates when the network distance of the first node in a queue is grater than $d_{SN}(q, p)$. The IER and the INE have its own different approach to find nearest neighbors in a spatial network. That is, the IER first locates Euclidean nearest neighbors in a global manner and then compute their network distances from a query. On the contrary, the INE first locates an edge covering a query, and then expand the network starting from the edge, in a local manner. Thus, the IER performs well where lots of parallel roads intersect others in an orthogonal way, like Manhattan of New York City, while the latter algorithm performs well where roads are made avoiding obstacles, like a mountain area. Therefore, we need to consider the following for the effective integration of the two algorithms.

Consideration 1: To acquire the actual k-nearest neighbors of q efficiently in the underlying network, the initial set of near-optimal candidates for k nearest neighbors should be obtained.

Consideration 2: Once the initial set of near-optimal candidates for k nearest neighbors has obtained, the final k-nearest neighbors of q in the underlying network should be obtained using the initial set, as rapidly as possible.

```
Algorithm K-NN(q, k)  /* q is the query point */
1. Determine k', the number of initial Euclidean k-NNs and c',
the number of edge connection from q for acquiring initial can-
didates, depending on k.
2. e(ni,nj) = find_edge(q)
3. {p1,…,pk'} = Euclidean-NN(q,k')  // k' < k
4. for each pi
     dN(q,pi) = compute_network_dist(q, e(ni,nj), pi)
5. sort {p1,…,pk'} in ascending order of dN(q,pi)
6. Q = <(ni, dN(q,ni)), (nj, dN(q,nj))> // sorted by distance
7. En = expand_network(e(ni,nj), c', Q)// En is the set of edges
in a network being expanded by edge e(ni,nj) within c' connec-
tions from the edge. Q is updated through network expansion. //
8. SE = find-points(En) // SE is a set of POIs covered by En
9. {p1,…,pk} = the k network nearest neighbors by merging
{p1,…,pk'} and SE sorted in ascending order of their network
distance (pm,…pk may be ∅ if the merged result contains just m-1
points with m ≤ k)
10. dmax = dN(q,pk) // if pk = ∅, dmax = ∞
11. for Pj which is originated from Euclidean-NN (q,k') {
12.   dN(q,pj) = compute_network_dist(q, e(ni,nj), pj)
13.   insert <nj, dN(q,pj)> into Q  }
14.   delele from Q the node n with the smallest dN(q,n)
15.   while(dN(q,n) < dmax) {
16.   for each non-visited adjacent node nj of n {
17.     Sp = find-point(e(nj,n))
18.     update {p1,…,pk} from {p1,…,pk} ∪ Sp
19.     dmax = dN(q,pk)
20.     dN(q,pj) = compute_network_dist(q, e(ni,nj), pj)
21.     insert <nj, dN(q,pj)> into Q  }  // end of for each
22.   delele from Q the next node n with the smallest
dN(q,n)
23. } /* end of while
End K-NN
```

Fig. 4. Our materialization-based k-NN query processing algorithm

To satisfy the first consideration, we are required to build the initial set of candidates for k nearest neighbors by integrating the initial set construction part of the Euclidean restriction algorithm with that of the network expansion one. Because we obtain the initial set of candidates by combining a global initial set and a local initial set, it is possible to acquire an initial set of near-optimal candidates, regardless the characteristic of the underlying road network (Manhattan or a mountain area). To satisfy the second consideration, once obtaining an initial set of near-optimal candidates, we are required to find the final k-nearest neighbors using the network expansion algorithm. This is because we can obtain the final k-nearest neighbors in an efficient way by performing a local network search using the near-optimal candidates. That is, because a global search on the network has been performed to obtain the initial set of candidates, it is sufficient to perform only a local search on the network for the final k-nearest neighbors, thus enabling to achieve a good retrieval performance. In addition, because it takes much time to compute a network distance between two nodes during network expansion, it is required to makes use of our node-node matrix file. This makes it possible to obtain the shortest network distance between any specified node and one of nodes incident an edge covering a query in the fastest way, thus remarkably reducing the network distance computation time and the number of disk I/Os accesses for visiting nodes. We propose a materialization-based k-NN query processing algorithm to satisfy the above considerations, as shown in Figure 4.

6 Performance Analysis

For our experiment, we make use of a road network consisting of 170,000 nodes and 220,000 edges [11]. We also generate 10,846 points of interest (POIs) randomly on the road network by using RunTime21 algorithm [12]. We implement our range and k-NN query processing algorithms under Pentium-IV 2.0GHz CPU with 1GB main memory, running Window 2003.

For the materialization, we use 229G node-node matrix file. For performance analysis, we compare our rage query processing algorithm with RNE and our k-NN query processing algorithm with INE scheme because RNE and INE are considered as the most efficient query processing algorithm [1]. We measure a time for answering a range query whose radius r is between 10 and 200. Figure 5 shows the rage query processing times of RNE and our materialization-based range query processing algorithm (called OMR). The RNE and our OMR require about 0.24 and 0.17 seconds, respectively, when r = 10. In addition, the RNE and our OMR require about 2.25 and 0.24 seconds, respectively, when r = 100. It is shown that our OMR achieves about up to one of magnitude better performance than the RNE and the performance improvement of our OMR over the RNE is increased as the range r is increased. This is because our materialization-based OMR can reduce the network distance computation time and the number of disk I/Os required for accessing nodes by using our node-node matrix file.

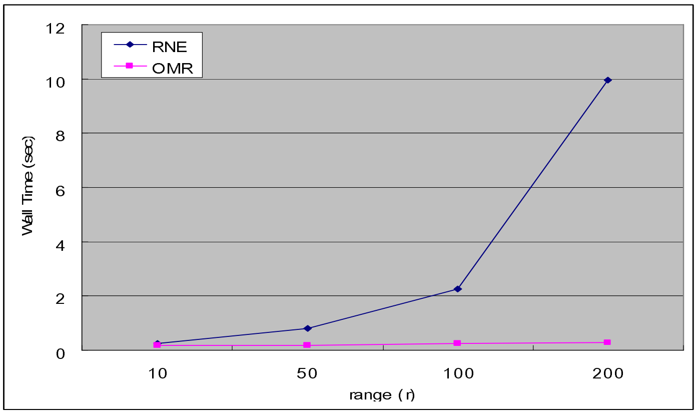

Fig. 5. Range query processing time

Figure 6 shows the k-NN query processing times of the INE and our materialization-based k-NN algorithm (called OMK). The INE and our OMK require about 0.059 and 0.058 seconds, respectively, when k = 1. In addition, the INE and our OMK require about 0.078 and 0.059 seconds, respectively, when k = 10. It is shown that the performance of OMK is nearly the same as that of INE when k=1 and the performance improvement of our OMR over the RNE is increased as k is increased. When k=100, our OMK achieves about up to 150% performance improvements over the INE since the INE and our OMK require about 0.098 and 0.069 seconds, respectively, This is because our materialization-based OMR can reduce the computation time of network distances between a pair of nodes met during network expansion and the number of disk I/Os required for accessing nodes by using our node-node matrix file.

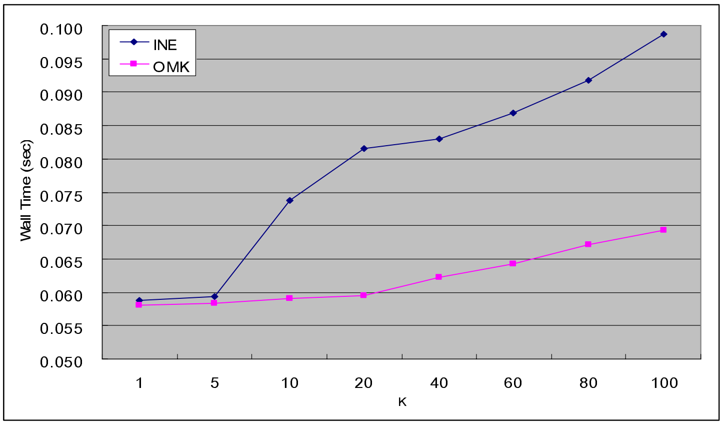

Fig. 6. k-NN query processing time

7 Conclusions and Future Work

In this paper, we designed efficient storage and index structures for spatial network databases (SNDB). Based on the structures, we proposed materialization-based range and k-NN query processing algorithms for SNDB. By using our node-node matrix file containing the shortest network distances of all the nodes in the spatial network, the proposed query processing algorithms can reduce the computation time of the network distances and the number of disk I/Os required for accessing the nodes It was shown that our range query processing algorithm achieved about up to one of magnitude better performance than the RNE and our k-NN query processing algorithm achieved about up to 150% performance improvements over INE. As future work, it is required to study on e-distance join and closest-pairs query processing algorithms for SNDB.

References

1. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query Processing in Spatial Network Databases" Proc. of VLDB, pp, 802-813, 2003.
2. S. Shekhar et al., "Spatial Databases – Accomplishments and Research Needs," IEEE Tran. on Knowledge and Data Engineering, Vol. 11, No. 1, pp 45-55, 1999.
3. J.-W. Chang, J.-H. Um, and W.-C. Lee, "An Efficient Trajectory Index Structure for Moving Objects in Location-Based Services," LNCS 3762, OMT Workshops, pp 1107-1116, 2005.
4. L. Speicys, C.S. Jensen, and A. Kligys, "Computational Data Modeling for Network-Constrained Moving Objects," Proc. of ACM GIS, pp 118-125, 2003.
5. C.S. Jensen, J. Kolář, T.B. Pedersen, and I. Timko, "Nearest Neighbor Queries in Road Networks," Proc. of ACM GIS, pp 1-8, 2003.
6. D. Pfoser and C.S. Jensen, "Indexing of Network Constrained Moving Objects," Proc. of ACM GIS, pp 25-32, 2003.
7. T. Seidl, N. Roussopoulos, and C. Faloutsos, "The R+-tree: a Dynamic Index for Multi-Dimensional Objects, Proc. of VLDB, 1987.
8. N. Jing, Y.-W. Huang, and E.A. Rundensteiner, "Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation," IEEE Tran. on Knowledge and data Engineering, Vol. 10, No. 3, pp 409-432, 1998.
9. http://www.pcworld.com
10. T. Seidl and H. Kriegel, "Optimal Multi-step k-Nearest Neighbor Search, Proc. of ACM SIGMOD, 1998.
11. www.maproom.psu.edu/dcw/
12. T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," GeoInformatica, Vol. 6, No. 2, pp 153-180, 2002.