The Islands Approach to Nearest Neighbor Querying in Spatial Networks

Xuegang Huang, Christian S. Jensen, Simonas Šaltenis

Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, DK-9220, Aalborg, Denmark
{xghuang,csj,simas}@cs.aau.dk

Abstract. Much research has recently been devoted to the data management foundations of location-based mobile services. In one important scenario, the service users are constrained to a transportation network. As a result, query processing in spatial road networks is of interest. We propose a versatile approach to k nearest neighbor computation in spatial networks, termed the Islands approach. By offering flexible yet simple means of balancing re-computation and pre-computation, this approach is able to manage the trade-off between query and update performance. The result is a single, efficient, and versatile approach to k nearest neighbor computation that obviates the need for using several k nearest neighbor approaches for supporting a single service scenario. The experimental comparison with the existing techniques uses real-world road network data and considers both I/O and CPU performance, for both queries and updates.

1 Introduction

An infrastructure is emerging that enables location-based mobile services, and we are witnessing substantial efforts in the research community to establish fundamental data management support for such services. Mobile services typically involve service users and so-called points of interest. We consider the scenario where these are located within a spatial network or, more specifically, a road network [12–14, 17, 21, 29]. The movements of the users, often termed moving objects, are constrained by the network, and the points of interest can only be visited by traveling along the network. The relevant notion of distance is network distance based on shortest-path computation.

Existing approaches to k nearest neighbor (kNN) computation in spatial networks can be divided into two types: approaches that compute kNN queries by incrementally scanning the network until k neighbors are found, and approaches that apply some form of pre-computation and "compute" kNN queries by looking up data collected in pre-computed data structure. Both types of approaches assume that the spatial network is represented by graph-like data structures.

The first type of approach, denoted as "online computation," naturally captures the dynamic aspects of the network, e.g., the emergence or disappearance of points of interest, and applies some form of network expansion-based search. This type of approach is able to output the network distances and paths to each kNN, as these are computed as part of the process. The data structures used in online computation capture the connectivity of the network and are easily updated. When compared to online computation, the second type of approach, termed “pre-computation,” typically has better query performance, but has difficulty in coping with frequent updates of the road network and the points of interest.

We consider the performance of queries as well as updates, as both efficient querying and update are important for location-based mobile services. In particular, we propose a novel approach, termed the Islands approach, to kNN processing in spatial networks. This approach computes the kNNs along with the distance to each, but does not compute the corresponding shortest paths. The rationale for this design decision is that a mobile user is expected to only be interested in the actual path to one nearest neighbor selected from the kNN result, and so the path computation is better left to a subsequent processing step.

The Islands approach is designed with the assumption that the overall I/O cost of queries and updates is the main performance evaluation criterion, and the approach aims to be efficient for varying frequencies of queries and updates, which yields broad applicability. The versatility of the approach is demonstrated by an experimental comparison with two other approaches that covers the cases these two are optimized for.

The paper makes three main contributions.

– The Islands approach offers an attractive generalization of the existing kNN query processing techniques for spatial networks. It employs a relatively simple data structure and an intuitive search algorithm. And it is applicable to a broad range of mobile service scenarios, thus avoiding the need for using more specialized algorithms for different scenarios.
– The Islands approach offers a direct and elegant way of controlling the amount of pre-computation performed and thus also the trade-off between query and update performance. This enables the approach to accommodate varying densities of points of interest and varying query versus update frequencies.
– The paper presents an experimental evaluation that is significantly more comprehensive than previous evaluations. Specifically, this is the first evaluation that covers both online computation and pre-computation and hat considers both query and update performance in a setting with real road network data. The paper thus offers new insight into relative merits of the existing approaches.

In Section 2, we proceed to introduce related work. Section 3 presents the Islands approach and its variations. This is followed by a section that compares the Islands approach with existing kNN techniques. Section 5 then presents the empirical performance study that characterizes the Islands approach as well as compares it with the existing algorithms. The last section summarizes and offers directions for future research.

2 Related Work

Nearest neighbor computation is a classical topic. Many existing algorithms assume an indexing structure, e.g., an R-tree, and search in a branch-and-bound manner [10, 18]. Many extensions and applications of kNN computation have also been proposed [1, 5, 11, 15, 20, 23, 24, 27, 28].

Query processing for objects moving in spatial networks, e.g., cars moving in road networks, has also received attention recently. However, most existing spatial query processing techniques cannot be applied directly in this setting, one reason being that the distance between two locations in a spatial network is the length of the shortest path in the network between these rather than being the Euclidean distance.

This paper assumes a specific data model and disk-based data structure for a spatial network and its associated data points as the foundation for its proposed algorithms. Among the several data models and data structures available [4, 6, 7, 19, 25], we adopt a fairly standard graph-based data model and structure [7, 19] so that the algorithms are generally applicable.

We consider several existing disk-based data structures for shortest path computation and general query processing in spatial networks [8, 17, 22]. The CCAM structure [22] aims to support network computations such as route evaluation and aggregate queries. In this structure, a two-way partition algorithm [3] is adapted to partition the spatial network and then arrange network nodes into disk pages. Another algorithm for partitioning a road network is proposed by Huang et al. [8], and Papadias et al. [17] propose a network storage scheme for supporting both network-based and traditional Euclidean-distance-based spatial query processing. Our storage scheme enhances this scheme to capture additional aspects of real-world road networks.

To provide a thorough discussion of the existing techniques for kNN computation in spatial networks, and to compare them in detail to the Islands approach, we defer consideration of these works to Section 4.

3 The Islands Approach

Following a definition of the assumed transportation network model, concepts and observations related to the use of islands are presented. Section 3.3 presents an algorithm for kNN computation based on islands, and Section 3.4 covers several extensions to the algorithm.

3.1 Transportation Networks and Query and Data Points

We consider location-based mobile services in road networks as our application scenario. In this scenario, mobile service users are moving in a road network. A number of facilities or so-called points of interest, e.g., gas stations or supermarkets, are located within the road network. We define the network distance between a user and a point of interest as the length of the shortest path from the users' current location to the point of interest. A k nearest neighbor query issued by a service user will return the k nearest points of interest to the user based on the network distance. Using *query point* to denote a user and *data point* to denote a point of interest, we proceed to model the elements of the network scenario.

A *road network* is defined as a two tuple $RN = (G, co\mathcal{E})$, where G is a directed, labeled graph and $co\mathcal{E}$ is a binary, so-called co-edge, relationship on edges. Graph G is given by $G = (V, E)$, where V is a set of vertices and E is a set of edges. Vertices model intersections and the starts and ends of roads. An edge e models the road in-between two vertices and is a three-tuple $e = (v_s, v_e, l)$, where $v_s, v_e \in V$ are, respectively, the start and the end vertex of the edge. The edge can be traversed only from v_s to v_e. The element l captures the travel length of the edge. Two edges e_i and e_j are in the co-edge relationship ($(e_i, e_j) \in co\mathcal{E}$), if and only if they represent the same bi-directional part of a road for which U-turn is allowed.

Next, a *location* on the road network is a two tuple $loc = (e, pos)$ where e is the edge on which the location is located and pos represents the distance from the starting vertex of the edge to loc.

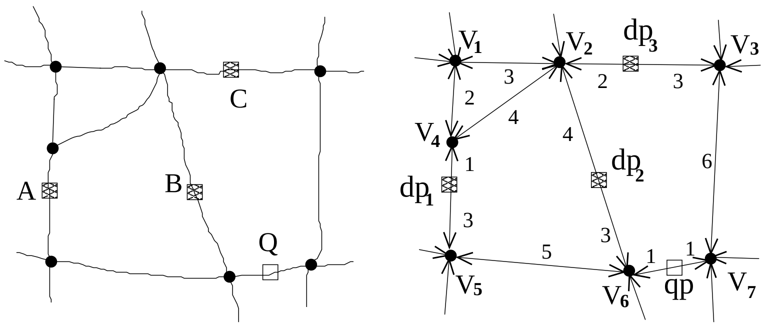


Fig. 1. Road Network Model

A *data point* is modeled as a set of locations, i.e., $dp = \{loc_1, \cdots, loc_k\}$. Note that adding and removing data points or their locations does not affect the road network itself, which is important for maintainability in practice. A *query point* qp is modeled as a location.

An edge with start vertex v_i and end vertex v_j is denoted by $e_{i,j}$. Figure 1 illustrates the concepts defined above, e.g., edge $e_{1,4} = (v_1, v_4, 2)$, data point $dp_1 = \{(e_{4,5}, 1), (e_{5,4}, 3)\}$, and query point $qp = (e_{7,6}, 1)$.

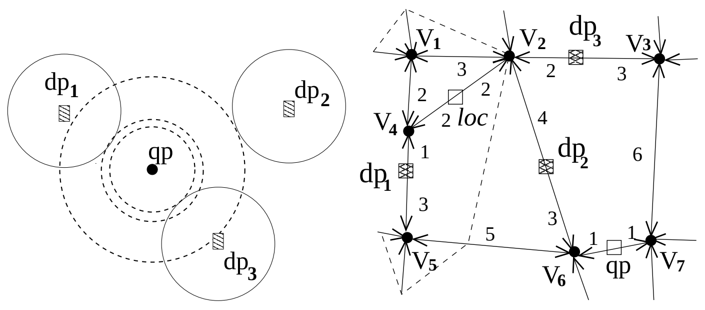
The example road network in Figure 1 is assumed to have only bi-directional roads with no u-turn restrictions and each data point has two positions—one on each side of a bi-directional road. The remainder of the description of the Islands approach is carried out under these assumptions.

3.2 Observations

Intuitively, an incremental expansion process starting from the query point can be used to find the k nearest data points in Euclidean space. To optimize the search process, one can "enlarge" each data point into a big circle—see Figure 2(a)—so that the expansion process will terminate early. As shown in the figure, data point dp_3 will be found as the nearest neighbor, dp_1 is the second-nearest neighbor and dp_2 is the third-nearest neighbor. After touching the border of dp_2, the 3NN search process can stop.

In a road network, given a distance value r, the *island* of a data point dp is the subset of the road network covered by a network expansion from dp with the range r. We define r as a *radius* of this island. Intuitively, all vertices with distance to dp less than or equal to the radius belong to the island. We denote these vertices as *the island's vertices*. A vertex of an island is an *internal vertex* of the island if all its neighboring vertices are vertices of the same island. A vertex of an island is a *border vertex* of this island if at least one of its neighboring vertices does not belong to this island. All the edges connecting the island's vertices are *the island's edges*. A location (or, a query point) in the road network is *inside* an island if its network distance to the data point of this island is less than or equal to the radius of the island.

As illustrated in Figure 2(b), for the part of the road network belonging to the island of dp_1 with a radius of 5, v_4 is an internal vertex and v_1, v_2, and v_5 are border vertices. The location $loc = (e_{4,2}, 2)$ is inside this island.



(a) Euclidean Islands (b) Network Island

Fig. 2. Observations on Islands

To record information about islands, each vertex in the network stores references to all the data points that are centers of the islands covering the vertex. The distance from the vertex to the data point is stored with each such reference. Then, similar to the Euclidean case in Figure 2(a), the network expansion process of a kNN query will be reduced, since a data point can be declared to be found when the expansion process visits a border vertex of this data point's island. If all islands have the same radius, and a query point is already inside l islands, the data points corresponding to these islands are the l nearest neighbors of the query point. The distances from the query point to these l neighbors are found from the above-mentioned pre-computed distances.

In general, the kNN search process includes two steps. First, we need to check the islands covering the query point. Second, if the number of such islands is smaller than k, a network expansion is needed to find additional islands.

If the islands have different radiuses, the islands approach uses the minimum radius, r_{min}, i.e., all data points are assumed to have islands with radius r_{min} (no larger than the islands they actually have). As will be explained later, having different island radiuses brings flexibility to the Islands approach. Specifically, the kNN query performance and the update efficiency can be controlled by changing the radiuses of the islands in different regions of the road network. We proceed to describe the Islands approach in more detail.

3.3 Islands-Based kNN Algorithm

The Islands approach consists of a pre-computation component and an online network-expansion component. The pre-computation component stores, for each vertex in the road network, references to the islands that cover the vertex and the network distances from the vertex to the data points that generate the islands. With this component, the network expansion, denoted as $IslandExpansion(qp, k)$, first checks the islands that the query point qp is inside and maintains the data points found in a priority queue, then starts a network expansion process from qp to find borders of new islands. The network expansion process terminates when the sum of the expansion radius and the minimum radius of all pre-computed islands exceeds the distance from the query point qp to the kth data point in the priority queue.

We proceed to describe $IslandExpansion(qp, k)$ algorithm in the following. It is similar to the INE algorithm [17], which in turn is a modified Dijkstra's single source shortest paths algorithm. Two priority queues, Q_{dp} and Q_v, are used in the algorithm to record the covered data points and vertices together with their distances to the query point, denoted as $d(qp, dp)$ and $d(qp, v)$. Both queues sort elements by the distance value and do not allow duplicate data points or vertices. The size of Q_{dp} is limited to k elements.

We introduce *update* and *deque* operations for the two queues. The *update*(dp/v, *dist*) operation inserts a new data point or vertex and the corresponding distance into the queue. If this data point or vertex is already in the queue then, if *dist* is smaller than the distance stored in the queue, the distance value in the queue is updated to *dist*. The *deque* operation removes and returns the vertex with the smallest distance. Suppose the minimum radius of all islands is r_{min}. The pseudo code of $IslandExpansion$ is given below. Queues Q_v and Q_{dp} are assumed to be empty initially.

```
(1)  procedure $IslandExpansion(qp, k)$
(2)    for each data point dp on edge qp.e: $Q_{dp}$.update(dp, $d(qp, dp)$)
(3)    $Q_v.update(qp.e.v_s, d(qp, dp.e.v_s)), Q_v.update(qp.e.v_e, d(qp, qp.e.v_e))$
(4)    for each dp, if its island covers $qp.e.v_s$ or $qp.e.v_e$: $Q_{dp}$.update(dp, $d(qp, dp)$)
(5)    if $\exists a$, such that $(a, qp.e) \in co\mathcal{E}$, do lines (2)–(4) assuming $qp = (a, a.l - qp.pos)$
(6)    Let $dp_k$ denote the $k$-th element in $Q_{dp}$, or $dp_k = \varnothing$, if there is no such element
(7)    $d_k \leftarrow d(qp, dp_k)$ // $d_k \leftarrow \infty$ if $dp_k = \varnothing$
(8)    $v \leftarrow Q_v$.deque, mark $v$ visited
(9)    while $d(qp, v) + r_{min} < d_k$
(10)     for each non-visited adjacent vertex $v_x$ of $v$
(11)       $Q_v$.update$(v_x, d(qp, v_x))$ // $d(qp, v_x)$ assumes the path $qp \to \cdots \to v \to v_x$
(12)       for each dp, the center of an island covering $v_x$: $Q_{dp}$.update(dp, $d(qp, dp)$)
(13)     $d_k \leftarrow d(qp, dp_k)$
(14)     $v \leftarrow Q_v$.deque, mark $v$ visited
(15)   return $Q_{dp}$
```

Note that in line 12 of the algorithm, $d(qp, dp) = d(qp, v_x) + d(v_x, dp)$, where $d(qp, v_x)$ is taken from Q_v and $d(v_x, dp)$ is the pre-computed distance stored with v_x. Analogous computation of $d(qp, dp)$ is also performed in line 4.

To see how the algorithm works, consider Figure 2(b) and let all three data points have islands with radius 6. Starting from the query point $qp = (e_{7,6}, 1)$, the algorithm $IslandExpansion(qp, 2)$ first adds vertices v_6 and v_7 to Q_v ($Q_v = \langle(v_6, 1), (v_7, 1)\rangle$). Then it checks the islands covering v_6 and v_7, and data point dp_2 is found. Starting with v_6, the expansion process finds the islands of dp_1, dp_2 and dp_3 through the adjacent vertices v_5 and v_2. Thus, $Q_{dp} = \langle(dp_2, 4), (dp_1, 9)\rangle$ and $Q_v = \langle(v_7, 1), (v_5, 6), (v_2, 8)\rangle$. At the next step, since $r_{min} = 6$ and the distance d_2 from the query point to the 2nd NN is 9, only vertex v_7 in Q_v needs to be checked (based on the while-loop criteria in line 9). Finally, dp_2 and dp_1 are the two data points returned. It can be observed that using the pre-computed information, the network expansion finds the data points dp_1 and dp_3 before reading the edges they are located at.

If $k = 1$, the algorithm starting from qp will find dp_2 in the first step. Since $r_{min} = 6$ and the distance from qp to dp_2 is $d_1 = 4$, the algorithm will finish without the network expansion process (since $d(qp, v_6) + r_{min} > d_1$). This, as mentioned in Section 3.2, is always the case if k or more islands cover the query point.

The $IslandExpansion$ algorithm uses disk-based data structures for the network and pre-computed data. Section 4 provides a detailed description of the data structures, and it compares the Islands approach with the existing road network kNN algorithms using examples. We proceed to discuss several extensions of the Islands approach.

3.4 Extensions

An accompanying technical report [9] covers several extensions to the Islands approach, which we proceed to describe briefly. First, *shrink* and *expand* operations are provided that can be applied to any island to change its r_{min}. This offers a basis for balancing the overall query and update performance. The following section describes how these operations can be used to achieve different query and update performance trade-offs in different parts of a road network.

Second, we have seen that islands in different parts of a road network can have different radiuses. Intuitively, the radiuses of islands in urban areas should be relatively smaller than those of islands in rural areas since the densities of points of interest are relatively higher and there are relatively more frequent updates of road network data. When a kNN query is issued close to the border of areas with different r_{min} values, algorithm $IslandExpansion$ can be improved to take into account the different r_{min} values. The accompanying technical report details how to modify the $IslandExpansion$ algorithm to better handle such cross-area expansions.

Third, we extend the Islands approach, make changes to both the pre-computation component and the network expansion algorithm, to accommodate road networks with uni-directional edges, data points that have more than one network location, optional U-turn restrictions, and turn restrictions at intersections.

4 The Islands Approach in Comparison to Existing Techniques

The Islands approach consists of a pre-computation component and a network expansion algorithm. With the *shrink* and *expand* operations and the procedure for handling cross-area expansions, the trade-off between the performance of kNN queries and road-network updates can be controlled. For comparison purposes, we proceed to survey and exemplify the existing kNN algorithms. We also describe the disk-based data structure for the road network and the pre-computed data.

4.1 Online *k* Nearest Neighbor Computation

Intuitively, kNN computation can be done by employing a best-first search through adjacent edges until k neighbors are found. In contrast to the traditional shortest-path algorithms in graph theory, the kNN search in spatial networks has to employ a disk-based data structure for representing the network, the objective being to minimize I/O.

Papadias et al. [17] introduce two algorithms for kNN computation in spatial network, namely Incremental Euclidean Restriction (IER) and Incremental Network Expansion (INE). As they show that the INE algorithm outperforms the IER algorithm, we focus on the INE algorithm. This algorithm performs incremental network expansions from the query point and examines data points in the order they are encountered during the expansion process. The INE algorithm is an adaptation of Dijkstra's single source shortest paths algorithm on graphs. It terminates when the expansion's range exceeds the network distance to the kth nearest neighbor. It can be seen as a special case of the Islands approach where each data point's island has a radius of 0.

The performance of the INE approach depends on the density of the data points. Intuitively, for a large road network with only few data points, the expansion process of the INE algorithm will have to scan large parts of the road network until enough data points are collected.

4.2 kNN Pre-Computation Approach

To reduce the cost of network expansion in queries, pre-computation techniques can be applied. Shahabi et al. [21] introduce a technique to transform a road network to a high dimensional space in which simpler distance functions can be used. However, this transformation requires pre-computation of the network distances between all pairs of vertices in the road network, which is often impractical.

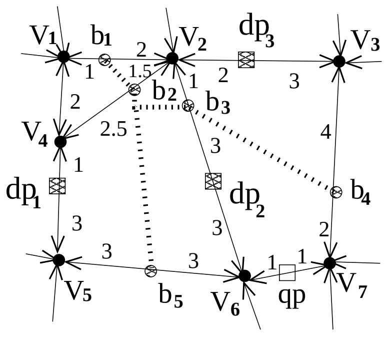


Fig. 3. Network Voronoi Diagram

Kolahdouzan and Shahabi [13] recently proposed the so-called VN3 technique for kNN computation in road networks. Starting from each data point, VN3 first creates a Network-Voronoi-Diagram [16], then pre-calculates the network distances within each Voronoi polygon. The network expansion within each Voronoi polygon can then be replaced by a look-up over the pre-computed distances.

Consider a Network-Voronoi-Diagram constructed for the example road network in Figure 1. As shown in Figure 3, the Voronoi polygon of dp_2 contains border points b_3, b_4, and b_5. Using the pre-computed information, a 2NN query from the query point qp first finds that qp is inside the Voronoi polygon of dp_2. Thus, dp_2 is its nearest neighbor. Then the network distance from qp to b_3, b_4, and b_5 can be found by look-up in a pre-computed distance table. To find the next nearest neighbor, the VN3 approach generates a candidate set consisting of "adjacent" data points, i.e., data points whose Voronoi polygons are adjacent to dp_2's polygon. Thus, dp_1 and dp_3 are included in the candidate set. Then a refinement step is used to find the actual network distance from qp to these candidate data points. Since the distances from border points of dp_1 and dp_3 to these data points are pre-computed, it requires just a look-up process to get the network distance from qp

to dp_1 and dp_3 via the border points b_3, b_4, and b_5. The VN3 approach continues this process until enough kNNs are found.

The VN3 approach excels in query performance when the density of data points is low, but is not efficient in situations when many data points are located in a small network area, e.g., points of interest in a city center. In addition, this approach does not provide a clear way of representing different "types" of data points in the road network. it is possible to construct a "multi-level" structure by constructing Voronoi diagrams for each type of data points, but such multi-level Voronoi-diagrams do not enable efficient processing of the kNN queries for multiple types of data points. An example of such query could be looking for k nearest tourist attractions such as museums, shopping malls, and parks.

4.3 Disk-Based Data Structure

The disk-based data structure for road network data used together with the INE, VN3, and Islands approaches, shown in Figure 4, is an adaptation of the data structures proposed for the INE [17] and VN3 [13] approaches. The structure has six components, the *Vertex-Edge*, *Edge-Data*, *Island-Precomputation*, *Voronoi-Polygon*, *Border-Adjacency*, and *Voronoi-Precomputation* component.

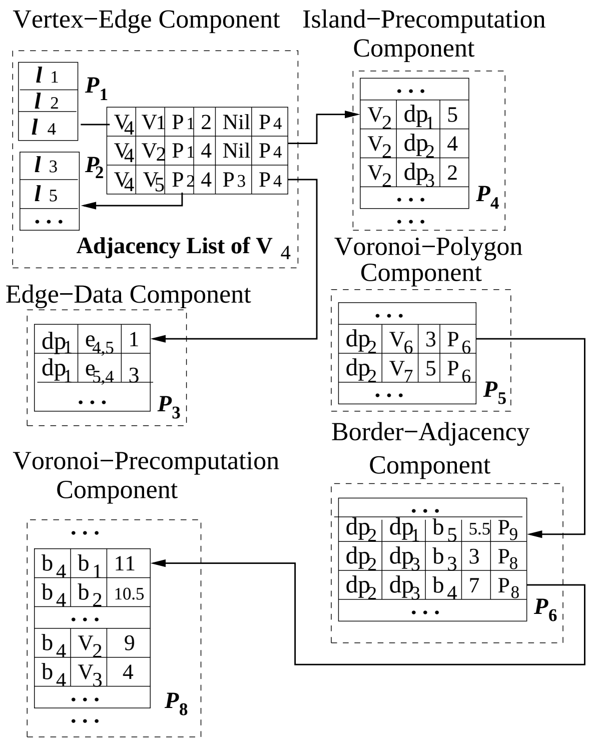


Fig. 4. Disk-Based Data Structure

The road network and data points are represented by the Vertex-Edge and Edge-Data components. Using the example road network in Figure 1, Figure 4 illustrates that the adjacency list l_4 for vertex v_4 is composed of entries standing for edges starting from v_4. The data point dp_1 located on edge $e_{4,5}$ is stored in an entry in the Edge-Data component.

Specifically, in the Vertex-Edge component, each entry denotes an edge and is of the form $(vsID, veID, ptNBVE, L, ptDP, ptI)$. Here, $vsID$ and $veID$ are the id's of the start and end vertices, $ptNBVE$ points to the disk page containing the end vertex, L is the length of this edge, $ptDP$ points to the disk page containing the data points on this edge, and ptI

points to the disk page containing Island-Precomputation data of the end vertex. Pointers are set to Nil if there is no linked page. Entries in the Vertex-Edge component are assigned to pages based on the Hilbert value of the start vertex.

Each entry in the Edge-Data component has the form $(dpID, eID, \mathit{offset})$, where $dpID$, eID denote the data point and edge, and offset is the distance from the start vertex to the data point. We assume the eID value can be obtained from the $vsID$ and $veID$ values in the Vertex-Edge entry. Otherwise, these two attributes are used instead. Both the Vertex-Edge and the Edge-Data components are used in the INE algorithm.

For each vertex, the Island-Precomputation component stores a list containing its distance to related islands. Each entry in the list has the form $(vID, dpID, D)$ where vID and $dpID$ denote the vertex and data point, and D is the network distance between them. These entries are arranged into pages based on the Hilbert values of the vertices. As illustrated in Figure 4, the list of vertex v_2 has three entries, describing its distances to the three data points. The $IslandExpansion$ algorithm uses the Vertex-Edge component, the Island-Precomputation component and the Edge-Data component (only in the first step).

The VN3 approach uses the Voronoi-Polygon, Border-Adjacency and the Voronoi-Precomputation components. The Voronoi-Polygon component stores, for each data point, the vertices inside its Voronoi polygon. This component is used to decide the Voronoi polygon where the query point is located and provides the first nearest neighbor. Each entry has the form $(dpID, vID, D, ptB)$, where $dpID$ denotes the data point generating this Voronoi polygon, vID is the id of a vertex inside the Voronoi polygon, D is their distance, and ptB points to the disk pages of the Border-Adjacency component containing the border points and adjacency information for all the Voronoi polygons. Each entry in the Border-Adjacency component has the form $(dpsID, dpeID, bID, D, ptP)$, where $dpsID$ and $dpeID$ denotes two data points whose Voronoi polygons are adjacent, bID denotes one border point of the two Voronoi polygons, D is the distance from the border point to the two data points, and ptP is the pointer to the disk page containing pre-computed distance values of this border point. The Voronoi-Precomputation component stores, for each border point, its distance to other border points and vertices of the same Voronoi polygons.

We assume that the edge where the query point is located is known before the query so that it can be visited directly. Otherwise, all the edges can be indexed using an R-tree, which can then be used for "map-matching." If the "id" or "name" of the edges can always be revealed for the query, a B^+-tree can be used to index these attributes and provide direct access to edges in the Vertex-Edge component. The whole disk-based data structure for the example road network in Figure 1, consisting of 9 disk pages, is presented in Figure 13 in the appendix. Each island is given a radius of 8. Each attribute value takes 1 unit size, and we set the page capacity to 54 units.

4.4 Example

Based on the example road network, we proceed to exemplify the workings of the INE, VN3, and Islands approaches. We employ an LRU buffer with a size of 2 pages and execute a 2NN query for query point $qp = (e_{7,6}, 1)$. We show the pages in the buffer and the total amount of I/Os for the three approaches. The I/O column denotes the

Approach	Steps	Q_v	Q_{dp}	d_2	Buffer	I/O
INE	1	$\langle(v_6,1),(v_7,1)\rangle$	$\emptyset$	∞	P_2	$1/0$
	2	$\langle(v_7,1),(v_5,6),(v_2,8)\rangle$	$\langle(dp_2,4)\rangle$	∞	P_2,P_3	$2/0$
	3	$\langle(v_5,6),(v_3,7),(v_2,8)\rangle$	$\langle(dp_2,4)\rangle$	∞	P_3,P_2	$2/0$
	4	$\langle(v_3,7),(v_2,8),(v_4,9)\rangle$	$\langle(dp_2,4),(dp_1,9)\rangle$	9	P_2,P_3	$2/0$
	5	$\langle(v_2,8),(v_4,9)\rangle$	$\langle(dp_2,4),(dp_1,9)\rangle$	9	P_2,P_3	$2/0$
	6	$\langle(v_4,9),(v_1,11)\rangle$	$\langle(dp_2,4),(dp_1,9)\rangle$	9	P_1,P_3	$3/0$
Island, $r_{min}=8$	1	$\langle(v_6,1),(v_7,1)\rangle$	$\langle(dp_2,4),(dp_1,9)\rangle$	9	P_2,P_4	$2/0$
Island, $r_{min}=7$	1	$\langle(v_6,1),(v_7,1)\rangle$	$\langle(dp_2,4)\rangle$	∞	P_2,P_4	$2/0$
	2	$\langle(v_7,1),(v_5,6),(v_2,8)\rangle$	$\langle(dp_2,4),(dp_1,9)\rangle$	9	P_2,P_4	$2/0$
	3	$\langle(v_5,6),(v_3,7),(v_2,8)\rangle$	$\langle(dp_2,4),(dp_1,9)\rangle$	9	P_2,P_4	$2/0$

(a) Example of the INE and Island ($r_{min}=7,8$) Approaches

Steps	Candidates	Distances	Results	Buffer	I/O
1	$\emptyset$	$\emptyset$	$\{(dp_2,4)\}$	P_2,P_5	$2/0$
2	$\{dp_1,dp_3\}$	$D(qp,b_4),D(b_4,dp_3),D(qp,b_5),$ $D(b_5,dp_1),D(qp,b_3),D(b_3,dp_3)$	$\{(dp_2,4)\}$	P_5,P_6	$3/0$
3	$\emptyset$	$\emptyset$	$\{(dp_2,4),(dp_1,9)\}$	P_8,P_9	$5/0$

(b) Example of the VN3 Approach

Fig. 5. Running Example of INE, Island, and VN3 Approach

amount of input/ouput (in pages). For the INE and Islands approaches, we also observe the content of the two queues Q_v and Q_{dp}, and the distance from qp to the second nearest data point, denoted as d_2. For the VN3 approach, we track the candidate set, the distance values used, and the final data points found.

It can be observed from Figure 5 that the query performance of the Islands approach is sensitive to the island radius used. When $r_{min} = 8$, the query results are found by checking the islands within which the query point is located. When the radius is decreased to 7, the network expansion takes 2 more steps to finish.

Update of network and data points for the INE approach is obvious—updates only affect one or adjacent pages in the Vertex-Edge and Edge-Data component. For the Islands approach, updates cause the associated islands to be re-computed. As an example, to update data point dp_1, the re-computation will need pages P_1, P_2, and P_3 for network expansion and will then read page P_4 for updating data. Updating network and data point for the VN3 approach, as discussed in [13], requires adjacent Voronoi-Polygons to be re-generated. We use a network expansion process to update the Voronoi polygon of a data point. For example, to update the data point dp_1, a network expansion starting from dp_1 will stop after neighboring data points dp_2 and dp_3 are found. The re-computation process use disk pages P_1, P_2, and P_3. Then pages P_5, P_6, P_7, and P_9 and possibly page P_8 are accessed for updating.

5 Performance Evaluation

Two real-world datasets are used in the evaluation of the INE, VN3, and Islands approaches. The first, AAL, contains the road network of the Aalborg area in the Northern Jutland region of Denmark along with real points of interest. The network contains

$11,300$ vertices, $13,375$ bi-directional edges, and 279 points of interest. The second, LA, represents the road network of Los Angeles, California. This data was obtained via the Internet [26] and converted into network files via the Tiger File Manager [2]. It contains $195,010$ vertices and $266,335$ bi-directional edges. We generate synthetic points of interest for this network.

We measure the performance of the three approaches in terms of CPU time and I/O cost. The CPU time checks, by loading the whole network and pre-computed data into physical memory, the actual running times of the experiments with the three approaches. To measure the I/O cost, we arrange the road network and pre-computation data into the data structures described in Section 4.3, we set the page size to 4k, and we employ an LRU buffer. The buffer size is set to 10% of the sum of the sizes of the Vertex-Edge and Edge-Data components. The AAL dataset contains 129 pages in the two components, and the LA dataset contains $4,132$ pages in the Vertex-Edge component. We disregard the space use that stems form the queues and variables used in the algorithms and thus do not consider them as part of the buffer.

Two series of experiments are conducted. The first series assumes that there are no updates to the road network and studies the effects on query performance of varying k, data point density, and islands radius. The density of data points is the ratio between the number of data points and the number of bi-directional edges in the road network. We define the maximum Euclidean distance between all vertices in the road network as D_{max}. The island radius used is represented as the fraction of D_{max}. In all experiments, islands of the same road network have the same radius.

The second series of experiments considers both query and update performance. We define the update ratio R_u as the ratio of updates being executed per query. The overall performance is the sum of the query and update cost. (To be consistent with the assumed application scenario, we assume an online-processing system where update operations have to be processed together with the query operations so as to provide correct query results). We use updates of edge lengths and updates of the positions of data points on an edge as standard update operations. Given an update ratio R_u and an amount of queries N, there are $N \cdot R_u$ updates on edges as well as data points. The experiments examine the effect on the overall performance of the three approaches of varying update ratio, data point density, and island radius.

In all experiments, the query points are randomly generated. For the first set of experiments, we execute a workload of 200 queries and report the average performance. For the second series of experiments, we increase the number of queries so as to get a proper amount of update operations (the update ratio is assumed to never exceed 0.1). Experiments with the same update ratio are conducted at least three times to obtain average performance figures.

The experiments are performed on a Pentium IV 1.3 GHZ processor with 512 MB of main memory and running Windows 2000. The C++ programming language is used.

5.1 Experiments on Query Performance

Query Performance Versus k We set the island radius to 0.1 of D_{max} and k is varied from 5 to 200. The density of the (real) data points in AAL is 0.02, while the density of the data points (generated) in LA is 0.005. The results are shown in Figure 6. It can be observed that with the growth of k, the computational cost of all three approaches increases. The CPU time of the Islands approach is lower than those of the other two. Both the VN3 and Islands approaches show less I/O than the INE approach. The Islands approach is better than VN3 with respect to I/O cost until k exceeds 50.

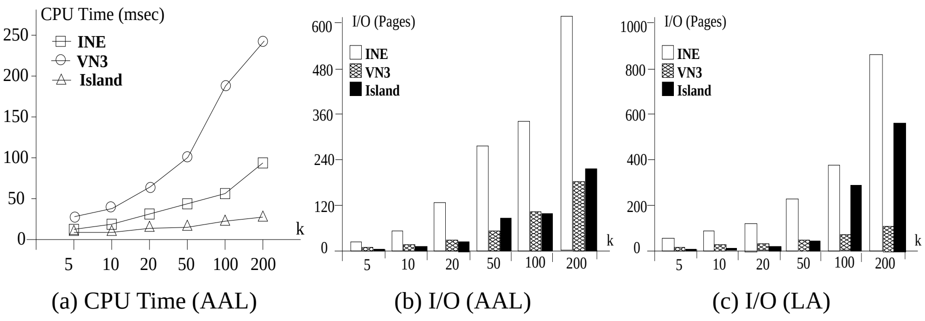


Fig. 6. Query Performance Versus k

Query Performance Versus Density of Data Points We keep the island radius fixed at 0.1 of D_{max} and the value k is set to 10. We now use synthetic data points in both AAL and LA, varying the density from 0.001 to 0.5. It can be seen from Figure 7 that as the density increases, the INE approach improves substantially and becomes competitive. The Islands approach has similar behavior. It has worse performance for the AAL network and data than the VN3 approach (as shown in Figure 7(b)) when the density is less than 0.005, but becomes the best among the three approaches when the density exceeds 0.005. For the LA network and data, the Islands approach always

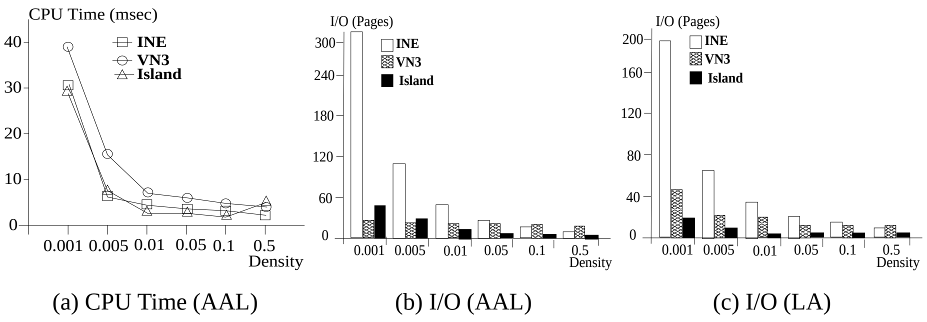
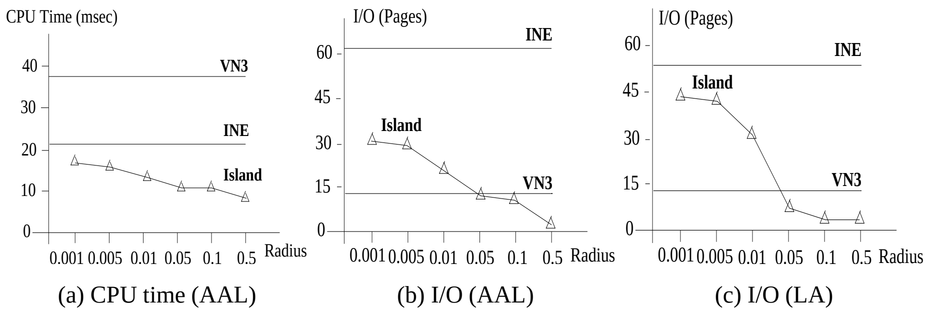


Fig. 7. Query Performance Versus Density

shows better performance than the VN3 approach. The two networks differ in that the connectivity among vertices and the density of edges in the LA network are much higher than in the AAL network. This means that for the same density of data points, the network expansion process finishes earlier in the LA network than in the AAL network. The Islands approach works well in the LA network, as each island is related to more network vertices, which makes it fast for the network expansion process to discover an island. The VN3 approach, in the LA network with its high connectivity and density, possesses more border vertices and pre-computed distance data. It thus requires more I/O in its filter and refinement steps.

Query Performance Versus Island Radius We set $k = 10$ and use the real data points in AAL and synthetic data points in LA ($\text{density} = 0.005$). To determine the impact of the island radius on the query performance, the radius is varied from 0.001 to 0.5 of D_{max}. Note that in Figure 8, we also draw horizontal lines for the INE and VN3 approaches. The Islands approach always has the best CPU performance. As for I/O, the VN3 approach is best when the radius is quite small. When the radius grows to 0.05 of D_{max}, the Islands approach is preferable.



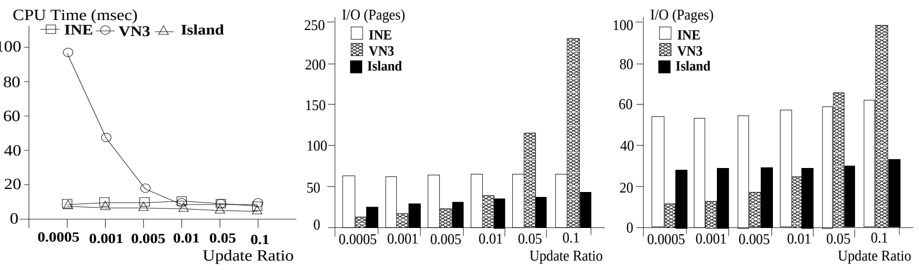
(a) CPU time (AAL) (b) I/O (AAL) (c) I/O (LA)

Fig. 8. Query Performance Versus Island Radius

5.2 Experiments on Overall Performance

We proceed to consider the overall costs for different densities and update ratios of the INE, the VN3, and the Islands approach with two island sizes. The value of k is set to 10 in these experiments (this value is not related to the update operation). The performance costs reported are the sums of the cost of all queries and the cost of all updates of edges as well as data points, divide by the numbers of queries.

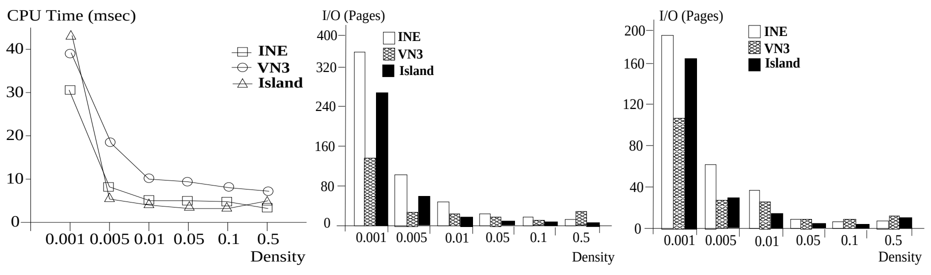
Overall Performance Versus Update Ratio We fix the island radius at 0.01 of D_{max}. We use real data points for the AAL network and synthetic data points with density 0.005 for the LA network. The update ratio is varied from 0.0005 to 0.1 per query. It can be seen from Figure 9 that the INE approach has stable overall performance for the different update ratios, since the update operation only needs to read one or two disk pages. The VN3 approach is better than the other two approaches when the update ratio is less than 0.01. The Islands approach with a radius of 0.01 exhibits almost the same trend as the INE approach.



(a) CPU Time (AAL) (b) I/O (AAL) (c) I/O (LA)

Fig. 9. Overall Performance Versus Update Ratio

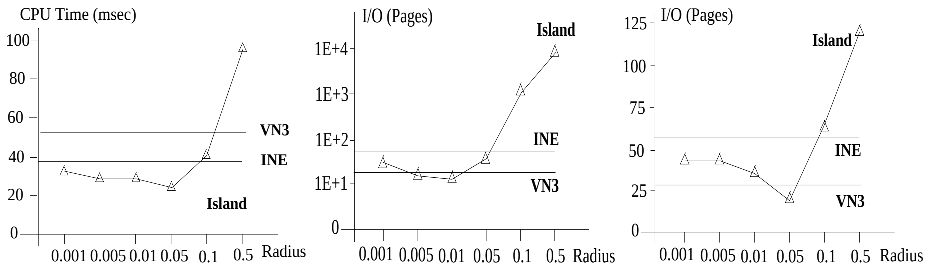
Overall Performance Versus Density of Data Points The island radius remains at 0.01 of D_{max}, and the update ratio is set to 0.01. We use synthetic data points with both networks, varying the density from 0.001 to 0.5, to determine the effect on the overall performances. Figure 10 illustrates that as the density increases, the overall performances of the three approaches improve. At a low density, i.e., 0.001, the VN3 approach has the best performance. When the density grows to 0.01 and beyond, the Islands approach becomes dominant. The INE approach becomes superior when the density reaches 0.5.



(a) CPU Time (AAL) (b) I/O (AAL) (c) I/O (LA)

Fig. 10. Overall Performance Versus Density

Overall Performance Versus Island Radius We retain the update ratio of 0.01 and use the real data points in AAL and synthetic data points in LA (with $\text{density} = 0.005$). The island radius is varied from 0.001 to 0.5 of D_{max}. Figure 11 has horizontal lines for the INE and VN3 approaches, for which the radius is not a parameter. It can be



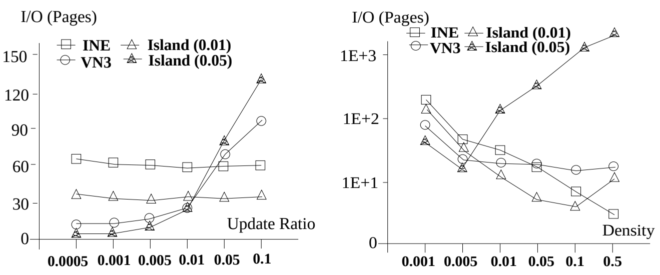
(a) CPU time (AAL) (b) I/O (AAL) (c) I/O (LA)

Fig. 11. Overall Performance Versus Island Radius

observed that the Islands approach has the best CPU performance when the radius is 0.05 or less. As for I/O, experiments on both the AAL and LA datasets show that the overall performance of the Islands approach is better than those of the INE and VN3 for certain radiuses (0.005 and 0.01 for AAL and 0.05 for LA). When the radius exceeds 0.05, the cost of re-computing the islands becomes substantial since islands grow large and overlap significantly.

Island Radius Versus Density and Update Ratio To obtain additional insight into the adaptability of the Islands approach, we conduct experiments on the LA data to check how this approach can be used to cope with different update ratios and densities of data points. We use islands with radiuses that are 0.01 and 0.05 of D_{max}.

Figure 12(a), where the density is 0.005, shows that using islands with radius 0.05 yields the best performance when the update ratio is less than 0.01. When the update ratio exceeds 0.01, the islands with radius 0.01 become the best.



(a) Island Radius Vs. Update Ratio (LA)

(b) Island Radius Vs. Density (LA)

Fig. 12. Islands Versus Update and Density

In the experiment shown in Figure 12(b), the update ratio is fixed at 0.01. We still use islands with radiuses of 0.01 and 0.05. When the density is lower than 0.005, the Islands approach with an island radius of 0.005 achieves the best overall performance. For higher densities, the Islands approach with a radius of 0.01 is a good choice. When the density grows to 0.5, the INE approach shows the best overall performance.

6 Summary and Future Work

This paper presents a versatile approach to k nearest neighbor computation in spatial networks, termed the Islands approach, that generalizes existing re-computation and pre-computation approaches. In particular, pre-computation is performed inside so-called islands, and re-computation is performed in-between islands. An island intuitively is a sub-network with vertices and edges that are no further than a certain distance, termed the radius, away from a data point. Variation of the radiuses of islands enables the approach to accommodate networks with few as well as many data points and few as well as many updates. This enables flexible management of the trade-off between update and query cost.

The paper experimentally compares the Islands approach with two popular kNN algorithms, namely INE and VN3. The experiments result show that the Islands approach is indeed more versatile than these and can be tuned to yield better performance in most cases.

In future work, it would be of interest to try to take into account additional semantics of road networks and transportation infrastructures. For example, real-time road conditions, such as road blocks or traffic jams, may be taken into account. Computing kNN queries in such “dynamic” networks offers new challenges [4, 6]. Next, the Islands approach is capable of using islands with different radiuses within different areas of the network. Techniques for how to dynamically maintain a partitioning of a network into different areas, each with its own, optimal island radius remains an open problem.

References

1. R. Benetis, C. S. Jensen, G. Karciauskas, S. Saltenis. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *Proc. IDEAS*, pp. 44–53, 2002.
2. T. Brinkhoff. The Tiger File Manager. http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/.
3. C. K. Cheng, Y. C. Wei. An Improved Two-Way Partitioning Algorithm with Stable Performance. In *IEEE Trans. CAD*, 10(12), pp. 1502–1511, 1991.
4. Z. Ding, R. H. Güting. Modelling Temporally Variable Transportation Networks. In *Proc. DASFAA*, pp. 154–168, 2004.
5. H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, A. E. Abbadi. Constrained Nearest Neighbor Queries. In *Proc. SSTD*, pp. 257–278, 2001.
6. R. H. Güting, V. T. de Almeida, and Z. Ding. Modeling and Querying Moving Objects in Networks. Fernuniversität Hagen, Informatik-Report 308, April 2004.
7. C. Hage, C. S. Jensen, T. B. Pedersen, L. Speičys, and I. Timko. Integrated Data Management for Mobile Services in the Real World. In *Proc. VLDB*, pp. 1019–1030, 2003.
8. Y. W. Huang, N. Jing, and E. Rundenstener. Effective Graph Clustering for Path Queries in Digital Map Databases. In *Proc. CIKM*, pp. 215–222, 1996.
9. X. Huang, C. S. Jensen, and S. Šaltenis. The Islands Approach to Nearest Neighbor Querying in Spatial Networks. DB Tech Report TR-12. Department of Computer Science, Aalborg University, 2005.
10. G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. In *TODS*, 24(2), pp. 265–318, 1999.
11. G. S. Iwerks, H. Samet, K. Smith. Continuous K-Nearest Neighbor Queries for Continuously Moving Points with Updates. In *Proc. VLDB*, pp. 512–523, 2003.
12. C. S. Jensen, J. Kolář, T. B. Pedersen, I. Timko. Nearest Neighbor Queries in Road Networks. In *Proc. ACMGIS*, pp. 1–8, 2003.
13. M. Kolahdouzan and C. Shahabi. Voronoi-Based Nearest Neighbor Search for Spatial Network Databases. In *Proc. VLDB*, pp. 840–851, 2004.
14. M. Kolahdouzan and C. Shahabi. Continuous K-Nearest Neighbor Search for Spatial Network Databases. In *Proc. STDBM*, pp. 33–40, 2004.
15. F. Korn, N. Sidiropoulos, C. Faloutsos, E. Sieel, Z. Protopapas. Fast Nearest Neighbor Search in Medical Image Databases. In *Proc. VLDB*, pp. 215–226, 1996.
16. A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. Spatial Tessellations, Concepts and Applications of Voronoi Diagrams. John Wiley and Sons Ltd., 2nd edition, 2000.
17. D. Papadias, J. Zhang, N. Mamoulis, Y. Tao. Query Processing in Spatial Network Databases. In *Proc. VLDB*, pp. 802–813, 2003.
18. N. Roussopoulos, S. Kelley, F. Vincent. Nearest Neighbor Queries. In *Proc. SIGMOD*, pp. 71–79, 1995.
19. L. Speičys, C. S. Jensen, A. Kligys. Computational Data Modeling for Network Constrained Moving Objects. In *Proc. ACMGIS*, pp. 118–125, 2003.
20. T. Seidl, H. P. Kriegel. Optimal Multi-Step k-Nearest Neighbor Search. In *Proc. SIGMOD*, pp. 154–165, 1998.
21. C. Shahabi, M. R. Kolahdouzan, M. Sharifzadeh. A Road Network Embedding Technique for K-Nearest Neighbor Search in Moving Object Databases. In *GeoInformatica*, 7(3), pp. 255–273, 2003.
22. S. Shekhar, D. Liu. CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations. In *TKDE*, 19(1), pp. 102-119, 1997.
23. Z. Song, N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *Proc. SSTD*, pp. 79–96, 2001.

24. Y. Tao, D. Papadias, Q. Shen. Continuous Nearest Neighbor Search. In *Proc. VLDB,* pp. 287–298, 2002.
25. M. Vazirgiannis, O. Wolfson. A Spatio Temporal Model and Language for Moving Objects on Road Networks. In *Proc. SSTD,* pp. 20–35, 2001.
26. http://www.census.gov/geo/www/tiger/tgrcd108/tgr108cd.html.
27. X. Xiong, M. F. Mokbel, W. G. Aref. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In *ICDE,* 2005.
28. C. Yu, B. C. Ooi, K. L. Tan, H. V. Jagadish. Indexing the Distance: An Efficient Method to KNN Processing. In *Proc. VLDB,* pp. 421–430, 2001.
29. J. S. Yoo, S. Shekhar. In-Route Nearest Neighbor Queries. In *GeoInformatica,* 9(2), pp. 117–137, 2005.

Appendix

v_1	v_2	P_1	3	Nil	P_4
v_1	v_4	P_1	2	Nil	P_4
v_2	v_1	P_1	3	Nil	P_4
v_2	v_3	P_2	5	P_3	P_4
v_2	v_4	P_1	4	Nil	P_4
v_2	v_6	P_2	7	P_3	P_4
v_4	v_1	P_1	2	Nil	P_4
v_4	v_2	P_1	4	Nil	P_4
v_4	v_5	P_2	4	P_3	P_4

(a) Page: P_1

v_3	v_2	P_1	5	P_3	P_4
v_3	v_7	P_2	7	P_3	P_4
v_5	v_4	P_1	4	P_3	P_4
v_5	v_6	P_2	5	Nil	P_4
v_6	v_5	P_2	5	Nil	P_4
v_6	v_2	P_1	7	P_3	P_4
v_6	v_7	P_2	2	Nil	P_4
v_7	v_3	P_2	6	Nil	P_4
v_7	v_6	P_2	2	Nil	P_4

(b) Page: P_2

dp_1	$e_{4,5}$	1
dp_1	$e_{5,4}$	3
dp_2	$e_{2,6}$	4
dp_2	$e_{6,2}$	3
dp_3	$e_{2,3}$	2
dp_3	$e_{3,2}$	3

(c) Page: P_3

v_1	dp_1	3
v_1	dp_2	7
v_1	dp_3	5
v_2	dp_1	5
v_2	dp_2	4
v_2	dp_3	2
v_3	dp_3	3
v_4	dp_1	1
v_4	dp_2	8
v_4	dp_3	6
v_5	dp_1	3
v_5	dp_2	8
v_6	dp_1	3
v_6	dp_2	3
v_7	dp_2	5

(d) Page: P_4

dp_1	v_1	3	P_6
dp_1	v_4	1	P_6
dp_1	v_5	3	P_6
dp_2	v_6	3	P_6
dp_2	v_7	5	P_6
dp_3	v_2	2	P_6
dp_3	v_3	3	P_6

(e) Page: P_5

dp_1	dp_2	b_5	5.5	P_9
dp_1	dp_3	b_1	4	P_7
dp_1	dp_3	b_2	3.5	P_7
dp_2	dp_1	b_5	5.5	P_9
dp_2	dp_3	b_3	3	P_8
dp_2	dp_3	b_4	7	P_8
dp_3	dp_1	b_1	4	P_7
dp_3	dp_1	b_2	3.5	P_7
dp_3	dp_2	b_3	3	P_8
dp_3	dp_2	b_4	7	P_8

(f) Page: P_6

b_1	b_2	3.5
b_1	b_3	3
b_1	b_4	11
b_1	b_5	9.5
b_1	v_1	1
b_1	v_2	2
b_1	v_3	7
b_1	v_4	3
b_1	v_5	7
b_2	b_1	3.5
b_2	b_3	2.5
b_2	b_4	10.5
b_2	b_5	9
b_2	v_1	4.5
b_2	v_2	1.5
b_2	v_3	6.5
b_2	v_4	2.5
b_2	v_5	6.5

(g) Page: P_7

b_3	b_1	3
b_3	b_2	2.5
b_3	b_4	10
b_3	b_5	8.5
b_3	v_2	1
b_3	v_3	6
b_3	v_6	6
b_3	v_7	7
b_4	b_1	11
b_4	b_2	10.5
b_4	b_3	10
b_4	b_5	6.5
b_4	v_2	9
b_4	v_3	4
b_4	v_6	4
b_4	v_7	2

(h) Page: P_8

b_5	b_1	9.5
b_5	b_2	9
b_5	b_3	8.5
b_5	b_4	6.5
b_5	v_1	8.5
b_5	v_4	6.5
b_5	v_5	2.5
b_5	v_6	2.5
b_5	v_7	4.5

(i) Page: P_9

Fig. 13. Sample Data Pages