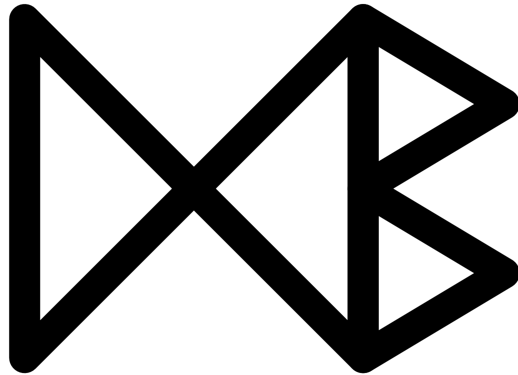


The Islands Approach to Nearest Neighbor Querying in Spatial Networks

Xuegang Huang, Christian S. Jensen, Simonas Šaltenis

October 19, 2006

TR-16

A DB Technical Report

Title	The Islands Approach to Nearest Neighbor Querying in Spatial Networks
	Copyright © 2006 Xuegang Huang, Christian S. Jensen, Simonas Šaltenis. All rights reserved.
Author(s)	Xuegang Huang, Christian S. Jensen, Simonas Šaltenis
Publication History	Extended version of: Xuegang Huang, Christian S. Jensen, Simonas Šaltenis, "The Islands Approach to Nearest Neighbor Querying in Spatial Networks." In *Proceeding of 9th International Symposium on Spatial and Temporal Databases*, Angra dos Reis, Brazil, August 22-24, 2005, pp. 73–90.

For additional information, see the DB TECH REPORTS homepage: ⟨www.cs.aau.dk/DBTR⟩.

Any software made available via DB TECH REPORTS *is provided "as is" and without any express or implied warranties, including, without limitation, the implied warranty of merchantability and fitness for a particular purpose.*

The DB TECH REPORTS icon is made from two letters in an early version of the Rune alphabet, which was used by the Vikings, among others. Runes have angular shapes and lack horizontal lines because the primary storage medium was wood, although they may also be found on jewelry, tools, and weapons. Runes were perceived as having magic, hidden powers. The first letter in the logo is "Dagaz," the rune for day or daylight and the phonetic equivalent of "d." Its meanings include happiness, activity, and satisfaction. The second letter is "Berkano," which is associated with the birch tree. Its divinatory meanings include health, new beginnings, growth, plenty, and clearance. It is associated with Idun, goddess of Spring, and with fertility. It is the phonetic equivalent of "b."

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. In this paper, 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, and it offers better overall query and update performance than do its predecessors. 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 [13, 14, 15, 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. This scenario contrasts the one where no network is assumed, movement is unconstrained, and Euclidean distance is used.

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.

In this paper, we consider the performance of query as well as update processing, as efficient query and update processing 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 a 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 disk access 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 comparison verifies the general applicability of the Islands approach, which has the best overall performance.

This paper makes three main contributions.

- The Islands approach offers an attractive generalization of the existing kNN query processing techniques for spatial networks. First, it employs a relatively simple data structure and an intuitive search algorithm. Second, 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, thus enabling substantial flexibility in managing the trade-off between query and update performance. By tuning the sizes of islands, the amount of pre-computation can be controlled. 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 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 for data constrained to spatial networks. 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 [11, 18]. Many extensions and applications of kNN computation have also been proposed [1, 6, 12, 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 [5, 7, 8, 19, 25], we adopt a fairly standard graph-based data model and structure [8, 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 [9, 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 [4] 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. [9], 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.

As described, existing techniques for kNN computation in spatial networks can be characterized as being either online processing techniques or pre-computation techniques. To provide a thorough discussion of the existing techniques 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 extensions to the algorithm that enable it to utilize islands of different radiuses and to accommodate additional semantics of transportation networks.

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.

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 $qp = (e_{7,6}, 1)$.

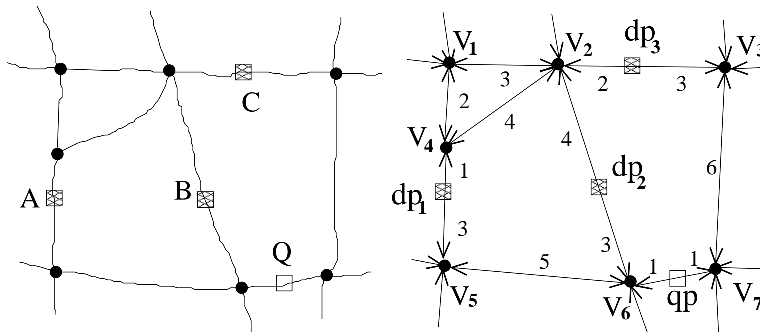


Figure 1: Road Network Model

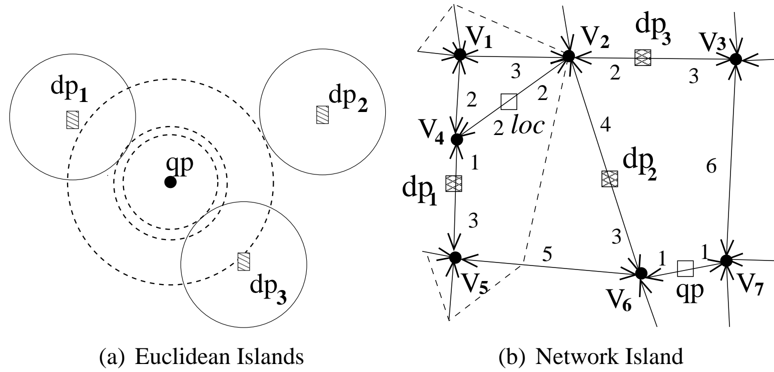
The example road network in Figure 1 is assumed to have only bi-directional roads with no u-turn restrictions and each data point has two positions—one on each side of a bi-directional road. The remainder of the description of the Islands approach is carried out under these assumptions. Extension of the Islands approach to capture various pragmatic road restrictions is covered in Section 3.4.

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

Figure 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 with 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 (2)–(4) assuming $qp = (a, a.l - qp.pos)$
(6) Let dp_k denote the k-th element in Q_{dp}, $dp_k = \varnothing$, 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
(13) Q_{dp}.*update*(*dp*, $d(qp, dp)$)
(14) $d_k \leftarrow d(qp, dp_k)$
(15) $v \leftarrow Q_v$.*deque*, mark v visited
(16) **return** Q_{dp}

Note that in line 13 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

Operations on the Radius In real-world applications kNN computation is based on a disk-based road network data structure. In addition to the query performance, the update efficiency of the road network data should be considered. There are two types of updates of the road network data: updates of the network itself, i.e., of vertices and edges, and updates of data points.

Updating a vertex or an edge may possibly require all islands covering this vertex or edge to be re-computed. An initial check can be made to compute the network distance from the updated vertex or edge to the centers of the islands covering it. If the distance remains the same, so do the islands. Updating a data point causes its island to be re-computed. Such re-computation does not influence any other islands. Since an update operation usually involves a network expansion, the island's radius, which determines how far the expansion process goes, can be tuned to control the update efficiency. For a large road network with relatively few data points, the radiuses of islands can be increased so that the kNN search terminates quickly. Conversely, in a small road network with many data points, the radiuses of islands can be decreased so that the costs of update operations are reduced.

We proceed to define two operations on the radius of an island. The *shrink operation* on an island reduces the radius of this island, which improves the local update performance. Note that if the radius, after being shrunk, is still no less than the smallest island radius r_{min} of the road network, the shrink operation has only minor impact on the query performance. Only if all islands are shrunk and r_{min} is decreased, the query performance will be reduced.

The *expand operation* on an island increases the radius of this island, which reduces the local update performance. Again, running the expand operation on individual islands will not necessarily increase the query performance. Only when the expand operation is run on all islands, the smallest radius r_{min} will increase, and the overall query performance will be improved. Next, we will describe how the expand operation can be used to increase the radiuses of islands in local areas of a spatial network, where queries will then terminate earlier, by using a larger, local r_{min} value.

Summarizing, the shrink and expand operations can be applied to all islands to change r_{min} so as to balance 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.

Cross-Area Island Expansion As has been described, islands in different parts of a road network can have different radiuses. Intuitively, radius of islands in urban areas should be relatively small since the densities of points of interest are relatively high and there are relatively frequent updates of road network data. In rural area, islands can be given much bigger radiuses since the densities of points of interest are much smaller and there are far fewer updates.

When a kNN query is issued close to the border of two areas that have different r_{min} values, algorithm $IslandExpansion$ introduced in Section 3.3 can be improved to take into account the different r_{min} values.

We proceed to discuss how to modify the $IslandExpansion$ algorithm to better handle such cross-area expansions.

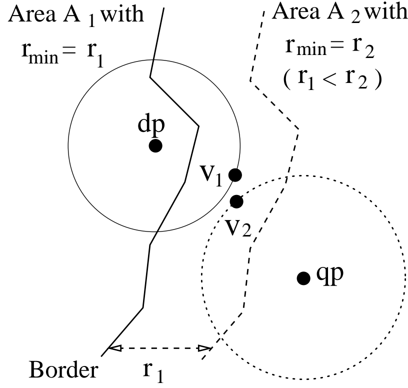


Figure 3: Cross-Area Island

We assume that islands of data points located near the border of two areas can be pre-computed across borders. As shown in Figure 3, data point dp is located near the border between areas A_1 and A_2. Note that the smallest island radius r_{min} in A_1 is r_1 and in A_2 is $r_2, (r_1 < r_2)$. Since dp is in A_1, without loss of generality, suppose the island for dp has a radius of r_1. A network expansion process starts from a query point qp in A_2 to find kNNs and stops at vertex v_2, which is quite close to the border between A_1 and A_2.

Based on the pseudo code of the $IslandExpansion$ algorithm, $d(qp, v_2) + r_2 \geq d_k$. Note that vertex v_1 belongs to the island of dp and $d(qp, v_1) > d(qp, v_2)$. But since $r_1 < r_2$, it is possible that $d(qp, v_1) + r_1 < d(qp, v_2) + r_2$. Thus, the expansion process starting from qp will be insufficient if it only considers r_2.

To handle this, we define an *isolation region* for road network A_2 (as shown in Figure 3). All vertices in A_2 whose network distance to the border of A_1 and A_2 is less than or equal to r_1 belong to this region. The $IslandExpansion$ algorithm running inside A_2 is correct if the expansion process does not enter this region. Otherwise, the radius r_1 has to be used instead of r_2 in the $IslandExpansion$ algorithm to ensure correctness. Note that for road network A_1, there is no need to define such a region since $r_1 < r_2$ and the expansion process starting at A_1 always uses r_1. In general, when two road-network areas meet, the isolation region must be pre-computed for the area with the larger r_{min}.

How the road-network is divided (possibly automatically) into areas with different r_{min} values is an interesting future research direction.

Pragmatic Road Restrictions As mentioned, the above discussion assumes that all edges in the road network are bi-directional and without U-turn restrictions. We proceed to extend the Islands approach to accommodate pragmatic road restrictions. We make changes to both the pre-computation component and the network expansion algorithm.

First, since edges in the real-world road networks are not always bi-directional, the network distance from one location to another may not always be the same as the way back. Thus, in the pre-computation component, each data point has two associated islands, i.e., the "incoming island" that is built by making network expansion from the data point using "incoming" edges, and the "outgoing island" that is constructed using "outgoing" edges. Then the $IslandExpansion$ algorithm finds k nearest neighbor data points to (from) a query point by using "incoming" ("outgoing") edges of each vertex until enough "outgoing" ("incoming") islands are found.

Second, as described in the data model, a data point can be associated with more than one location. Two ways exist for processing such a constraint. The first way is to treat each location as a different data point. Then each location has its island. The network expansion process will eliminate islands denoting different locations of the same data point. The second way is to define the data point as a new vertex in the road network. Then, edges are defined to link the vertex with the locations associated with the data point. The lengths of these edges are set to zero and no changes are made to the $IslandExpansion$ algorithm.

Third, since U-turn restrictions are quite common in the real world, we propose two ways to process U-turn restrictions. Note that we only need to consider edges with U-turn allowed that have at least one data point. For such data points, we can either add one more location of the data point at the corresponding co-edge (as assumed in Figure 2(b)), or we can always check the co-edge during the network expansion process.

Fourth, for two roads that meet at a road intersection, there may exist turn restrictions that prohibit the direct movement from one road to the other through the intersection. As shown by, e.g., Speičys et al. [19], such turn restrictions can generally be handled by adding new vertices and zero-length edges to the original network, thus obtaining a new network. By using this new network, the Islands approach needs no modifications to handle turn restrictions.

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 disk access.

Papadias et al. [17] introduce two algorithms for kNN computation in spatial network, namely Incremental Euclidean Restriction (IER) and Incremental Network Expansion (INE). Based on the observation that the Euclidean distance between any two locations never exceeds their network distance, the IER algorithm obtains k Euclidean nearest neighbors and arranges them in ascending order of their network distance to the query point. Subsequent Euclidean neighbors are retrieved incrementally until the next Euclidean neighbor has larger Euclidean distance than the network distance from the query point to the kth neighbor.

The second algorithm, INE, performs incremental network expansion 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.

It has been shown [17] that the INE algorithm outperforms the IER algorithm in every aspect. However, there are still cases where the INE approach seems to be relatively inefficient. Specifically, its performance 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.

The disk-based road network data structure used by the IER and INE algorithms has been adapted into our testbed data structure. We use this structure because it preserves connectivity and locality of the road network and because it is robust with respect to updates of the road network as well as data points.

4.2 kNN Pre-Computation Approach

To save the cost of network expansions in online computations, pre-computation techniques can be designed to pre-calculate a certain amount of network distances between data points and vertices. 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. The major drawback of this method is that it involves an off-line pre-computation of the network distances between all pairs of vertices and uses high-dimensional spatial indexes which

limits its applicability and leads to poor performance. Kolahdouzan and Shahabi [14] propose 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 4, 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.

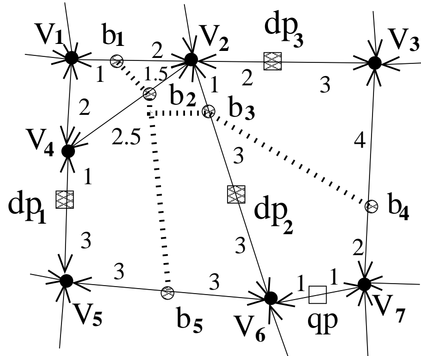


Figure 4: Network Voronoi Diagram

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.

We are also aware of two very recent publications on the same topic. First, Cho et al. [3] propose the UNICONS (UNIque Continuous Search) approach [3], for efficient k nearest neighbor and continuous k nearest neighbor queries in road networks. The UNICONS approach pre-computes and stores k nearest neighbors to a selected amount of *condensing* points in the network. The condensing point is defined as the network vertex that has at least three adjacent vertices. However, the approach does not provide a systematic way of choosing the *condensing* points. In addition, such pre-computation is fragile to face updates to the networks (i.e., a simple update to a vertex or edge will cause all the pre-computation data to be re-generated). Second, Hu et al. [10] propose a novel approach, termed SPIE, which considers to make reduction on the networks so that the number of edges can be decreased. To achieve this, a set of inter-connected *shortest path trees* (SPT) are generated based on the network. Starting from a vertex (a root node) in a road network, a Dijkstras algorithm is called to grow the tree. During the tree-growth process, a new tree node n is added if its distances to other existing tree nodes are preserved by the tree. This is determined by checking if there is any adjacent edge of n that connects it to a tree node closer than the tree path. If such edge, denoted as *shortcut*, exists, this node becomes a new root and a new SPT starts to grow from it. The SPTs is then transformed into the SPIE structure which provides further improvement to the k nearest neighbor query performance. The SPIE approach has severe shortcomings which makes it impractical. First, this approach assumes that the data points are distributed on network vertices which does not fit in many situations that data points are represented with linear-referencing, i.e., they locate on edges in-between two network vertices. And it will make the network more complex by adding each data point as

vertices to the network. Second, the modern transportation networks often involve one way streets, u-turn restrictions, and turn-restriction on road junctions. These made the road network not as simple as a undirected graph. The SPIE approach will not benefit from making reductions on the networks by considering these actual situations.

Due to the focus of our discussion, we choose to focus on comparing our approach with the VN3 and INE approaches. We proceed to introduce a disk-based data structure for the INE, the VN3, and the Islands approach.

4.3 Disk-Based Data Structure

The disk-based data structure for road network data used together with the INE, the VN3, and the Islands approach is shown in Figure 5. This structure is an adaptation of the data structures proposed for the INE [17] and VN3 [14] approaches.

The structure consists of six components, named as follows: *Vertex–Edge*, *Edge–Data*, *Island–Precomputation*, *Voronoi–Polygon*, *Border–Adjacency*, and *Voronoi–Precomputation* component.

The road network and data points are represented by the Vertex-Edge and Edge-Data components. As illustrated in Figure 5, based on the example road network in Figure 1, 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 has the form $(vsID, veID, ptNBVE, L, ptDP, ptI)$, where $vsID$ 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.

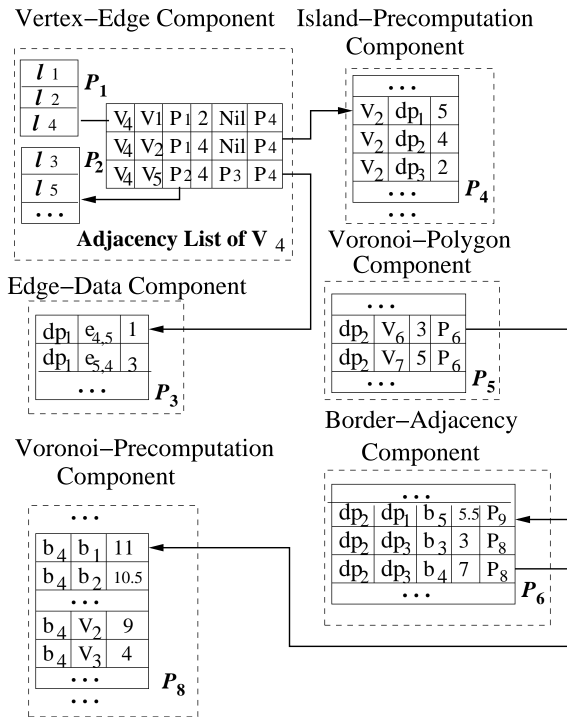


Figure 5: Disk-Based Data Structure

Each entry in the Edge-Data component has the form $(dpID, eID, offset)$, 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 5, 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 component, the Border-Adjacency component, and the Voronoi-Precomputation component. 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 Appendix 6. 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 disk access for the three approaches. The *D.A.* column denotes the amount of disk reads (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 6 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.

4.5 Update Operations

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. Updating network and data point for the VN3 approach, as discussed in [14], requires adjacent Voronoi-Polygons to be re-generated. We can 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 uses 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.

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. We proceed to describe in detail how to handle updates to the network data and the data points. These include the *insertion*, *deletion* of network vertices, edges, data points and changes to edge length as well as positions of data points on edges. We assume an in-memory list *Island-Table* is available. Each entry in the list has the form $(dpID, r)$, where $dpID$ denotes a data point and r is the island size of this data point. As discussed in previous sections, the island size r can be changed in various

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

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

Steps	Candidates	Distances	Results	Buffer	D.A.
1	$\emptyset$	$\emptyset$	$\{(dp_2,4)\}$	P_2,P_5	2
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
3	$\emptyset$	$\emptyset$	$\{(dp_2,4),(dp_1,9)\}$	P_8,P_9	5

(b) Example of the VN3 Approach

Figure 6: Running Example of INE, Island, and VN3 Approach

area of the network to balance between update and query performance. We maintain an in-memory list for the update operations. When an island needs to be re-generated, the value r of each island in *Island-Table* provides the island size for the re-generation.

Update of vertex: When a new vertex is inserted to the network, we first need to update the *Vertex-Edge* component. Next, for each island covering adjacent vertices of this new vertex, we check if this island also covers the new vertex (based on the *Island-Table*) and update the *Island-Precomputation* component. Similarly, when a vertex is deleted from the network, we read all the islands covering this vertex, re-generate these islands and update the *Island-Precomputation* component as well as the *Island-Table*. Modification on a vertex will not change the data structure if it does not change the network topology related to this vertex. Otherwise, the modification to the vertex is a combination of deletion and insertion steps on this vertex.

Update of edge: When a new edge is inserted or deleted, assuming that the two vertices of the edge are already in the network, we read all the islands covering the any of the two vertices and re-generate these islands. Next, when the length of an edge is increased, we need to re-generate those islands that cover both vertices. If the length is decreased, all the islands covering any of the two vertices of the edge have to be re-generated.

Update of data point: When a data point is inserted, we need to make a network expansion from this data point and add the data point to all the network vertices inside this island. The *Island-Table* is also updated. Similarly, when a data point is deleted, we also make a network expansion to extract this data point from each vertex inside the island of this data point. The *deletion* and *insertion* processes can be used

as a combination of operations when the data point's position on an edge is changed.

4.6 Further Improvement to Islands

In the network expansion process of the *IslandExpansion* algorithm, when a new network vertex is dequeued from Q_v, all the islands (i.e., the data points) covering this vertex have to be read from the *Island-Precomputation* component to update the queue Q_{dp}. This operation is not efficient as a "discovered" island may still be accessed in the network expansion process. To reduce the redundancy, we propose to partition the *Island-Precomputation* component into two parts, i.e., the *Island-Precomputation*^{i} component and the *Island-Precomputation*^{b} component. Both components have the same format as the *Island-Precomputation* component. The first component, *Island-Precomputation*^{i}, records a vertex, a data point and their distance if the vertex is an *internal vertex* of the data point's island. The second component, *Island-Precomputation*^{b}, saves a vertex, a data point and their distance if the vertex is the *border vertex* of the island of the data point. In accordance to this modification, each entry in the *Vertex-Edge* component needs to have two pointers to disk pages. One pointer links the entry to the islands where the start vertex of the entry is the *internal vertex* and the other links the entry to the islands where the start vertex is the *border vertex*.

Data generation for the two components is obvious. Specifically, in the network expansion process of generating each island, we only need to record the vertices that are "dequeued" from the queue Q_v and check each vertex with the definitions of *internal vertex* and *border vertex* to decide the component for storing this island.

To modify the *IslandExpansion* algorithm with the two components, in step 4 of the algorithm, it is necessary to access islands from both the *Island-Precomputation*^{i} and the *Island-Precomputation*^{b} components that cover vertices $qp.e.v_s$ and $qp.e.v_e$. Next, at steps 12 and 13 inside the while-loop (i.e., the expansion process), the algorithm only needs to read islands from the *Island-Precomputation*^{b} component until k nearest neighbors are found.

Intuitively, by having these two components, the efficiency of the k nearest neighbor query is further improved as the expansion process gets fewer accesses to the islands data. The update operation on this "improved" islands is the same as described in the previous section but can be slightly more complex as the two components need to be updated at the same time. To evaluate on how this further improvement can influence the efficiency of Islands approach, we denote this as *FI_Island* (Further Improved Islands) approach and compare it with the aforementioned Islands approach in the next section.

5 Performance Evaluation

Two real-world datasets are used in the evaluation of the discussed approaches. The first dataset, 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 dataset, LA, represents the spatial network data of Los Angeles, California. This data was obtained via the Internet [26] and converted into network files via the Tiger File Manager [2]. The LA dataset contains $195,010$ vertices and $266,335$ bi-directional edges. We generate synthetic points of interest for the LA network.

We measure the performance of the these approaches in terms of CPU time and cost of disk access. 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 disk access 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 from the queues and variables used in the algorithms and thus do not consider them as part of the buffer.

Three series of experiments are conducted. The first series assumes that there are no update 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 (In the case that the radius r_{min} is too small for certain islands, i.e., a data point is far away from at lease one vertex of its edge, we increase the size of this particular island so that this island covers at least two vertices of the data point's edge).

The second series of experiments considers both query and update performance on the INE, VN3 and Islands approaches. 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.

The third series of experiments checks the pre-computation cost of the Islands approach. This includes the CPU time and disk access cost on pre-computation of each island and the space requirements on storing the pre-computed distance data.

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.

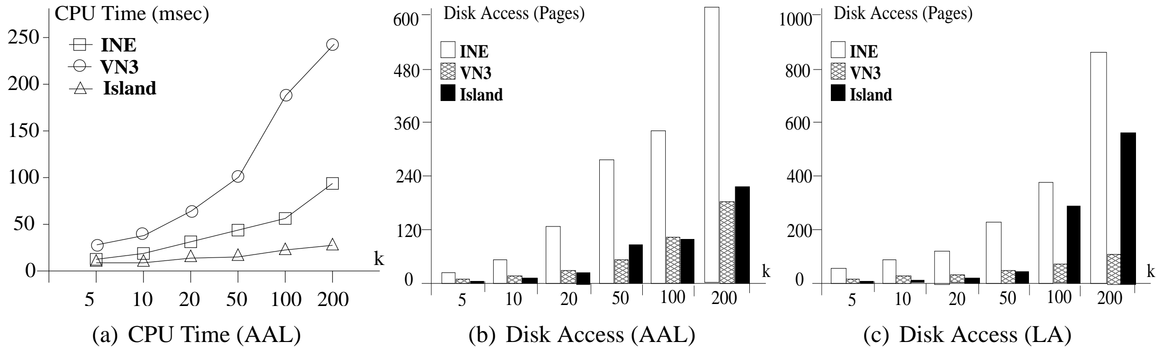
The experiments are performed on a Pentium IV 1.3 GHZ processor with 512 MB of main memory and running Windows 2000. The C++ programming language is used.

5.1 Experiments on Query Performance

In the first series of experiments, we present both CPU time and disk access costs of the experiments on AAL data and focus on checking the disk access in the experiments with the LA data.

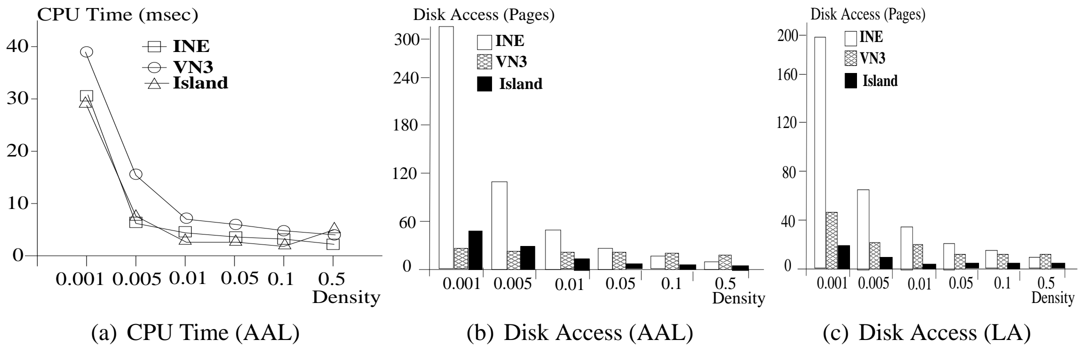
Query Performance Versus k In this experiment, the island radius is set to 0.1 of D_{max} and k is varied from 5 to 200. We use the real world data points for the AAL road network and synthetic data points for LA road network. The density of data points in AAL is 0.02 while the density for LA is 0.005. The results are shown in Figure 7. 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 better than those of the other two. Both the VN3 and Islands approaches show less disk access than the INE approach. The Islands approach is better than VN3 with respect to disk access cost until k grows beyond 50.

Query Performance Versus Density of Data Points In this experiment, the island radius is 0.1 of D_{max}. The value k is set to 10. We remove the real data points in AAL and use synthetic data points in both AAL and LA road network. The density is varied from 0.001 to 0.5 to check the performance of the three approaches. It can be seen from Figure 8 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 8(b)) when the density is less than 0.005, but becomes the best among the three approaches when the density exceeds 0.005.



(a) CPU Time (AAL)   (b) Disk Access (AAL)   (c) Disk Access (LA)

Figure 7: Query Performance Versus k



(a) CPU Time (AAL)   (b) Disk Access (AAL)   (c) Disk Access (LA)

Figure 8: Query Performance Versus Density

For the LA network and data, the Islands approach always 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 for finding kNNs in the LA network finishes earlier than in the AAL networks. The Islands approach is more favorable 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 disk access in its filter and refinement steps.

Query Performance Versus Island Radius In this experiment, 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 island radius on the query performance, the radius is varied from 0.001 to 0.5 of D_{max}. Note that in Figure 9, we also draw horizontal lines for the INE and VN3 approaches. It can be observed that the Islands approach always has a better CPU performance. As for disk access, when the radius is quite small, the VN3 approach has less disk access. When the radius grows to 0.05 of D_{max}, the Islands approach begins to show the best query performance among the approaches.

Query Performance On Further-Improved Island To evaluate the improvement to Islands approach (described in Section 4.6), we compare the query performance of *FI_Island* with the Islands approach on the effect of k, Island radius, and the density of data points. As illustrated in Figure 10, when k grows bigger and the data point density decreases, the *IslandExpansion* algorithm needs to access more Islands to discover enough nearest neighbors. Compared to the *Island-Precomputation* component of the Islands approach, the *Island-Precomputation^b* component of *FI_Islands* only stores necessary data for the expansions process (i.e., only the border vertices of islands remember the islands), the *FI_Island* approach exhibits a

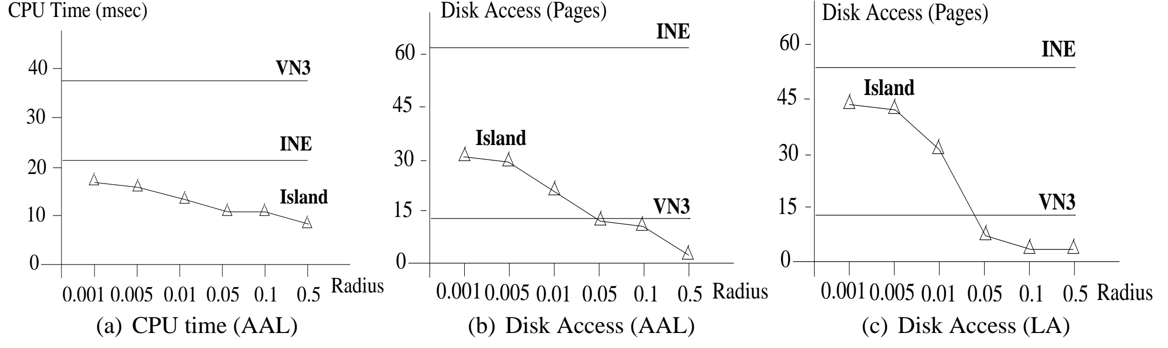


Figure 9: Query Performance Versus Island Radius

slightly better query performance than the Islands. In other cases, performance of both approaches is very close.

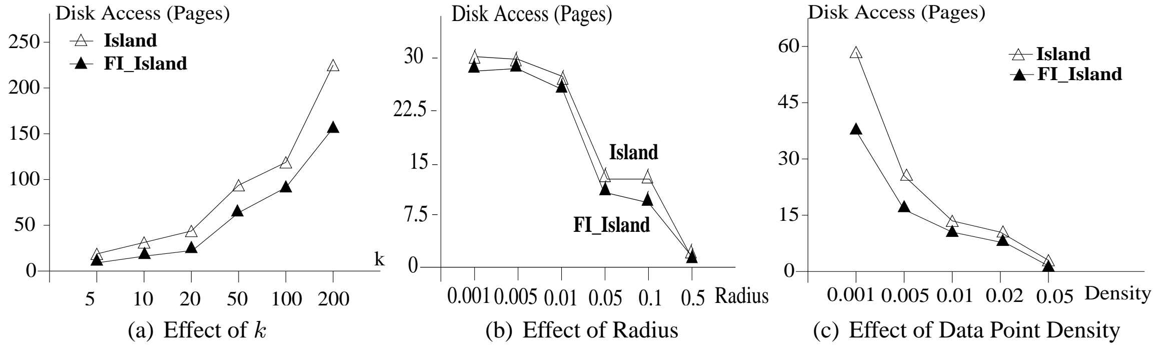


Figure 10: Query Performance of Further-Improved Island (AAL)

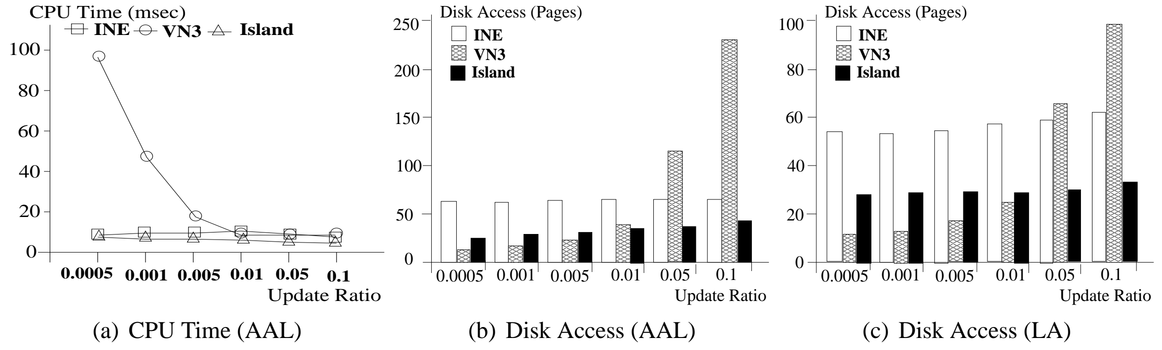
5.2 Experiments on Overall Performance

In the second series of experiments, we present the experimental result on comparing the overall costs for different densities and update ratios of the INE, the VN3, and the Islands approach with two islands sizes. The value of k is set to 10 in these experiments—this value is not related to the update operation. We report the overall performance costs by adding the cost of all queries to the cost of all updates of edges as well as data points and dividing the totals by the numbers of queries.

Overall Query and Update Performance Versus Update Ratio In this experiment, the island radius is fixed at 0.01 of D_{max}. We use real data points for the AAL road network and synthetic data points at a density of 0.005 for the LA road network. The update ratio is varied from 0.0005 to 0.1 per query. It can be seen from Figure 11 that the INE approach has a stable overall performance for 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 smaller than 0.01. The Islands approach, with a radius of 0.01 exhibits almost the same trend as the INE approach.

Overall Performance Versus Density of Data Points In this experiment, the island radius remains at 0.01 of D_{max}. The update ratio is set to 0.01. We remove the real data points in AAL and use synthetic data points in both networks.

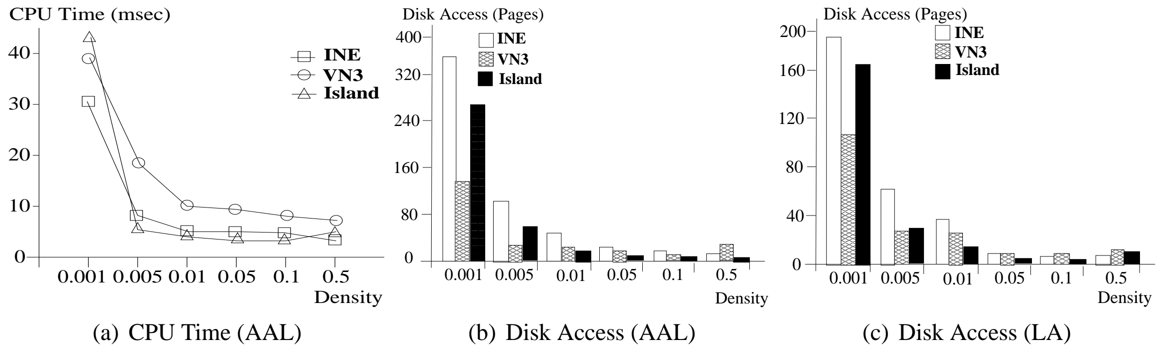
The density is varied from 0.001 to 0.5, to determine the overall performances of the three approaches



(a) CPU Time (AAL) (b) Disk Access (AAL) (c) Disk Access (LA)

Figure 11: Overall Performance Versus Update Ratio

for varying densities. As can be seen from Figure 12, as the density increases, the overall performances the three approaches improve. At a lower density, i.e., 0.001, the VN3 approach has 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) Disk Access (AAL) (c) Disk Access (LA)

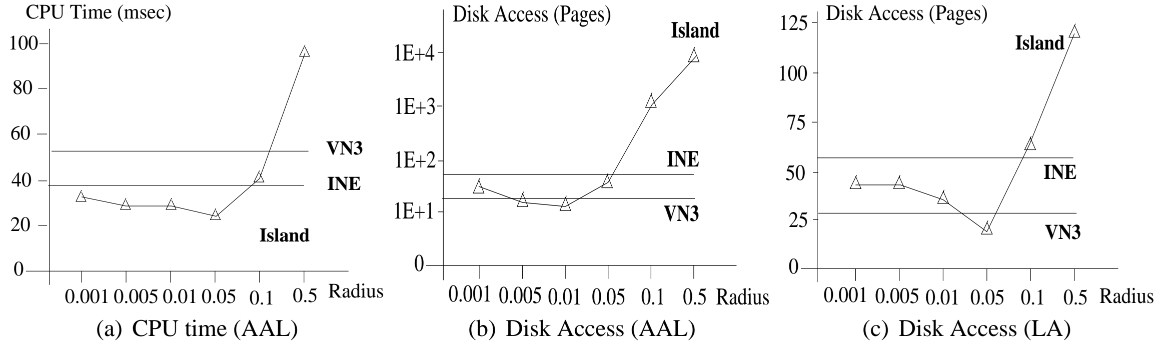
Figure 12: Overall Performance Versus Density

Overall Performance Versus Island Radius In this experiment, we set the update ratio to 0.01 and use the real data points in AAL and synthetic data points in LA (with $\text{density} = 0.005$). To check the impact of the island radius on the query performance, the radius is varied from 0.001 to 0.5 of D_{max}.

In Figure 13, we also draw two horizontal lines for the INE and VN3 approaches, for which the radius is not a parameter. It can be observed that the Islands approach has the best CPU performance when the radius is 0.05 or smaller. As for disk access, 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 two island sizes, setting the radius to 0.01 and 0.05 of D_{max} in the experiments.

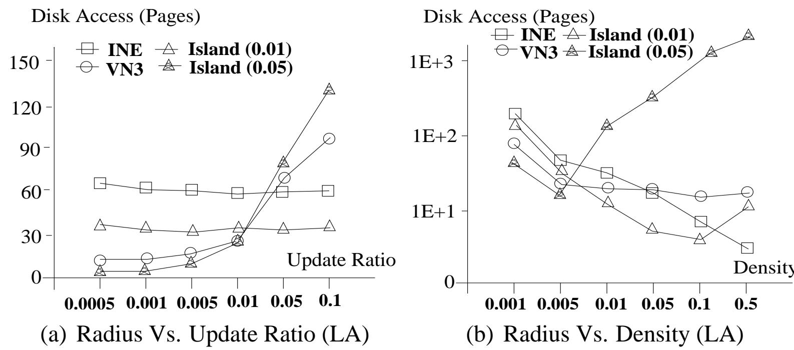
In Figure 14(a), we set the density to 0.005. It can be observed that the islands with radius 0.05 has the best performance when the update ratio is smaller than 0.01. When the update ratio grows to higher than 0.01, the islands with radius 0.01 become the best. In the experiment shown in Figure 14(b), the



(a) CPU time (AAL) (b) Disk Access (AAL) (c) Disk Access (LA)

Figure 13: Overall Performance Versus Island Radius

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. Only when the density grows to 0.5, the INE approach shows the best overall performance.

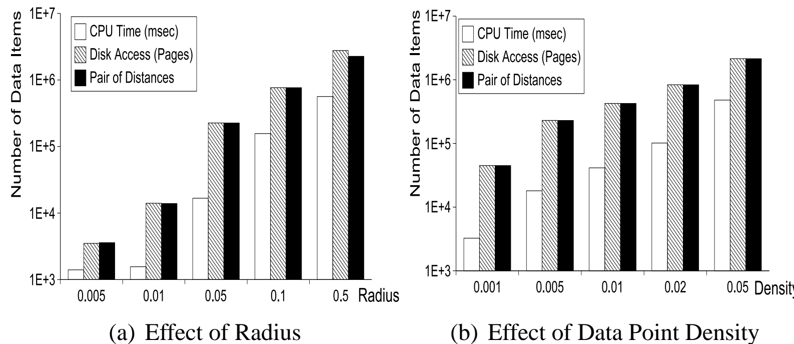


(a) Radius Vs. Update Ratio (LA) (b) Radius Vs. Density (LA)

Figure 14: Islands Versus Update and Density

5.3 Experiments on Pre-computation Cost

To evaluate on the pre-computation cost of the Islands approach, we list the CPU time, disk access, and the amount of pre-computed distance pairs for the Islands approach on AAL network. Note that we do not include the CPU time and disk access cost of assigning each pre-computed pairs into disk pages as the cost of this process varies among different hardware settings. As illustrated in Figure 15, with the growth of Island radius and the data point density, the time complexity and the space requirement (for storing the pre-computed data) increases. We omit the comparing of Islands and VN3 approach on the pre-computation cost. The comparison of Islands and *FI_Island* is also saved as the latter has almost the same pre-computation cost as the Islands approach (as described in Section 4.6).



(a) Effect of Radius (b) Effect of Data Point Density

Figure 15: Islands Pre-computation Cost (AAL)

6 Summary and Future Work

This paper presents a versatile approach to k nearest neighbor computation in spatial networks, termed the Islands approach. This approach 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. As a result, the Islands approach is thus attractive for use in supporting location-based mobile services.

Several possible directions for future work exist.

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 [5, 7]. 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 *VLDB J.,* 15(3), pp. 229–249, 2006.

[2] T. Brinkhoff. The Tiger File Manager. http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/.

[3] H. -J. Cho, C. -W. Chung. An Efficient and Scalable Approach to CNN Queries in a Road Network. In *Proc. VLDB,* pp. 865–876, 2005.

[4] 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.

[5] Z. Ding, R. H. Güting. Modelling Temporally Variable Transportation Networks. In *Proc. DASFAA,* pp. 154–168, 2004.

[6] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, A. E. Abbadi. Constrained Nearest Neighbor Queries. In *Proc. SSTD,* pp. 257–278, 2001.

[7] R. H. Güting, V. T. de Almeida, and Z. Ding. Modeling and Querying Moving Objects in Networks. Fernuniversität Hagen, Informatik-Report 308, April 2004.

[8] 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.

[9] 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.

[10] H. Hu, D. L. Lee, J. Xu. Fast Nearest Neighbor Search on Road Networks. In *Proc. EDBT,* pp. 186–203, 2006.

[11] G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. In *TODS,* 24(2), pp. 265–318, 1999.

[12] 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.

[13] C. S. Jensen, J. Kolář, T. B. Pedersen, I. Timko. Nearest Neighbor Queries in Road Networks. In *Proc. ACMGIS,* pp. 1–8, 2003.

[14] M. Kolahdouzan and C. Shahabi. Voronoi-Based Nearest Neighbor Search for Spatial Network Databases. In *Proc. VLDB*, pp. 840–851, 2004.

[15] M. Kolahdouzan, C. Shahabi. Alternative Solutions for Continuous K Nearest Neighbor Queries in Spatial Network Databases. In *GeoInformatica,* 9 (4), pp. 321–341, 2005.

[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

Figure 16: Sample Data Pages