Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases

Mohammad Kolahdouzan and Cyrus Shahabi

Department of Computer Science
University of Southern California
Los Angeles, CA, 90089, USA
[kolahdoz,shahabi]@usc.edu

Abstract

A frequent type of query in spatial networks (e.g., road networks) is to find the K nearest neighbors (KNN) of a given query object. With these networks, the distances between objects depend on their network connectivity and it is computationally expensive to compute the distances (e.g., shortest paths) between objects. In this paper, we propose a novel approach to efficiently and accurately evaluate KNN queries in spatial network databases using first order Voronoi diagram. This approach is based on partitioning a large network to small Voronoi regions, and then pre-computing distances both within and across the regions. By localizing the pre-computation within the regions, we save on both storage and computation and by performing across-the-network computation for only the border points of the neighboring regions, we avoid global pre-computation between every node-pair. Our empirical experiments with several real-world data sets show that our proposed solution outperforms approaches that are based on on-line distance computation by up to one order of magnitude, and provides a factor of four improvement in the selectivity of the filter step as compared to the index-based approaches.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

1 Introduction

Many researchers have focused on the problem of K nearest neighbor (KNN) queries in spatial databases. This type of query is frequently used in Geographical Information Systems and is defined as: given a set of spatial objects (or points of interest), and a query point, find the K closest objects to the query. An example of KNN query is a query initiated by a GPS device in a vehicle to find the 5 closest restaurants to the vehicle. With spatial network databases (SNDB), objects are restricted to move on pre-defined paths (e.g., roads) that are specified by an underlying network. This means that the shortest network distance (e.g., shortest path, shortest time) between objects (e.g., the vehicle and the restaurants) depend on the connectivity of the network rather than the objects' locations.

The majority of the existing work on KNN queries are based on either computing the distance between a query and the objects on-line, or utilizing index structures. The solution proposed by the first group is based on the fact that the current algorithms (e.g., Dijkstra) for computing the distance between a query object q and an object O in a network will automatically result in the computation of the distance between q and the objects that are (relatively) closer to q than O. These approaches apply an optimized network expansion algorithm (e.g., [9]) with the advantage that the network expansion only explores the objects that are closer to q and computes their distances to q during expansion. However, the main disadvantage of these approaches is that they perform poorly when the objects are not densely distributed in the network because then they require to retrieve a large portion of the network for distance computation. The second group of approaches is designed and optimized for metric or vector spatial index structures (e.g., m-tree and r-tree, respectively). These approaches require precomputations of the distances between objects and object groups based on their distances to some reference nodes (this is more intelligent as compared to a naive

approach that pre-computes and stores distances between all the node-pairs in the network). These solutions filter a small subset of possibly large number of objects as the candidates for the closest neighbors of q, and require a refinement step to compute the actual distances between q and the candidates to find the actual nearest neighbors of q. The main drawback of applying these approaches on SNDB is that they do not offer any solution as how to efficiently compute the distances between q and the candidates. Moreover, applying an approach similar to the first group to perform the refinement step in order to compute the distances between q and the candidates will render these approaches, which traverse index structures to provide a candidate set, redundant since the network expansion approach does not require any candidate set to start with. In addition to this drawback, approaches that are based on vector index structures are only appropriate for spaces where the distance between objects is only a function of their spatial attributes (e.g., Euclidean distance) and cannot properly approximate the distances in a network (see [12]).

A comprehensive solution for spatial queries in SNDB must fulfill these real-world requirements: 1) be able to incorporate the network connectivity to provide exact distances between objects, 2) efficiently answer the queries in real-time in order to support KNN queries for moving objects, 3) be scalable in order to be applicable to usually very large networks, 4) be independent of the density and distribution of the points of interest, 5) be adaptive to efficiently cope with database updates where nodes, links, and points of interest are added/deleted, and 6) be extendible to consider query constraints such as direction or range.

In this paper, we propose a novel approach that fulfills the above requirements by reducing the problem of distance computation in a very large network, in to the problem of distance computation in a number of much smaller networks plus some additional table lookups. The main idea behind our approach, termed Voronoi-based Network Nearest Neighbor (VN^3), is to first partition a large network in to smaller/more manageable regions. We achieve this by generating a first-order *network* Voronoi diagram over the points of interest. Each cell of this Voronoi diagram is centered by one object (e.g., a restaurant) and contains the nodes that are closest to that object in *network* distance (and not the Euclidian distance). Next, we pre-compute the intra and inter distances for each cell. That is, for each cell, we pre-compute the distances between all the edges (or *border* points) of the cell to its center. We also pre-compute distances only across the border points of the *adjacent* cells. This will reduce the pre-computation time and space by localizing the computation to cells and handful of neighbor-cell node-pairs. Now, to find the k nearest-neighbors of a query object q, we first find the first nearest neighbor by simply locating the Voronoi cell that contains q. This can be easily achieved by utilizing a spatial index (e.g., R-tree) that is generated for the Voronoi cells. We prove that the next nearest neighbors of q are within the adjacent cells of the previously explored ones (see Section 4.1), which can be efficiently retrieved from a lookup table. We then utilize the intra-cell pre-computed distances to find the distance from q to the borders of the Voronoi cell of each candidate, and finally the inter-cell pre-computed distances to compute the actual network distance from q to each candidate (see Section 4.2). The local pre-computation nature of VN^3 also results in low complexity of updates when the network is modified.

Note that the application of the Voronoi diagrams to KNN queries have been extensively studied in computation geometry. The solution is based on calculating the K-th order Voronoi diagrams of a network. This solution is impractical for real-world scenarios since it requires that the value of K be predetermined. Moreover, K-th order Voronoi cells have complex shapes and the corresponding algorithms have a very high complexity. However, our proposed VN^3 approach utilizes only the first order Voronoi diagrams to only answer the $first$ NN queries. The other $(k-1)$ neighbors are found efficiently by utilizing our proven properties and the pre-computed distances.

To the best of our knowledge, the Incremental Network Expansion (INE) approach presented in [9] is the only other approach that efficiently supports the exact KNN queries on spatial network databases. However, this approach suffers from poor performance when the objects (e.g., restaurants) are not densely distributed in the network. Our empirical experiments with real-world data sets (presented in Section 5) show that VN^3 outperforms INE in query processing time by a factor of 1.5 to 12 depending on the density of the points of interest. Moreover, we show that the size of the candidate set generated by the filter step of VN^3 has lower variance and is up to four times smaller than that generated by the traditional approaches optimized for vector index structures. Also, we show that VN^3's performance is independent of the density and distribution of the points of interest, and the location of the query object. Finally, we show that the required computation and space for the pre-computation component of VN^3 is three orders of magnitude less than that of the naive solution that pre-computes all the node-pair distances.

The remainder of this paper is organized as follows. We review the related work on K nearest neighbor queries in Section 2. We then provide a review of the Voronoi diagrams, the basis of our proposed VN^3 approach, in Section 3. In Section 4, we discuss our proposed VN^3 approach and its extensions. Finally, we discuss our experimental results and conclusions in Sections 5 and 6, respectively.

2 Related Work

Numerous algorithms for k-nearest neighbor queries are proposed. This type of queries is extensively used in geographical information systems, shape similarity in image databases, pattern recognition, etc. A majority of the algorithms are aimed at m-dimensional objects and are based on utilizing one of the variations of multidimensional vector or metric index structures. There are also other algorithms that are based on precalculation of the solution space or the computation of the distance from a query object to its nearest neighbors on-line and per query. In this section, we consider each group in turn.

The algorithms that are based on index structures usually perform in two filter and refinement steps and their performance depend on their selectivity in the filter step. These approaches can be divided in two group: vector and metric index structures.

Vector Index structures: Roussopoulos et al. in [10] present a branch-and-bound R-tree traversal algorithm to find nearest neighbors of a query point. The main disadvantage of this approach is the depth-first traversal of the index that incurs unnecessary disk accesses. Korn et al. in [7] present a multi-step k-nearest neighbor search algorithm. The disadvantage of this approach is that the number of candidates obtained in the filter step is usually much more than necessary, making the refinement step very expensive. Seidl et al. in [11] propose an optimal version of this multi-step algorithm by incrementally ranking queries on the index structure. Hjaltason et al. in [4] propose an incremental nearest neighbor algorithm that is based on utilizing an index structure and a priority queue. All of these approaches are designed to utilize spatial index structures and aimed to minimize number of candidates, index nodes and disk accesses required to obtain candidates. There are two major shortages with these approaches that render them impractical for networks. first, networks are metric space, i.e., the distance between two objects depends on the connectivity of the objects and not their spatial attributes; however, the filter step of these approaches is based on Minkowski distance metrics (e.g., Euclidean distance). Hence, the filter step of these approaches cannot be used for, or properly approximate exact distances in networks. Second, these approaches do not propose any method to calculate the exact network distance between objects and the query in their refinement step, rather they assume that the distance function can be easily calculated.

Metric Index structures: These approaches are also based on a filter and refinement process, but as opposed to the vector index structures, they index and filter the objects considering their metric distance. Chiueh in [2] proposes Vantage Point (VP) tree structure for image indexing. This algorithm partitions a data set according to the distances between the objects and a reference (vantage) point. The median value of the distances is used to separate the objects into balanced subsets and a recursive algorithm is applied on each subset. This approach builds the tree based on a top-down recursive process, which does not guarantee a balanced tree. Ciaccia et al in [3] propose M-tree, a balanced tree that partitions objects based on their relative distances and stores these objects into fixed-sized nodes. The main disadvantage of these approaches is that they do not offer any solution on how to efficiently compute the distances between the query and the candidates (i.e., the same as the second shortage of the approaches based on vector index) which is required by the refinement step.

Berchtold et al. in [1] suggest precalculating, approximating and indexing the solution space for the nearest neighbor problem in m dimensional spaces. Precalculating the solution space means determining the Voronoi diagram of the data points. The exact Voronoi cells in m dimensional space are usually very complex, hence the authors propose indexing approximation of the Voronoi cells. This approach is only appropriate for the first nearest neighbor problem in high-dimensional spaces. Jung et al. in [6] propose an algorithm to find the shortest distance between any two points in a network. Their approach is based on partitioning a large graph into layers of smaller subgraphs and pushing up the pre-computed shortest paths between the borders of the subgraphs in a hierarchical manner to find the shortest path between two points. This approach can potentially be used in conjunction with one of the approaches that are based on metric index; however, the main disadvantage of this approach is its poor performance when multiple shortest path queries from different sources are issued at the same time.

Jensen et al. in [5] propose a data model and definition of abstract functionality required for NN queries in SNDB. They use algorithms similar to Dijkstra to calculate the shortest distance from a query to an object on-line. Finally, Papadias et al. in [9] propose a solution for nearest neighbor queries in network databases by introducing an architecture that integrates network and Euclidean information and captures pragmatic constraints. Their approach is based on generating a search region for the query point that expands from the query. The advantages of this approach are: 1) it offers a method that finds the exact distance in networks, and 2) the architecture can support other spatial queries like range search and closest pairs. Since the number of links and nodes that need to be retrieved and examined are inversely proportional to cardinality ratio of entities and number of nodes in the network, the main disadvantage of this approach is a dramatic degradation in performance when the above cardinality ratio is (far) less than 10%, which is the usual case for real world scenarios (e.g., the real

data sets representing the road network and different types of entities in the State of California show that the above cardinality ratio is usually between 0.04% and 3%). This is because spatial databases are usually very large and small values for the above cardinality ratio will lead to large portions of the database to be retrieved. This problem also happens for large values of K (see Section 5 for a thorough comparison).

3 Background: Voronoi Diagram

Our proposed approach to address the nearest neighbor queries is based on the *Voronoi diagram*. A Voronoi diagram divides a space into disjoint polygons where the nearest neighbor of any point inside a polygon is the generator of the polygon. In this section, we review the principles of the Voronoi diagrams. We start with the Voronoi diagram for *2-dimensional* Euclidean space and present only the properties that are used in our approach. We then discuss the *network Voronoi diagram* where the distance between two objects in space is their shortest path in the network rather than their Euclidean distance and hence can be used for spatial networks. A thorough discussion on regular and network Voronoi diagrams is presented in [8].

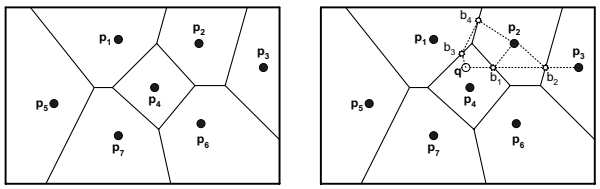
3.1 Definition

Consider a set of limited number of points, called *generator points*, in the Euclidean plane (in general, generators can be any type of spatial object). We associate all locations in the plane to their closest generator(s). The set of locations assigned to each generator forms a region called *Voronoi polygon* or *Voronoi cell*, of that generator. The set of Voronoi polygons associated with all the generators is called the Voronoi diagram with respect to the generators set. The Voronoi polygons of a Voronoi diagram are collectively exhaustive because every location in the plane is associated with at least one generator. The polygons are also mutually exclusive except for their boundaries. The boundaries of the polygons, called *Voronoi edges*, are the set of locations that can be assigned to more than one generator. The Voronoi polygons that share the same edges are called *adjacent polygons* and their generators are called *adjacent generators*. The Voronoi polygon and Voronoi diagram can be formally defined as: Assume a set of generators $P = \{p_1, ..., p_n\} \subset \Re^2$, where $2 < n < \infty$ and $p_i \neq p_j$ for $i \neq j$, $i, j \in I_n = \{1, ..., n\}$. The region given by:

$$VP(p_i) = \{p \mid d(p, p_i) \leq d(p, p_j)\} \text{ for } j \neq i, j \in I_n$$

where $d(p, p_i)$ specifies the minimum distance between p and p_i (e.g., length of the straight line connecting p and p_i in Euclidean space), is called the *Voronoi Polygon* associated with p_i, and the set given by:

$$VD(P) = \{VP(p_1), ..., VP(p_n)\}$$



a. Generators & Voronoi polygons b. Proof of Properties 5 & 7

Figure 1: Example of a Voronoi diagram

is called the *Voronoi Diagram* generated by P. Figure 1a shows an example of a Voronoi diagram, its polygons and generators.

3.2 Properties

We review four of the basic geometric properties of the Voronoi diagrams. The proofs for these properties are presented in [8]. These properties are the basis for the extended properties we introduce in Section 4.

Property 1: The Voronoi diagram of a point set P, $VD(P)$, is unique.

Property 2: The nearest generator point of p_i (e.g., p_j) is among the generator points whose Voronoi polygons share similar Voronoi edges with $VP(p_i)$.

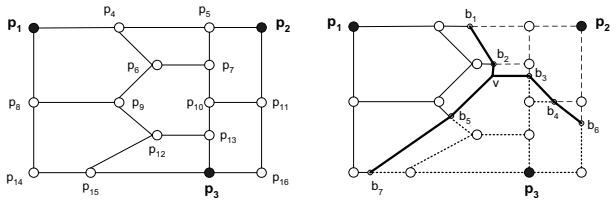
Property 3: Let n and n_e be the number of generator points and Voronoi edges, respectively, then $n_e \leq 3n - 6$.

Property 4: From property 3, and the fact that every Voronoi edge is shared by exactly two Voronoi polygons, we notice that the average number of Voronoi edges per Voronoi polygon is at most 6, i.e., $2(3n - 6)/n = 6 - 12/n \leq 6$. This means that on average, each generator has 6 adjacent generators. We use this property to derive the complexity of our algorithm.

3.3 Network Voronoi Diagram

A network Voronoi diagram ([8]), termed *NVD*, is defined for graphs and is a specialization of Voronoi diagrams where the location of objects is restricted to the links that connect the nodes of the graph and distance between objects is defined as the length of the shortest distance (e.g., shortest path or shortest time) in the network rather than their Euclidean distance. Spatial networks (e.g., road networks) can be modeled as weighted graphs where the intersections are represented by nodes of the graph and roads are represented by the links connecting the nodes. The weights can be the distances of the nodes or they can be the time it takes to travel between the nodes (representing shortest times).

Assume a weighted graph $G(N, L)$ that consists of a set of nodes $N = \{p_1, ..., p_n, p_{n+1}, ..., p_o\}$, where the first n elements (i.e., $P = \{p_1, ..., p_n\}$) are the generators (e.g., points of interest in a road network), and a set of links $L = \{l_1, ..., l_k\}$ that connects the nodes. Also assume that the network distance from a point



a. Original graph b. Network Voronoi diagram

Figure 2: Example of a network Voronoi diagram (NVD)

p on a link in L to p_i in N, $d_n(p, p_i)$, is defined as the shortest network distance from p to p_i. For all $j \in I_n \setminus \{i\}$, we define:

$$Dom(p_i, p_j) = \{p | p \in \bigcup_{o=1}^{k} l_o, d_n(p, p_i) \leq d_n(p, p_j)\}$$

$$b(p_i, p_j) = \{p | p \in \bigcup_{o=1}^{k} l_o, d_n(p, p_i) = d_n(p, p_j)\}$$

The set $Dom(p_i, p_j)$, called the *dominance region* of p_i over p_j on links in L, specifies all points in all links in L that are closer to p_i or of equal distance to p_j. The set $b(p_i, p_j)$, called *bisector* or *border points* between p_i and p_j, specifies all points in all links in L that are equally distanced from p_i and p_j. Consequently, the *Voronoi link set* associated with p_i and *network Voronoi diagram* are defined as following, respectively:

$$V_{link}(p_i) = \bigcap_{j \in I_n \setminus \{i\}} Dom(p_i, p_j)$$

$$NVD(P) = \{V_{link}(p_1), ..., V_{link}(p_n)\}$$

where $V_{link}(p_i)$ specifies all the points in all the links in L that are closer to p_i than any other generator point in N. Similar to VD defined in Section 3.1, elements of NVD are also collectively exhaustive and mutually exclusive except for their border points. Note that b is a set of points which unlike Voronoi diagram in Euclidean space, cannot directly generate polygons. However, by properly connecting adjacent border points of a generator g to each other without crossing any of the links, we can generate a bounding polygon, called *network Voronoi polygon*, we term $NVP(g)$, for that generator. Note that generation of $NVP(g)$ only requires local network information, i.e., the links and nodes that are in the area between g and its adjacent generators are used to generate $NVP(g)$.

An example of NVD is shown in Figure 2. Figure 2a depicts the original graph where p_1, p_2 and p_3 are the generators. We can assume that the set of generators is the set of *points of interest* (e.g., hotels, restaurants,...) and p_4 to p_{16} are the intersections of a road network that are connected to each other by the set of streets L. Figure 2b shows the NVD of the graph where each line style corresponds to a Voronoi link set of a generator. As shown in the figure, some links are completely contained in V_{link} of a generator (e.g., the link connecting p_6 and p_9 is completely inside $V_{link}(p_1)$), while others are partially contained in different V_{link}'s (e.g., the link connecting p_4 and p_5 is divided between and contained in $V_{link}(p_1)$ and $V_{link}(p_2)$). The figure also shows how adjacent border points should be connected to each other: if two adjacent border points are between two similar generators (e.g., b_5 and b_7 are between p_1 and p_3), they can be connected with an arbitrary line that does not cross any of the members of L. Three or more adjacent border points (e.g., b_2, b_3 and b_5) can be connected to each other through an arbitrary auxiliary point (e.g., v in the figure). By using arbitrary lines and auxiliary points, NVPs will become non-unique. However, since objects in a graph can only be located on links, different NVPs will contain exactly identical Voronoi link sets and hence are unique in this respect. Moreover, as shown in the figure and unlike Voronoi polygons in the Euclidean space, common edges between two NVPs may contain more than two border points and are not necessarily straight lines. Despite this, properties 3 and 4 of Section 3.2 are still valid for NVPs as shown in [8].

4 Voronoi-Based Network Nearest Neighbor: VN^3

In this section, we describe VN^3 as our proposed approach to evaluate the nearest neighbor queries in spatial networks. VN^3 is based on the properties of the Network Voronoi diagrams and also *localized* pre-computation of the network distances for a very small percentage of neighboring nodes in the network. The intuition is that the NVPs of an NVD can directly be used to find the first nearest neighbor of a query object q. Subsequently, NVPs' adjacency information can be utilized to provide a candidate set for other nearest neighbors of q. Finally, the pre-computed distances can be used to compute the actual network distances from q to the generators in the candidate set and consequently refine the set. The filter/refinement process in VN^3 is iterative: at each step, first a new set of candidates is generated from the NVPs of the generators that are already selected as the nearest neighbors of q, then the pre-computed distances are used to select "only the next" nearest neighbor of q. Hence, the filter/refinement step must be invoked k times to find the first k nearest neighbors of q. Note that this is different from the usual filter/refinement process where the two steps are invoked consecutively. VN^3 consists of the following components:

1. Pre-calculation of the solution space: As a major component of the VN^3 filter step, the NVD for the points of interest (e.g., hotels, restaurants,...) in a network must be calculated and its corresponding NVPs must be stored in a table. Note that sep-

arate NVDs and set of NVPs must be generated for different types of points of interest.

2. Utilization of an index structure: In the first stage of the filter step, the first nearest neighbor of q is found by locating the NVP that contains q (e.g., using *Contain(q)* function in spatial databases). This stage can be expedited by using a spatial index structure generated on the NVPs. Note that although an NVD is based on the network distance metric, its NVPs are regular polygons and can be indexed using index structures that are designed for the Euclidean distance metric (e.g., R-tree). This means that the *Contain(q)* function invoked on an R-tree index structure on NVPs will return the NVP whose generator has the minimum network distance to q.

3. Pre-computation of the exact distances for a very small portion of data: The refinement step discussed in Section 4.2 requires that for each NVP, the network distances between its border points be pre-computed and stored. These pre-computed distances are used to find the network distances across NVPs, and from the query object to the candidate set generated by the filter step.

4.1 VN^3 Filter Step

Our proposed approach to generate the candidate set for nearest neighbors of a query point is based on the first two components of VN^3 discussed in Section 4. This requires the pre-calculation of NVD and generation of a spatial index structure for NVPs of the NVD. We first introduce the following properties that can be concluded from properties 1 to 4 in Section 3.2. These two properties help the filter step to constraint its search space to only the adjacent Voronoi polygons.

Property 5: Property 2 suggests that the second nearest generator to *"any location"* inside a Voronoi polygon $V(p_i)$ is among the adjacent generators of p_i.
Proof: This property can be proved by contradiction. Consider Figure 1b where the first nearest neighbor of an arbitrary point q is p_4 (i.e., q is inside $V(p_4)$). Now suppose that the second nearest neighbor of q is $p_3 \notin$ {adjacent generators of p_4}. This requires that the shortest path between q and p_3, $L(q, p_3)$, intersects with at least one of the adjacent polygons of $V(p_4)$ ($L(q, p_3)$ in Figure 1b intersects $V(p_2)$ at points b_1 and b_2). Note that in case of NVD, $L(q, p_3)$ may not be a straight line as shown in the figure. From the definition of the Voronoi polygons we know that $d_n(b_2, p_3) = d_n(b_2, p_2)$. We also know that the shortest path in networks and Euclidean distance functions obey triangular inequality, meaning that $d_n(b_1, b_2) + d_n(b_2, p_2) \geq d_n(b_1, p_2)$. We can conclude that $d_n(b_1, b_2) + d_n(b_2, p_3) \geq d_n(b_1, p_2)$, and by adding $d_n(q, b_1)$ to both sides of the inequality, we can ultimately conclude that $d_n(q, p_3) \geq d_n(q, p_2)$. This means that p_2 is a closer generator to q than p_3, or at least has the same distance, which contradicts our initial assumption.

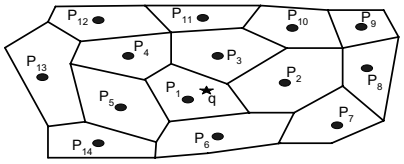


Figure 3: Sample network Voronoi diagram

Property 6: Let $G = \{g_1, ..., g_k\} \in P$ be the set of the first k nearest generators of a location q inside $V(g_1)$, then g_k is among the adjacent generators of $\{G \setminus g_k\}$. This property is in fact the generalization of Property 5.
Proof: The proof of this property by contradiction is similar to the proof of the previous property. We know that the shortest path from q to g_k, $L(q, g_k)$, must intersect with one of the edges of $V(g_k)$, e.g., E_k. Suppose $L(q, g_k)$ intersects E_k at point b_k, and E_k is a common edge between $V(g_k)$ and $V(x)$ where the contradiction of this property requires that $x \notin \{G \setminus g_k\}$. It is clear that $L(q, g_k)$ must also intersect with $V(x)$ at at least another point like b_x. By comparison with Property 5, we can conclude that $d_n(b_x, x) \leq d_n(b_x, b_k) + d_n(b_k, g_k)$, suggesting that x is closer to q than g_k and hence $x \in \{G \setminus g_k\}$. This contradicts our initial assumption.

Using a hypothetical NVD shown in Figure 3, we can now easily describe our filtering approach to generate the candidate set, C, for nearest neighbors of a query point q. The definition of NVD requires that the first nearest neighbor of q be P_1 since $V(P_1)$ contains q, hence P_1 can be found by issuing the *Contain(q)* function on an index structure generated for the NVPs of the NVD. Property 5 suggests that the second nearest neighbor of q is among the adjacent generators of P_1, i.e., $C = \{P_2, P_3, P_4, P_5, P_6\}$. Note that during the generation of NVD, the adjacent generators/NVPs are determined. We store the adjacency information of the NVPs in a lookup table. Hence, finding adjacent NVPs does not require any spatial operation, rather, they can easily be found from a lookup table by one disk block access. At this stage, we need to invoke the refinement step (see Section 4.2) to compute the exact distances between q and all the generators in C to find the second nearest neighbor. Let us assume that the second nearest neighbor of q is P_3. Property 6 requires that the third nearest neighbor of q be among the adjacent generators of $\{P_1, P_3\}$, i.e., $C = \{P_2, P_4, P_5, P_6, P_{10}, P_{11}, P_{12}\}$. Note that P_{12} is adjacent to P_3 as their polygons share the same vertex. Consecutive nearest neighbors of q can then be found using the same iterative approach.

4.1.1 Analysis

In this section, we analyze the complexity of the filter step of VN^3 with respect to the size of the candidate

set as well as the number of disk block accesses.

Size of the Candidate Set: As described in Section 4.1, the first neighbor is found by applying the *contain()* function on an R-tree index that is generated for NVPs, and hence does not require any distance computation. Property 4 in Section 3.2 suggests that on average, 6 generators have to be examined to explore the second nearest neighbor. Property 6 suggests that at least one of the adjacent generators of any newly found neighbor must have already been explored as a nearest neighbor. Hence, from the second nearest neighbor on, exploration of a new nearest neighbor will lead to only 5 (on average) new generators that must be examined to find the next nearest neighbor. This will lead to a candidate set with an average size of $(5K + 1)$, equal to complexity of $O(K)$. This is a conservative bound as our experimental results (Section 5) show that the average size of the candidate set is usually much smaller than $(5K + 1)$, and becomes very close to K as the value of K increases.

Disk Block Accesses: Usage of the *Contain()* function on the R-tree index to find the first nearest neighbor incurs a complexity of $O(log(n))$, where n is the number of generators of the network. We also showed that $O(K)$ generators have to be examined and hence retrieved from the database. Hence, the complexity of the number of disk block accesses required by the filter step becomes $O(K + log(n))$.

4.2 VN^3 Refinement Step

As we discussed in Section 4.1, once a nearest neighbor of a query point q is found and the candidate set C is updated, the distances from q to all the elements of C must be computed in order to find the next nearest neighbor. In this section, we discuss alternative approaches that are based on properties of NVPs to find the distances between q and the elements of C.

The intuition behind all the proposed approaches is that in an NVD, all possible paths that can connect an object from outside an NVP to a node inside it, including the polygon's generator, must pass through the border points of the polygon. In the sequel we use $BoP(e)$ to specify the set of border points of an entity e. During the generation of the NVPs (described in [8]), the shortest distance between the border points of a Voronoi polygon to the polygon's generator is determined and stored in a lookup table. Hence, if we calculate the distance from the outside object to the border points of a Voronoi polygon, we can find the minimum distance from the object to the generator of the polygon. As an example, consider the NVD shown in Figure 4 where $N = \{P_1, ..., P_{14}\}$ are the generators of the NVD, e.g., points of interest in a road network. Note that only the nodes and links inside $NVP(P_1)$ are drawn. In the figure, $B = \{b_1, ..., b_{40}\}$ specify the border points of the NVPs. An example that describes the intuition is the distance computation between q and P_9. The values of $d_n(P_9, b_{34})$, $d_n(P_9, b_{35})$ and $d_n(P_9, b_{36})$ are determined during the generation of $NVP(P_9)$. Hence, we can compute the distance between q and P_9 as:

$$\begin{aligned} d_n(q, P_9) = \quad & min(d_n(q, b_{34}) + d_n(b_{34}, P_9), \\ & d_n(q, b_{35}) + d_n(b_{35}, P_9), \\ & d_n(q, b_{36}) + d_n(b_{36}, P_9)) \end{aligned}$$

Trivially, we need to have the distance from q to b_{34}, b_{35}, and b_{36} in order to find $d_n(q, P_9)$. As stated above, q can only be connected to $\{b_{34}, b_{35}, b_{36}\}$ through $\{b_1, ..., b_8\}$, which are the border points of the NVP that contains q, $BoP(NVP(q))$. This means that:

$$\begin{aligned} d_n(q, b_{34}) = \quad & min(\ d_n(q, b_1) + d_n(b_1, b_{34})\ ,\ ...\ , \\ & d_n(q, b_8) + d_n(b_8, b_{34})\ ) \end{aligned}$$

As shown above, finding the network distance from q to P_9, or in general to all the elements of C, requires: 1) Query to border computation: computing the network distances from q to $BoP(NVP(q))$ and 2) Border to border computation: for the set of generators that are candidates for the next nearest neighbor, g, computing the network distances from $BoP(NVP(q))$ to the border points of the NVPs of g, $BoP(NVPs(g))$. We address these two requirements in the following sections.

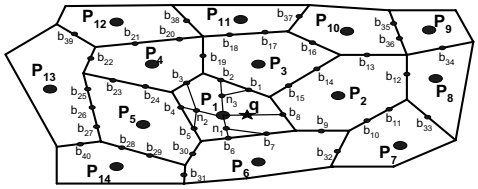


Figure 4: Sample network Voronoi diagram

4.2.1 Query to Border Computation

We propose two alternative approaches to compute the distances from a query point q to the border points of its surrounding NVP, $BoP(NVP(q))$.

1. On-line Progressive Expansion, OPE: With this approach, we use a method similar to what proposed in [9] to find the distance between q and $BoP(NVP(q))$. The average number of disk block accesses required by this approach is $\frac{n}{mb}$ (m points of interest, n nodes in the network, b connected nodes on one disk block) which grows rapidly for large networks with small number of points of interest (see Section 5. Due to lack of space and because our chosen approach in the experiments is OPC, we do not include the discussion for this approach. See the full-paper or [9] for details.

2. Off-line Precalculation, OPC: This approach suggests that in addition to the third component of VN^3 discussed in Section 4, the distances from each border point to all the nodes inside the polygons that contain the border point be pre-computed in an off-line process.For example, for the NVD shown in Figure 4, this approach suggests that in addition to the distances from $(b_1, ..., b_8)$ to each other and to P_1, their distances to (n_1, n_2, n_3) be pre-computed. Note

that unlike the figure, number of border points of an NVP in a real world scenario is much less than the number of nodes inside the NVP (see results in Section 5). Also note that each border point is shared by at least two NVPs and hence its distances to all the nodes in these NVPs have to be computed, e.g., the distances from (b_6, b_7) to the nodes inside $NVP(P_6)$ should also be computed.

The main advantage of this approach is a significant boost in performance since it eliminates the need for both the execution of complex algorithms (e.g., Dijkstra in OPE) and retrieving large amount of data for distance computations. Rather, the distance from q to $BoP(NVP(q))$ can be retrieved from a lookup table in one disk block access. The disadvantage of this approach is the requirement for an off-line process to pre-calculate and store the above network distances. However, we believe that this approach has a low overall computation and space complexity (similar to the discussion in Section 4.2.2). Our experimental results in Section 5 confirm this intuition, where they show that the network distances for only less than 0.5% of the network (as compared to pre-computing the distances between all the nodes in the network) must be calculated and stored.

4.2.2 Border to Border Computations

To find the network distances from $BoP(NVP(q))$ to the border points of the NVP of any generator, $BoP(NVP(g))$, we propose pre-computing the point-to-point network distances between the border points of each NVP "separately" (the third component of VN^3 discussed in Section 4). For example, this approach suggests that for the NVD shown in Figure 4, the point-to-point network distances among $\{b_1, ..., b_8\}$ (corresponding to $NVP(P_1)$) be pre-computed. It also suggests that the point-to-point network distances among $\{b_1, b_2, b_{14}, ..., b_{19}\}$ (corresponding to $NVP(P_3)$) be pre-computed. Note that each border point (e.g., b_1) belongs to at least two NVPs (e.g., $NVP(P_1)$ and $NVP(P_3)$) and hence, its distances to all the border points of two NVPs must be pre-computed. The intuition for this approach is that once the point-to-point network distances among the border points of "each" NVP is computed, these distances can be used to find the network distances between the border points of "any" two NVPs. The other intuition is that this approach has low complexity with respect to both space and computation. The reasons are: 1) The pre-computation is only performed for the border points of each NVP separately, and in real world scenarios (as opposed to the example shown in Figure 4), the ratio of the total number of the border points to the total number of the nodes in the network is small (see Section 5), and 2) the pre-computation is performed for each NVP separately and not across all NVPs, and the border points of each NVP are fairly close to each other.

The first stage to find the network distances from $BoP(NVP(q))$ to $BoP(NVP(g))$ is to find those NVPs through which the shortest path from q to g will pass. In order to find these NVPs, we introduce the following properties that can be concluded from properties 5 and 6 in Section 4.1.

Property 7: In a Voronoi diagram, if (g_1, g_2) is the set of first two nearest generators of q, then the shortest path from q to g_2 can only go through $\{V(g_1), V(g_2)\}$ and hence, through the common edge of $\{V(g_1), V(g_2)\}$.

Proof: The proof of this property is by contradiction. If the shortest path from q to g_2 crosses a Voronoi polygon $V(g_k)$ where $g_k \notin \{g_1, g_2\}$, then the portion of the shortest path that is inside $V(g_k)$ is closer to g_k than g_2 and consequently, q will become closer to g_k than g_2. This contradicts our assumption that g_2, after g_1, is the nearest generator to q. As an example, consider Figure 1b and suppose that $\{p_4, p_2\}$ is the set of first two nearest generators of q. Suppose that the shortest path from q to p_2 is the line $L =< q, b_3, b_4, p_2 >$, which crosses $NVP(p_1)$ at points b_3 and b_4. Since any point on the line segment $< b_3, b_4 >$ is closer to p_1 than p_2, this requires that p_1 must be a closer generator to q than p_2, contradicting our assumption about the set of first two nearest generators of q. Note that in network Voronoi Diagrams, the shortest network path from q to g_2 can only cross the common edge of $V(g_1)$ and $V(g_2)$ at one of their common border points.

Property 8: In a Voronoi diagram, if $(g_1, ..., g_k)$ is the set of first k nearest generators of q, then the shortest path from a location q to g_k can only go through a combination of $\{V(g_1), ..., V(g_k)\}$ and hence, through a combination of the common edges of $\{V(g_1), ..., V(g_k)\}$.

Proof: This property is the generalization of Property 7 and its proof is similar to the proof of Property 7.

Property 9: In a Voronoi diagram, if the shortest path from q to a generator g_k passes through $NVP(g_i)$, then g_i is closer to q than g_k.

Proof: Property 8 suggests that the shortest path from q to g_k can only pass through a combination of $\{V(g_1), ..., V(g_{k-1})\}$. Hence, g_i must be a member of $\{V(g_1), ..., V(g_{k-1})\}$ and subsequently, it is closer to q than g_k.

Using the hypothetical NVD shown in Figure 4, we now describe our progressive approach to find the distances from q to $BoP(NVPs(CG))$ (CG are candidates for the next nearest generator of q). As shown in the figure, the first nearest generator of q is p_1 since $NVP(p_1)$ contains q. Property 7 suggests that the shortest path from q to its second nearest generator, say p_i, can only go through the common edges between $NVP(p_1)$ and the $NVP(p_i)$. Hence, to find the second nearest generator of q, we first compute the *minimum possible network distances*, d_{mpn}, from q to $(p_2, ..., p_6)$ through each of the generators' shared border points

with p_1:

$$d_{mpn}(q,p_2) = \quad d_n(q,b_8) + d_n(b_8,p_2)$$
$$d_{mpn}(q,p_3) = \quad min[\ d_n(q,b_1) + d_n(b_1,p_3)\ ,\ d_n(q,b_2) + d_n(b_2,p_3)\ ]$$
$$...$$
$$d_{mpn}(q,p_6) = \quad min[\ d_n(q,b_6) + d_n(b_6,p_6)\ ,\ d_n(q,b_7) + d_n(b_7,p_6)\ ]$$

Formally, the minimum possible network distance from q to p_i, $d_{mpn}(q,p_i)$, is the minimum distance from q to p_i for a path that only passes through any of the NVPs whose generators are already selected as the nearest generators of q. Hence, the path can only crosses the common edges between those NVPs. Note that all of the above d_n's are either pre-computed (as we proposed in Section 4.2.2) or calculated using one of the approaches discussed in Section 4.2.1. The generator with the shortest d_{mpn} is then selected as the second nearest generator of q. Note that because of the triangular inequality property, $d_{mpn}(q,p_i) \geq d_n(q,p_i)$. However, Property 7 (and 8) require that $d_{mpn}(q,p_i) = d_n(q,p_i)$ when p_i is the next nearest generator of q. Let us now assume that p_2 is the second nearest generator of q. Property 8 suggests that the shortest path from q to the third nearest neighbor, $p_i \in (p_3,p_4,p_5,p_6,p_7,p_8,p_{10})$, can only go through one of the $\{\ BoP(\{NVP(p_1) \cup NVP(p_2)\}) \cap BoP(NVP(p_i))\ \}$. For example, the property requires that at this stage, the path from q to p_6 can only go through b_6, b_7, or b_9:

$$d_{mpn}(q,p_6) = \quad min[\ d_n(q,b_6) + d_n(b_6,p_6)\ ,\ d_n(q,b_7) + d_n(b_7,p_6)\ ,\ d_n(q,b_8) + d_n(b_8,b_9) + d_n(b_9,p_6)\ ]$$

Note that the actual shortest path from q to b_9 may pass through a different point set than (b_8) (e.g., $SP(q,b_9)$ may pass through (b_1,b_{15})). However, at this stage, where only p_1 and p_2 are found as the nearest generators of q, Property 8 suggests if $SP(q,p_6)$ does indeed go through b_9, then $SP(q,b_9)$ must go through b_8. In other words, the "only possible" path from q to b_9 at this stage is through b_8 and hence, only the length of this path must be computed for $d_{mpn}(q,p_6)$. This significantly simplifies our approach by replacing the need to compute the "actual shortest" network distances from q to $BoP(NVPs(CG))$, with the need to only compute their "minimum possible" network distances. If we assume that p_3 is the third nearest generator of q, then Property 8 suggests that the minimum possible shortest path from q to b_9, in addition to going through b_8, can pass through a combination of (b_1,b_2) and (b_{14},b_{15}) as well.

We now propose two alternative approaches to find d_{mpn} from a query point to $BoP(NVPs(CG))$. The intuition for both of the approaches is that at each step k, we find the d_{mpn} from q to $BoP(p)$, where $P = \{NVP(g_1) \cup NVP(g_2) \cup ... \cup NVP(g_{k-1})\}$ and $\{g_1,...,g_{k-1}\}$ is the set of the $(k-1)$-st nearest generators of q.

1. NVP Expansion, NVP-E: This approach works as follows. For the NVD shown in Figure 4, first we generate an auxiliary network, AN_1, containing nodes $N = \{b_1,...,b_8,q\}$ (i.e., $BoP(NVP(p_1))$ and the query object). Note that all of the elements of AN_1 are connected to each other and their distances are equal to their network distances in the original network that are either pre-computed as part of the third component of VN^3 (Section 4), or are calculated by one of the approaches discussed in Section 4.2.1. We use the distances from q to $\{b_1,...,b_8\}$ to find the second nearest generator (assume P_2 is the second nearest generator). We then generate a new auxiliary network, AN_2, containing nodes $N = \{b_8,...,b_{15}\}$ (i.e., $BoP(NVP(P_2))$). Note that based on Property 6, the NVP of the k-th nearest generator must have at least one common edge/border point with the NVPs of the first $(k-1)$-st nearest generators. Hence, we generate a new network from AN_1 and AN_2, $AN = \{AN_1 \cup AN_2\}$, and using Dijkstra's algorithm, compute the distances from q to the nodes in the new network AN. Note that computing the network distances between all the nodes in AN is not necessary. At this stage, we have the minimum possible network distances from q to the nodes in AN, i.e., $BoP(\{NVP(p_1) \cup NVP(p_2)\})$. Consequently, the minimum possible network distances for the k-th step can be found by generating $AN = \{AN_1 \cup ... \cup AN_{k-1}\}$. The intuition here is that the average number of border points for each NVP is small (our experimental results in Section 5 confirms this), and hence, this approach can be efficiently executed in memory.

2. Distance Computing Optimization, DCO: The use of the Dijkstra's algorithm in NVP-E requires that the distances from q to all the nodes in AN be recalculated every time a new nearest generator is explored. This is unnecessary because once a new nearest generator of q is found and its NVP is added to AN, d_{mpn} from q to only a very small number of borders in AN must be reexamined. The distance d_{mpn} from q to the border points of an NVP may "only" change if the distance from q to at least one of the border points of that NVP is changed. Consequently, DCO works as follows: suppose $\{g_1,...,g_{n-1}\}$ is the set of the first $(n-1)$-st nearest generators of q and d_{mpn}'s from q to all the $BoP(\mathcal{V} = \{NVP(g_1) \cup ... \cup NVP(g_{n-1})\})$ are computed. When the next nearest generator, g_n, is explored, first the length of the minimum possible paths from q to $BoP(NVP(g_n))$ are computed. These paths can only pass through $BoP(\{NVP(g_n) \cap \mathcal{V}\})$. Next, only d_{mpn} for the NVPs whose border point(s) have a new smaller value of d_{mpn} are re-examined. This will reduce the computation complexity by eliminating unnecessary distance computations.

4.3 VN^3 Storage Schema

Figure 5 shows an example of a simple schema needed for the NVD of Figure 4. The proposed schema consists of a spatial component (NVPs component in the figure) that is used to find the first NN of a query, and

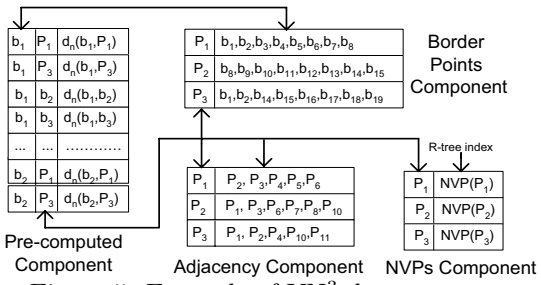


Figure 5: Example of VN^3 data structures

three look up tables: an adjacency component that is used for the filter step of VN^3, a pre-computed component that is used for the refinement step, and a border point component.

4.4 VN^3 Updates

The intuition for updates in VN^3 is that the modification of the original network may require changes in the original NVPs or may require some of the network distances to be recomputed. Due to "local precomputation" nature of VN^3, an update in the network does not result in updating the entire network Voronoi diagram; rather, it only affects the NVPs in a local scope.

a. Adding/Removing Links/Nodes: Suppose a link L or a node N is added to or removed from a network. If L/N is contained in one NVP, the distances from the border points of that NVP to each other is recomputed. If these distances stay the same as before, then the NVP's shape will remain the same; otherwise, the NVP must be recomputed. Subsequently, if the distances from the border points of adjacent NVPs are also changed, those NVPs are re-examined. If L is contained in a set of NVPs, $\mathcal{N}$, then the adjacent NVPs of $\mathcal{N}$ may change and hence must be regenerated.

b. Adding/Removing Points of Interest: Suppose a point of interest P is added to, or removed from the network. Then only the $NVP(P)$ and (some of) its adjacent NVPs will change (i.e., become smaller/larger). For example, for the NVD shown in Figure 5, if a new point of interest P located in $NVP(p_1)$ is added to the network, then $NVP(p_1)$ and some of the $\{NVP(p_2), ..., NVP(p_6)\}$ will change. Similarly, if p_1 is removed, then the area covered by $NVP(p_1)$ will be covered by $\{NVP(p_2), ..., NVP(p_6)\}$.

5 Performance Evaluation

We conducted several experiments to: 1) compare the performance of VN^3 with its only competitor, the INE approach presented in [9], 2) evaluate the overhead of the pre-computations proposed in Sections 4 and 4.2.1, and 3) compare the performance of the filter step of VN^3 with that of the traditional approaches optimized for spatial index structures. We used two real-world data sets. The first data set is obtained from NavTech Inc., used for navigation systems with GPS devices installed in cars, and represents a network of approximately 110,000 links and 79,800 nodes of the road system in the downtown Los Angeles. The second data set is obtained from USGS and consists of a set of points representing hospitals, major buildings, and churches in the US containing approximately 5200, 14000 and 126000 objects, respectively. The experiments were performed on an IBM ZPro with dual Pentium III processors, 512MB of RAM, and Oracle 9.2 as the database server. We present the average results of 1000 runs of K nearest neighbor queries where K varied from 1 to 500.

1. Overall Performance of VN^3:

Our experiments show that the total query response time of VN^3 is up to one order of magnitude less than that of INE. Table 1 shows the results of comparing query response time between VN^3 and the INE approach proposed in [9]. The first and second columns specify the entities (or points of interest) and their population and cardinality ratio (i.e., the number of entities over the number of links in the network), respectively. Note that for the given data set, restaurants and hospitals represent the entities with the maximum and minimum cardinality ratios. As shown in the table, when $K = 1$, and regardless of the density of the entities, VN^3 generates the result set almost instantly. This is because a simple *contain*() function is enough to find the first NN. However, depending on the density of the entities, INE approach requires between 0.49 to 12.4 seconds to provide the first NN. Also, both approaches have almost similar CPU processing times (values inside "()"), with VN^3's CPU time tend to be more than INE's for larger values of K. This is because the major computation component in INE is maintaining a sorted queue which grows for larger values of K (e.g., for auto services, the queue size is 50 for $K = 5$ and 6600 for $K = 250$). However, VN^3 has a more complex computation requirement as it requires computation of the distances from the query to the borders of a newly found neighbor every time a new neighbor is explored. However, as shown in the table, the time required by the database to retrieve the links from network is the dominant factor and the CPU times are almost negligible. Depending on the density of the entities, the time incurred by INE to retrieve the network from the database is between 1.5 (for high densities and larger Ks) and 12 (for low densities and higher values of K) times more than that incurred by VN^3. This is because for lower densities of entities, INE requires larger portion of the network to be retrieved. For example, while there are only 340 links retrieved from the database to find the 10 closest restaurants to a query, 17900 links (equal to 16% of the network) need to be retrieved to find the 10

		Query Processing Time (Sec.)											
		K=1		K=5		K=10		K=25		K=50		K=100	
		VN^3	INE	VN^3	INE	VN^3	INE	VN^3	INE	VN^3	INE	VN^3	INE
Entities	Qty (density)	(cpu) disk	(cpu) disk	(cpu) disk	(cpu) disk	(cpu) disk	(cpu) disk	(cpu) disk	(cpu) disk	(cpu) disk	(cpu) disk	(cpu) disk	(cpu) disk
Hospital	46 (0.0004)	(0) 0.018	(0.3) 12.4	(1.5) 6.5	(1.7) 78.3	(4.5) 14.0	(3.8) 165.1	(15.3) 35.1	(10.1) 430.2	- -	- -	- -	- -
Shopping Centers	173 (0.0016)	(0) 0.020	(0.09) 3.6	(0.45) 3.3	(0.5) 21.1	(1.3) 6.9	(1.1) 44.0	(3.4) 18.1	(3.1) 118.0	- -	- -	- -	- -
Parks	561 (0.0053)	(0) 0.021	(0.03) 1.4	(0.15) 1.5	(0.2) 8.2	(0.37) 2.8	(0.3) 15.3	(1.4) 6.4	(0.8) 36.4	(2.5) 13.3	(1.6) 71.1	- -	- -
Schools	1230 (0.0115)	(0) 0.027	(0.015) 0.6	(0.06) 0.75	(0.07) 3.5	(0.18) 1.46	(0.14) 6.6	(0.7) 3.9	(0.36) 15.6	(1.9) 7.5	(0.7) 32.2	- -	- -
Auto Services	2093 (0.0326)	(0) 0.030	(0.013) 0.57	(0.01) 0.65	(0.05) 2.43	(0.09) 1.4	(0.09) 4.3	(0.58) 2.95	(0.23) 10.0	(1.65) 6.68	(0.44) 19.4	(2.78) 13.1	(0.87) 38.00
Restaurants	2944 (0.0580)	(0) 0.032	(0.01) 0.49	(0.01) 0.57	(0.03) 1.34	(0.04) 1.48	(0.06) 2.7	(0.26) 2.8	(0.15) 6.8	(0.8) 6.1	(0.3) 13.3	(1.85) 12.8	(0.6) 26.0

Table 1: Query processing time of VN^3 vs. INE

closets hospital to the same query object. Note that INE does not retrieve the required links in one step, rather, only a small number of links are retrieved from the database at each step. Note that VN^3 also requires pre-computed values to be retrieved from the database, and the number of required pre-computed values increases for lower densities of the entities and larger values of K. However VN^3 retrieves the required data in only one step, resulting in much faster data retrieval time.

Table 2 shows the overhead incurred by the pre-computations proposed as the third component of VN^3. As shown in the table, for entities with higher densities (e.g., restaurants) which generate smaller and more number of NVPs, the average number of nodes inside each NVP and number of border points per NVP are less. This will lead to faster pre-computation process since the pre-computations are performed in smaller size local areas. The third column of the table shows the total number of border-to-border pre-computations, which is almost constant for entities with different densities. This is because when there are more number of NVPs (e.g., restaurants), the average number of border points are smaller and when there are less number of NVPs (e.g., hospital), the average number of border points are larger. Finally, the suggested pre-computations for off-line precalculation method (Section 4.2.1), fourth column in the table, increases for entities with smaller densities. This is because the average number of nodes inside each NVP grows rapidly. Note that a naive approach that pre-computes all the pair node distances in the given network requires 3.2 billion pre-computations. However, in VN^3, the highest number of pre-computations required by OPC method is still three order of magnitude less than that of the naive approach.

2. Performance of the VN^3 Filter Step: Figure 6 depicts the performance of the VN^3 filter step with respect to the size of the candidate set when KNN queries are performed for the second data set. For each value of K (x-axis) we performed 1000 queries where the location of the query point is randomly selected, and we averaged the results. Two observations can

Entities	Points inside each NVP	Average BPs per NVP	Number of Pre-comp. Bor-Bor	Number of Pre-comp. OPC
Hospital	1698	52	232,000	8,781,000
Shopping	458	25	225,600	4,653,000
Parks	142	14	239,500	2,630,000
Schools	64	10	246,000	1,787,000
Auto Svc.	38	7	239,900	1,611,000
Restaurants	27	6	243,600	1,348,000

Table 2: Overhead of pre-computations

be made from the figure. First, the ratio of the size of the candidate set (we term SKS) over K decreases as K increases. For example, while 13 candidates are selected when K=3 (4.3 times the value of K), only 25 candidates are selected when K=10 (2.5 times the value of K). The figure also shows that for large values of K, the size of the candidate sets become very close to K. The reason for this is that as K increases, once a generator g is explored as the K-th nearest neighbor of a query object q, the possibility that some of its adjacent generators have already been explored as the $(K-1)$-st nearest neighbors and no longer need to be examined increases. This is a very important feature of VN^3's filter step since for large values of K, the average number of points of interest that must be examined significantly decreases. The second observation is that the VN^3 filter step behaves independently from the density of the points of interest and their distribution in the network. For example, while Churches have a cardinality ratio of almost 24 times the Hospitals, the difference between the corresponding generated candidate sets is only 1.5% (for K=1000) to 11% (for K=3). This means that whether the points of interest are very dense or sparsely scattered in the network, the performance of the VN^3 filter step does not change. This is because the average number of adjacent generators specified in Property 4 is "independent" of the density of the points of interest, their distribution, and the underlying network.

We also performed KNN queries on the second data set using the approaches proposed in [11] and [4]. Figure 7 compares the minimum and maximum values of $\frac{SKS}{K}$ for VN^3 with those of [11] and [4], which are represented in the figure by "Seidl" and "Hjaltason", respectively. As shown in the figure, there is a significant

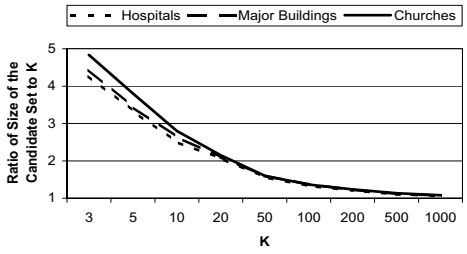
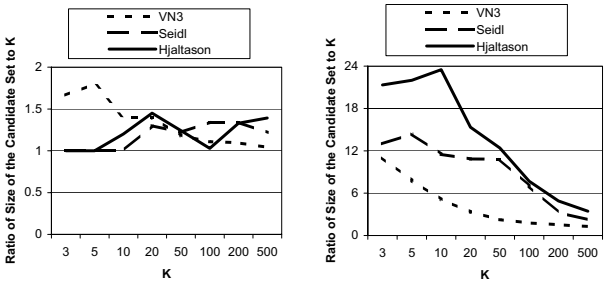


Figure 6: Performance of the filter step of VN^3



a. Minimum size of the candidate set

b. Maximum size of the candidate set

Figure 7: VN^3 filter step vs. Seidl & Hjaltason

difference between the minimum and maximum values of $\frac{SKS}{K}$, e.g., while the minimum sizes of the candidate sets generated by Seidl and Hjaltason are equal to K when K=5, their maximum sizes are 15 and 22 times larger than K, respectively. The average size of the candidate set generated by these approaches are up to 4 times larger than that of VN^3. We conclude that VN^3 filter step outperforms approaches optimized for index structures by providing better average values and a lower variance on the size of the candidate set, resulting in a more deterministic query response time.

6 Conclusion

In this paper we presented a novel approach for K nearest neighbor queries in spatial network databases. Our approach, VN^3, is based on precalculating the network Voronoi polygons (NVP) and pre-computing some network distances. We showed that since NVPs preserve network distances, they can immediately be used to find the first nearest neighbor of a query object. We also introduced new properties of NVPs that prove the nearest neighbors of the query object are within the adjacent NVPs of the previously explored nearest neighbors. Subsequently, we proposed alternative approaches that utilize the pre-computation component of VN^3 to compute the exact network distances from the query object to its potential nearest neighbors. We finally discussed how VN^3 can cope with database updates. The main features of VN^3 are as follow.

- VN^3 outperforms INE in query response time by a factor of 1.5 to 12 depending on the value of K and density of the points of interest.
- The VN3's filter step results in up to 4 times less number of candidates as compared to that of the traditional approaches. In addition, the size of VN3's candidate set has less variance across different locations of the query points and densities of the points of interest, resulting in more deterministic query response time.
- Although VN^3 is built on complex properties to prove its correctness, but it is a straightforward approach to implement by utilizing simple data structures such as R-tree and lookup tables.
- The pre-computation required by VN^3 has low computation and space complexities due to performing the pre-computations in local areas as opposed to across the entire network. This also results in reasonable update costs.

We plan to extend VN^3 to address similar KNN queries such as group and continuous KNN.

7 Acknowledgement

This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0238560 (CAREER), IIS-0324955 (ITR), unrestricted cash gift from the Microsoft Corporation, and in part by grant 03CRSA0631 from the US Geological Survey (USGS).

References

[1] S. Berchtold, B. Ertl, D. A. Keim, H.-P. Kriegel, and T. Seidl. "Fast Nearest Neighbor Search in High-Dimensional Space". In *ICDE 1998, Orlando, Florida, USA*.

[2] T. Chiueh. "Content-Based Image Indexing". In *VLDB 1994, Santiago de Chile, Chile*.

[3] P. Ciaccia, M. Patella, and P. Zezula. "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces". In *The VLDB Journal*, pages 426–435, 1997.

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

[5] C. S. Jensen, J. Kolarvr, T. B. Pedersen, and I. Timko. "Nearest Neighbor Queries in Road Networks". In *ACMGIS 2003, New Orleans, Louisiana, USA*.

[6] S. Jung and S. Pramanik. "An Efficient Path Computation Model for Hierarchically Structured Topological Road Maps". In *IEEE Transaction on Knowledge and Data Engineering*, 2002.

[7] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. "Fast Nearest Neighbor Search in Medical Image Databases". In *VLDB 1996, Mumbai (Bombay), India*.

[8] 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.

[9] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. "Query Processing in Spatial Network Databases". In *VLDB 2003, Berlin, Germany*.

[10] N. Roussopoulos, S. Kelley, and F. Vincent. "'Nearest Neighbor Queries". In *SIGMOD 1995, San Jose, California*.

[11] T. Seidl and H.-P. Kriegel. "Optimal Multi-Step k-Nearest Neighbor Search". In *SIGMOD 1998, Seattle, Washington, USA*.

[12] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh. "A Road Network Embedding Technique for k-Nearest Neighbor Search in Moving Object Databases". In *ACMGIS 2002, McLean, VA, USA*.