Indexing Network Voronoi Diagrams [⋆]

Ugur Demiryurek, and Cyrus Shahabi

University of Southern California
Department of Computer Science
Los Angeles, CA 90089-0781
[demiryur, shahabi]@usc.edu

Abstract. The Network Voronoi diagram and its variants have been extensively used in the context of numerous applications in road networks, particularly to efficiently evaluate various spatial proximity queries such as k nearest neighbor (kNN), reverse kNN, and closest pair. Although the existing approaches successfully utilize the network Voronoi diagram as a way to partition the space for their specific problems, there is little emphasis on how to efficiently find and access the network Voronoi cell *containing* a particular point or edge of the network. In this paper, we study the index structures on network Voronoi diagrams that enable exact and fast response to contain query in road networks. We show that existing index structures, treating a network Voronoi cell as a simple polygon, may yield inaccurate results due to the network topology, and fail to scale to large networks with numerous Voronoi generators. With our method, termed Voronoi-Quad-tree (or VQ-tree for short), we use Quad-tree to index network Voronoi diagrams to address both of these shortcomings. We demonstrate the efficiency of VQ-tree via experimental evaluations with real-world datasets consisting of a variety of large road networks with numerous data objects.

1 Introduction

The latest developments in wireless technologies as well as the widespread use of GPS-enabled mobile devices have led to the recent prevalence of location-based services. An important class of location based queries consists of proximity queries such as k Nearest Neighbor(kNN) query [15, 32, 21, 6, 7] and its variations, e.g., Reverse k Nearest Neighbor (RkNN) [23, 29], k Aggregate Nearest Neighbor (kANN) [28]. The proximity queries in general search for data objects that minimize a distance-based function with reference to one or more query objects.

With proximity queries, potentially the distance between the query point and every object in the database (e.g., all the points-of-interest) must be computed in order to find the closest (or the k closest) object(s) to the query point. Hence, the main research focus has been on indexing the objects to avoid the exhaustive search. Earlier studies assumed Euclidean distance as the distance function and hence indexed the objects in Euclidean space (e.g., [32, 30, 21, 24]) using R-tree [4] like index structures. With the advent of online mapping systems such as Google Maps and Mapquest and the availability of accurate

⋆ This research has been funded in part by NSF grants IIS-0238560 (PECASE), IIS-0534761,IIS-0742811 and CNS-0831505 (CyberTrust), and in part from the METRANS Transportation Center, under grants from USDOT and Caltrans.Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We thank Prof. Ulrich Neumann for his insightful discussions and comments.

nation-wide road network data, the proximity queries have been extended from Euclidean space to the road network space as natural artifact. The challenge in processing proximity queries on road networks is that the computation of the distance function is complex and hence the indexing techniques incorporated some sort of pre-computation of distances (in network) into their structures. One such approach is based on using network Voronoi diagrams [12].

A network Voronoi diagram is a specialization of a Voronoi diagram in which the locations of objects are restricted to the network edges and the distance between objects is defined as the length of the shortest network distance (e.g., shortest path or shortest time), instead of the Euclidean distance. Any network node located in a Voronoi cell has a shortest path to its corresponding Voronoi generator that is always shorter than that to any other Voronoi generator. A large number of studies adopted network Voronoi diagrams [12] to evaluate variety of proximity queries on road networks (e.g., [7, 11, 13, 27, 17]). For example, in [13] Okabe et al. introduced six different types of network Voronoi diagrams (each corresponds to very important real-world applications) whose generators are based on points, sets of points, lines and polygons, and whose distances are given by inward/outward distances, and additively/multiplicatively weighted shortest path distances.

Given a query point q and network Voronoi diagram (NVD), the first step in answering any proximity query is to locate the network Voronoi cell $NVC(p_i)$ that contains q (the generator p_i of $NVC(p_i)$ is the nearest neighbor of q). We refer to this operation as $contain(q)$ in the rest of the paper. Considering the large size of the underlying space (e.g., a continental size road network) with numerous data objects as well as the online nature of the queries that requires fast response-time, an index structure is necessary to efficiently access the portion of NVD associated with q. Although the existing approaches successfully used network Voronoi diagrams as a pre-computation approach for partitioning the network space, they overlooked the indexing techniques that enable efficient evaluation of $contain(q)$. Currently, indexing network Voronoi diagram with R-tree (referred as Voronoi R-tree or VR-tree for short) is the only known method for locating the network Voronoi cell that contains a particular point or edge of the network. VR-tree is first proposed in [7] and later used in many other approaches based on NVD (e.g., [11, 27, 17]).

In this paper, we show that VR-tree has two main problems. First, VR-tree may yield inaccurate results due to the way the Voronoi cells are formed in network space, i.e., although a NVD is generated based on the network distance metric, its Voronoi cells are created and indexed as regular polygons in Euclidean space. This inconsistency may result in a network edge belonging to a cell $NVC(p_i)$, to be classified as a member of the cell $NVC(p_j)$ because due to the network topology, the edge falls inside the polygon of $NVC(p_j)$ even though its network distance is closer to the generator of $NVC(p_i)$. For example, Figure 1 depicts the network Voronoi diagram of a hypothetical road network where each line style corresponds to network Voronoi cells of the generators p_1, p_2 and p_3. With VR-tree the network Voronoi cells are formed by connecting the border points (i.e., $\{b_1, b_2, ..., b_7\}$) [1] and bounded by straight line segments (i.e., bold lines in the Figure). As shown, the edges marked by *false-negative* are included in the Voronoi cell of p_1 $NVC(p_1)$, however the network distance from any point on the false-negative edges to p_3 is shorter than that to p_1.

Second, VR-tree is inefficient because of the non-disjoint partitioning of the space. Specifically, VR-tree splits the network space with hierarchically nested and largely overlapping minimum bounding rectangles (MBR) created around network Voronoi cells. The

[1] We discuss the network Voronoi diagram generation in Section 4.1

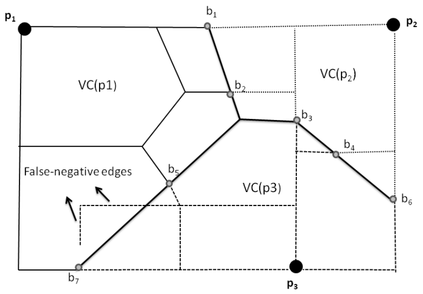


Fig. 1. Network Voronoi Diagram

overhead of executing *contain*(q) query is prohibitively high particularly in large networks with a dense (but perhaps large) set of data objects. This is because VR-tree has to redundantly visit the parent node(s) of the overlapping MBRs (aka, backtracking problem) in the index structure.

To address both of the aforementioned drawbacks, we propose a new indexing approach for network Voronoi diagrams based on region Quad-tree [18], termed *Voronoi-Quad-tree* or VQ-tree for short. VQ-tree, unlike VR-tree that approximates network Voronoi cells using regular polygons in the Euclidean space, enables exact representation of the network Voronoi cells based on quad-tree blocks in the network space, and hence always yields correct results. VQ-tree does not suffer from the backtracking problem of VR-tree. This is because VQ-tree enables disjoint decomposition of the network space and encodes each of the quad-tree blocks to indicate the identity of the network Voronoi cell of which it is a member. Thus, once the quad-tree block containing q is located, VQ-tree immediately identifies the nearest Voronoi generator based on the encoded value of that block. Our experiments with real-world datasets show that the ratio of false-negative edges is %16 on average with respect to the total number of edges in the network and VQ-tree outperforms VR-tree with 12 times improved response time (see Section 5).

The remainder of this paper is organized as follows. In Section 2, we review the related work about proximity queries in spatial networks. In Section 3, we overview Network Voronoi diagrams and it's properties. In Section 4, we establish the theoretical foundation of the proposed solution for indexing Network Voronoi diagrams for efficient and accurate processing of proximity queries in spatial networks. In Section 5, we present the results of our experiments with a variety of spatial networks with large number of query and data objects. Finally, in Section 6 we conclude and discuss our future work.

2 Related Work

The most widely studied class of proximity queries consists of k nearest-neighbor (kNN) and its variations. The research on kNN query processing can be categorized into two main areas, namely, Euclidean space and road networks. In the past, numerous algorithms (e.g., [32, 30, 21, 24, 9]) have been proposed to solve kNN problem in the Euclidean space. All of these approaches are applicable to the spaces where the distance between objects is only a function of their spatial attributes (e.g., Euclidean distance). In network spaces, however, the query and data objects are located in predefined network segments, where the distance between a pair of objects is defined as the length of the shortest path connecting them.

The challenge with processing kNN queries in road-network space is that the computation of the distance function (e.g., shortest path) is complex. Therefore, to enable efficient evaluation of kNN queries in road networks, the research in this area largely focused on techniques which utilize precomputed network distances and/or partial results. One common example of such techniques is the network Voronoi diagrams. Kolahdouzan and Shahabi proposed first network Voronoi based kNN search technique, termed VN3 [7, 8]. They retrieve the kNN of a query point q based on precomputed first-order network Voronoi diagram. Specifically, they first find the network Voronoi cell that contains q and then, to find k-1 nearest neighbors, search the adjacent Voronoi polygons iteratively. With their approach, they indexed the Voronoi cells with R-tree (i.e., VR-tree) to reduce the $contain(q)$ query to a point location problem in the Euclidean space. In [14], Papadias et al. introduced Incremental Network Expansion (INE) and Incremental Euclidean Restriction (IER) methods to support kNN queries in spatial networks. While INE is an adaption of the Dijkstra algorithm, IER exploits the Euclidean restriction principle in which the results are first computed in Euclidean space and then refined by using the network distance. Several other kNN algorithms are proposed based on the improved (precomputation) version of INE [1, 25, 5]. In [19], Samet et al. proposed *shortest path quadtree* algorithm for efficient evaluation of both shortest path and kNN queries in road networks. VQ-tree is mainly different than the shortest path quadtree for the following reason. With SPQ-tree, N region quad-trees are created, one for each vertex of a road network (with N vertices), where each quad-tree(SPQ-tree) represents the adjacency list of its corresponding vertex as regions. However, VQ-tree is a single quad-tree created for the entire road network with each of its encoded quad-blocks corresponding to one network Voronoi cell. In [13], Okabe et al. introduced a variety of network Voronoi diagrams where they assumed Voronoi generators as points, sets of points, lines and polygons, and network distances as inward/outward, and additively/multiplicatively weighted shortest path distances. Although they proposed very useful network Voronoi diagram based solutions to real-world road network problems, they did not focus on indexing techniques that efficiently find and access the network Voronoi cells in large scale road networks. In [11], Nutanong et al. proposed a technique called local network Voronoi diagram (LNVD) to continuously monitor kNN queries in road networks. With their approach, instead of creating NVD that covers the entire road network, they construct a network Voronoi diagram for a subspace around the query point. In different studies Zhao, Xuan, Taniar and Safar et al. utilized network Voronoi diagrams to evaluate different types of proximity queries including group kNN [16], mulitple kNN [31], reverse kNN [22], and range [26] queries in road networks. With all these studies, VR-tree is used to index the network Voronoi cells. However, as we mentioned VR-tree may return false results and inefficient in large networks with numerous data objects.

3 Background

In this section, we review the principles of Euclidean and Network Voronoi diagrams. We first introduce 2-dimensional Euclidean space Voronoi diagrams and describe the properties of Voronoi diagrams. We then explain the network Voronoi diagram. We refer readers to [12] for a comprehensive discussion of Euclidean and network Voronoi diagrams.

3.1 Voronoi Diagrams

Let $P : \{p_1, p_2, .., p_n\}$ be a set of n distinct sites (i.e., generator points) distributed in the Euclidean space. These generator points can be considered any spatial type of objects (e.g., gas station, restaurant). We define the *Voronoi diagram* of P as the subdivision of the space into n cells, one for each site in P, with the property that a point q lies in the cell corresponding to a site p_i if and only if $distance(q,p_i) < distance(q,p_j)$ for each $p_j \in P$ with $j \neq i$. Figure 2 shows the ordinary Voronoi diagram of eight points where the distance metric is Euclidean.

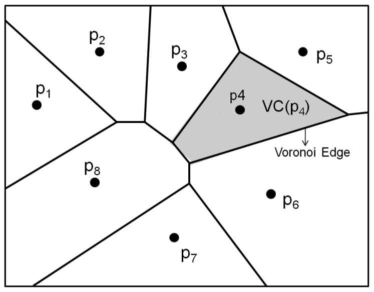


Fig. 2. Voronoi diagram in Euclidean space

We refer to the region containing the point p_i as its Voronoi cell $VC(p_i)$ or Voronoi polygon (see $VC(p_4)$ in the Figure). In Euclidean space, $VC(p_i)$ is a convex polygon. Each edge of $VC(p_i)$ is a segment of the perpendicular bisector of the line segment connecting p to another point of the set P. We call each of these edges a Voronoi edge. The Voronoi cells that have common edges are called *adjacent cells* and their generators are called *adjacent generators*. The Voronoi cells are collectively exhaustive and mutually exclusive except their boundaries (i.e., Voronoi edges). We define the Voronoi cell and Voronoi diagram as follows.

Definition 1. *Consider $P : \{p_1, p_2, .., p_n\}$ where $2 \leq n$ and $p_i \neq p_j$ for $i \neq j$, $i, j \in I_n = 1, ...n$. The region given by $VC(p_i) = p|d(p,p_i) \leq (p,p_j)$ where $d(p,p_i)$ is the minimum Euclidean distance between p and p_i is called the Voronoi Cell (VC) associated with p_i.*

Definition 2. *The set of Voronoi cells given by $VD(P) = \{VC(p_1), ..., VC(p_n)\}$ is called the Voronoi Diagram (VD) generated by P.*

3.2 Network Voronoi Diagrams

With network Voronoi diagrams (NVD), the VD described above is generalized by replacing the Euclidean space with a spatial network (e.g., road network), hence the distance with the network distance (e.g., shortest-path) between the objects.

Definition 3. *A road network is represented as a directional weighted graph $G(N,E)$, where N is a set of nodes representing intersections and terminal points, and E ($E \subseteq N \times N$) is a set of edges representing the network edges each connecting two nodes. Each edge e is denoted as $e(n_i, n_j)$ where n_i and n_j are starting and ending nodes, respectively.*

In this study, we consider planar graph where edges intersect only at their endpoints. We assume that Voronoi generators are located on the network segments as the graph nodes. Each edge connecting nodes p_i, p_j stores the network distance $d_N(p_i,p_j)$. For nodes that are not directly connected, $d_N(p_i,p_j)$ is the length of the shortest path from p_i to p_j.

Given a weighted graph $G(N,E)$ consisting of a set of nodes $N = \{p_1, ...p_n, p_{n+1}, ..p_o\}$ where the first n nodes represent the Voronoi generators and a set of edges $E = \{e_1, ...e_k\}$ that connects the nodes, we define the set *dominance region* and *border points* as follows,

Definition 4. *The dominance region of p_i over p_j*

$Dom(p_i, p_j) = \{p | p \in \bigsqcup_{o=1}^{k} e_o, d_N(p, p_i) \leq d_N(p, p_j)\}$ *represents all points in all edges in E that are closer (or equal distance) to p_i than p_j.*

Definition 5. *The border points between p_i and p_j $b(p_i, p_j) = \{p | p \in \bigsqcup_{o=1}^{k} e_o, d_N(p, p_i) = d_N(p, p_j)\}$ represent all points in all edges that are equally distanced from p_i and p_j.*

Definition 6. *Based on the above definitions, the Voronoi edge set V_{edge} of p_i as $V_{edge}(p_i) = \bigsqcup_{j \in I_n \backslash \{i\}} Dom(p_i, p_j)$ represents all the points in all edges in E that are closer to p_i than any other generator point in N. Consequently, we define network Voronoi diagram $NVD(P)$ w.r.t set of points P as $NVD(P) = \{V_{edge}(p_1), ..., V_{edge}(p_n)\}$.*

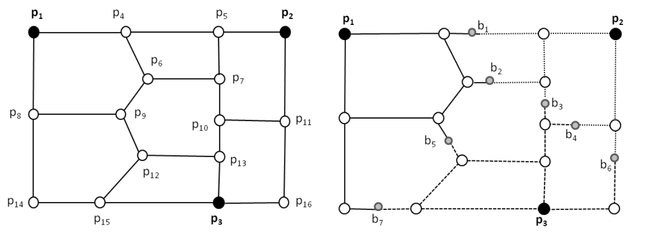
Similar to VD described in Section 3.1, the elements of NVD are mutually exclusive and collectively exhaustive.

4 Indexing Network Voronoi Diagrams

In this section, we will first explain how to construct a network Voronoi diagram in road networks and then discuss two different index structures, namely the Voronoi R-tree and Voronoi Quad-tree that efficiently identifies the subdivision of the network space that contains a particular query point or network edge.

4.1 Network Voronoi Diagram Construction

The network Voronoi diagrams can be constructed using parallel Dijkstra algorithm [2] with the Voronoi generators as multiple sources. Specifically, one can expand shortest path trees from each Voronoi generator simultaneously and stop the expansions when the shortest path trees meet.



(a) Road Network (b) Network Voronoi Diagram

Fig. 3. A Road network and network Voronoi diagram

Figure 3 shows an example of road network and the corresponding network Voronoi diagram. Figure 3a depicts the original weighted graph $G(N, E)$ which consists of $N = \{p_1, p_2, p_3, p_4, ...p_{16}\}$ nodes where p_1, p_2, and p_3 are the Voronoi generators (i.e., data objects such as restaurants, hotels) and p_4 to p_{16} are the intersections on a road network that are interconnected by a set of edges. Figure 3b shows the NVD of the road network where each line style corresponds to the shortest path tree based on the generators $\{p_1, p_2, p_3\}$. Each shortest path tree composes a network Voronoi cell and some edges (e.g., $e(p_4, p_5)$) can be partially contained in different network Voronoi cells. The border points b_1 to b_7 are the nodes where the shortest path trees meet as a result of the parallel Dijkstra algorithm. The border points between any two generator p_i and p_j are equally distanced from p_i and p_j. Figure 4 shows a real network Voronoi diagram with respect to 50 data objects in Los Angeles road network. Each network node marked with a different color corresponds to a network Voronoi cell.

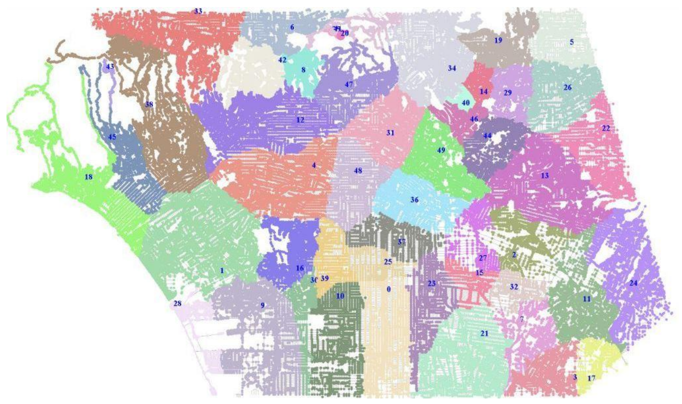


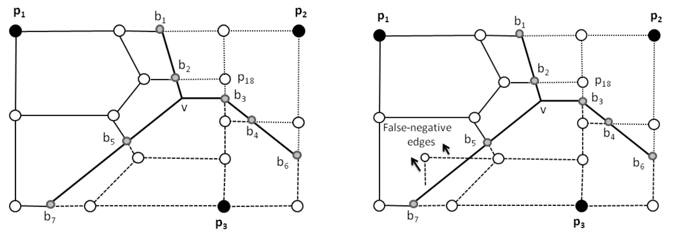
Fig. 4. Network Voronoi diagram with $P = \{p_1, ..., p_{50}\}$ in Los Angeles road network.

4.2 Index Generation on Network Voronoi Diagram

As we discussed, to answer any proximity query with respect to a query point q, one first needs to find the Voronoi cell that contains q. There remains a basic question concerning how to efficiently access the portion of the NVD associated with a particular query point q. This can be achieved by utilizing a spatial index structure that is generated on Voronoi cells. Below, we discuss two types of spatial index structures that can be used to index NVCs, namely, the Voronoi R-tree(VR-tree) and Voronoi Quad-tree (VQ-tree).

4.2.1 The Voronoi R-tree (VR-tree) VR-tree is first introduced in [7] where NVD is used to evaluate kNN queries in road networks. VR-tree is based on the R-tree [4] that splits the network space with hierarchically nested Minimum Bound Rectangels (MBR) generated around network Voronoi cells. Given the location of a query point q, a $contain(q)$ query invoked on VR-tree starts from the root node and iteratively checks the MBRs (of NVCs) with respect to a q to decide whether or not to further search the child nodes.

VR-tree has two main shortcomings. First, VR-tree may yield inaccurate results for a $contain(q)$ query. This is because VR-tree makes the simplifying assumption that although the NVD is computed based on the network distance metric, its NVCs are treated as regular polygons (by connecting border points of NVCs) and indexed using R-tree that is designed for the Euclidean distance metric. However, such approach may cause misclassification of the network edges (i.e., false-negative edges) in the network Voronoi cells, and hence inaccurate results. Specifically, a network edge belonging to a network Voronoi cell of p_i $NVC(p_i)$ may be classified as a member of another network Voronoi cell $NVC(p_j)$. For



(a) NVC in VR-tree (b) False-negative edges

Fig. 5. Network Voronoi cell construction in VR-tree

instance, continuing with our running example in Figure 3, Figure 5(a) shows how adjacent border points are 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. 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). As a result, similar to its Euclidean counterpart, the NVCs are represented with polygons in the network space. However, to illustrate why VR-tree may fail to yield correct results, consider Figure 5(b) where we introduce two new edges (as an extension of p_{12}) to the road network. As shown, although the new edges (marked by false-negative edges in the Figure) are included inside the Voronoi cell of p_1, the network distance from any point on the false-negative edges to p_3 is shorter than that to p_1. Thus, with VR-tree, when q is located on false-negative edges, a *contain*(q) will return incorrect Voronoi generator as the NN. With our example we only show one particular case that can happen in real-world road networks. Arguably, it is possible to increase the number of such examples under different road network topologies. Figure 6 depicts the NVC of a particular data object in Los Angeles road network where border nodes and false-negative edges are marked by light blue and red color, respectively.

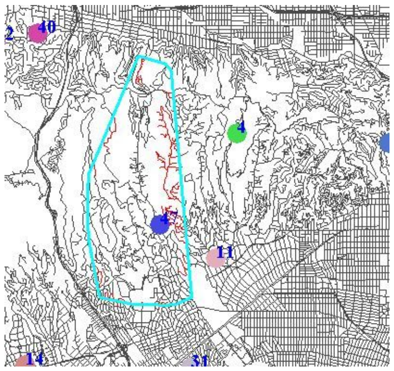


Fig. 6. False-negative edges of a NVC in Los Angeles road network

One naive solution to the inaccuracy problem of VR-tree is to perform an additional refinement step. Specifically, one can maintain false-negative edges (along with their corresponding Voronoi generators) in a separate index structure and, for each *contain*(q) query, check q against this index structure. If q is located in any of the false-negative edges, the corresponding Voronoi generator is returned as the nearest neighbor. Otherwise, VR-tree continues the search based on MBRs of the Voronoi cells as explained above.

Second, VR-tree is inefficient due to non-disjoint partitioning of the space. Specifically, with VR-tree the hierarchy of NVCs is enforced by minimum bounding rectangles created around network Voronoi cells. Depending on the different topologies of the road network and the distribution of the objects on the network segments, the overlapping areas of MBRs of network Voronoi cells may be quite large, and hence significant computation overhead in traversing R-tree for *contain*(q) query. For example, Figure 7 illustrates the MBRs of network Voronoi cells in Figure 4. For the sake of clarity, we do not include the Voronoi cells in the picture. As shown, the MBRs around network Voronoi cells result in a *non-disjoint decomposition* of the underlying space which means that the location occupied by a Voronoi cell may be contained in several bounding boxes. This degrades the search performance in VR-tree because of the backtracking [4] problem, i.e., the parent node(s) of the overlapping MBRs have to be accessed repeatedly in order to search the child nodes that contain q. Thus, with VR-tree the amount of work often depends on the overlapping areas of MBRs. We also implemented VR-tree with R+ tree [20] to reduce the impact of overlapping areas. However, we observe that the performance of VR+ tree is still less as compared to VQ-tree (see Section 5.2.4).

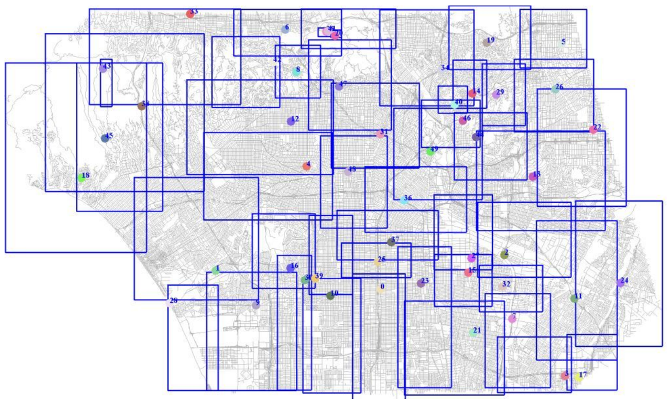


Fig. 7. Minimum bounding rectangles on network Voronoi cells

4.2.2 The Voronoi Quad-tree (VQ-tree) The alternative to VR-tree is to index network Voronoi cells using Quad-tree [18, 3], termed Voronoi Quad-tree (VQ-tree), that enables disjoint decomposition of the underlying space. The main observation behind VQ-tree is that each color coded area in Figure 4 is a spatially contiguous region in the network space. The regions are mutually exclusive as they do not have any overlapping areas and collectively exhaustive as every location in the network space is associated with at least one generator. Therefore, an *exact approximation* of the network Voronoi diagram can be obtained by using a *region quad-tree* [18] where the leaf nodes of the quad-tree correspond to a region in a Voronoi cell in NVD. In particular, with VQ-tree the root node represents the rectangular region enclosing the entire span of the road network (and hence NVD) under consideration. We subdivide this rectangular region into four equal quadrants where each quadrant is one of the four child nodes of the root. Subsequently, we recursively subdivide the quadrants until each quadrant contains only one network Voronoi cell information.

That is, for each quadrant, we search for two (or more) different color-coded nodes [2]. If we find such a quadrant (meaning that the quadrant includes more than one network Voronoi cell), we subdivide that quadrant into four subquadrants. This subdivision process continues recursively until all nodes in a quadrant have the same color code.

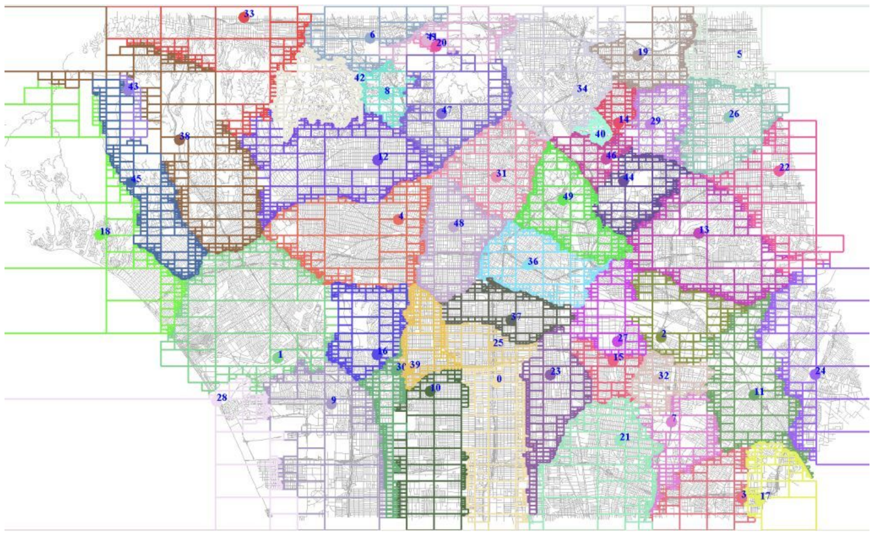


Fig. 8. VQ-tree on Los Angeles road network

Figure 8 illustrates the quad-blocks generated on the road network in Figure 4. We note that the leaf nodes of VQ-tree does not store any information about the network nodes. As shown in Figure 9, the leaf nodes only store the region information (i.e., coordinates) of the quad-blocks as well as a single value (e.g, a color code or a integer number) which indicates the identity of the network Voronoi cell of which the quad-tree block is a member. We note that a leaf node in the quad-tree corresponds to a particular subdivision of a network Voronoi cell.

As shown in 8, each network Voronoi cell NVC_i consists of disjoint quad-tree blocks. The disjoint decomposition of the network Voronoi diagram with VQ-tree addresses the two drawbacks of VR-tree. Specifically, unlike VR-tree that roughly estimates the network Voronoi cells with polygons in the Euclidean space, VQ-tree enables the exact representation of the network Voronoi cells using quad-tree blocks and hence always yield correct results. VQ-tree does not suffer from the backtracking problem of VR-tree, and hence fast response time for $contain(q)$. This is due to non-overlapping partitioning of the network Voronoi cells: once the quad-tree block containing q is located in the leaf nodes, VQ-tree immediately identifies the nearest Voronoi generator based on the value (e.g, a color code) of that block.

Algorithm 1 presents the outline for VQ-tree. Given a set of N nodes with their color codes and bounding box $[x_1; x_2]$x$[y_1; y_2]$ that contains N as an input, Algorithm 1 creates VQ-tree by recursively splitting the quadrants until all the nodes in a quadrant have the same color code.

[2] During NVD construction parallel Dijkstra algorithm can encode each node with a Voronoi cell identifier, e.g., a color

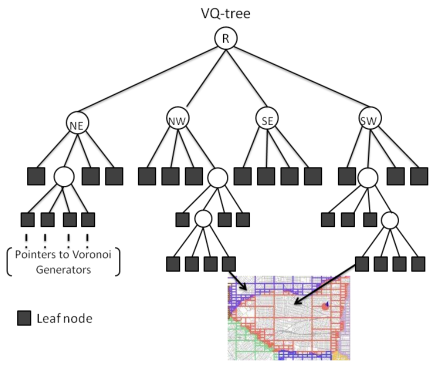


Fig. 9. VQ-tree

Algorithm 1 VQ-Tree Algorithm

```
$VQuadTree(N, x_1, x_2, y_1, y_2)${
/* Scan distinct color codes in the region
$cellColor[] \Leftarrow checkRegion(N, x_1, x_2, y_1, y_2)$;
/* If there exist more than one color-code then split
**if** $cellColor.length > 1$ **then**
    /*Initialize intermediate node
    $node \Leftarrow QuadTreeNode()$;
    /*Set Quadrants
    $node.SE \Leftarrow VQuadTree(N, x_1, (x_2{+}x_1)/2, y_1, (y_1{+}y_2)/2)$;
    $node.SW \Leftarrow VQuadTree(N, (x_2{+}x1)/2, x_2, y_1, (y_1{+}y_2)/2)$;
    $node.NE \Leftarrow VQuadTree(N, x_1, (x_2{+}x_1)/2, (y_1{+}y_2)/2, y_2)$;
    $node.NW \Leftarrow VQuadTree(N, (x_2{+}x_1)/2, x_2, (y_1{+}y_2)/2, y_2)$;
**else**
    /*Create leaf node
    $QuadTreeLeafNode(cellColor[0])$;
**end if**
}
```

5 Experimental Evaluation

5.1 Experimental Setup

We conducted experiments with different spatial networks and various parameters to evaluate the performance of VQ-tree and VR-tree. We measured the ratio of false-negative edges with varying object cardinality (i.e., number of Voronoi generators) and object distribution in the road network. In addition, we compared the precomputation, index rebuilding (for dynamic environments) and response time of VQ-tree and VR-tree with respect to different network sizes and object cardinality. As of our dataset, we used California (CA), Los Angeles (LA) and San Joaquin County (SJ) road network data (obtained from Navteq [10]) with approximately 1,965,300, 304,162 and 24,123 nodes, respectively. Since the experimental results with LA and SJ networks differ insignificantly, we only present the results from the CA and LA datasets. We conducted our experiments on a workstation with 2.7 GHz Pentium Core Duo processor and 12GB RAM memory. For each set of experiments, we only vary one parameter and fix the remaining to the default values in Table 1.

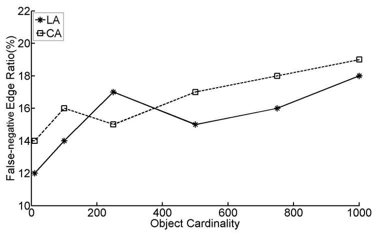
Table 1. Experimental parameters

Parameters	Default	Range
Object Cardinality	100	10,50,100,500,1000
Road Network	LA	SJ, LA, CA
Object Distribution	Uniform	Uniform, Gaussian

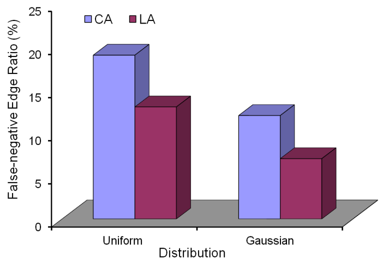
5.2 Results

5.2.1 Ratio of False-negative Edges First, we study the ratio of false-negative edges with respect to object cardinality (i.e., number of Voronoi generators) and object distribution. To identify false-negative edges, we compare the encoded values (i.e., color code) of each node based on VR-tree and VQ-tree. Specifically, we first encode each edge to its corresponding Voronoi generator by using VR-tree polygons and then compare the encoded values to that we obtained from VQ-tree. We repeat each experiment 100 times and report the average number of incorrectly encoded (i.e., false-negative) edges with respect to total number of edges in the network. Figure 10(a) shows the ratio of false-negative edges of both networks where the object cardinality ranging from 10 to 1000. As illustrated, the ratio of incorrectly identified edges is %16 on average in both networks. The maximum recorded false-negative edge ratio for LA and CA road networks is %24 and %29, respectively.

Figure 10(b) illustrates the ratio of false-negative edges with different object distribution for both CA and LA road networks. We observe that the number of false-negative edges is less in Gaussian distribution. This is because as objects are clustered in the spatial network with Gaussian distribution, the corresponding shortest path trees would be less disperse and hence spatially close border points. As mentioned, with VR-tree we encode the edges based on the Euclidean polygon generated by connecting the border points. The more spatially close border points provides the more accurate presentation of the NBCs and hence less false-negative edges.



(a) Impact of object cardinality



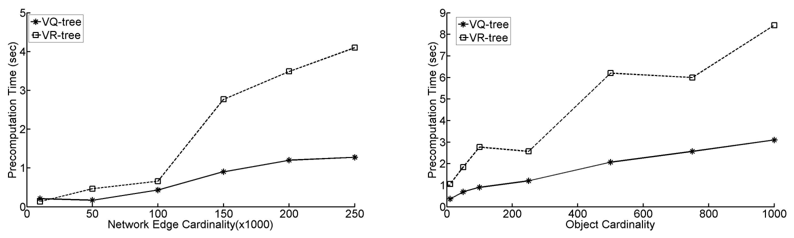
(b) Impact of object distribution

Fig. 10. Impact of object cardinality and distribution

5.2.2 Precomputation Time With another set of experiments, we compare the precomputation (i.e., index construction) time of VR-tree and VQ-tree with varying network sizes and number of objects. In order to evaluate the impact of network size, we conducted experiments with the sub-networks of CA dataset ranging from 50K to 250K segments.

We set the the node size of VR-tree to 4K bytes in all cases. Figure 11(a) shows the precomputation time of VQ-tree and VR-tree in CA road network with varying network size. The results indicate that the precomputation time increases with the network size in both methods where VQ-tree outperforms VR-tree with all numbers of edges. This is because as the network size increases the perimeters of the polygons (and hence the number of connected line segments that form a polygon) grow in VR-tree. Arguably, the overhead of generating MBRs (to be used in VR-tree) around the polygons composed of numerous connected line strings is time-consuming as the coordinates (that form the lines) needs to be scanned to find the ultimate corners of the MBR. On the other hand, VQ-tree is constructed based the underlying space (rather than objects in VR-tree) by recursively dividing the road network to quad-blocks each corresponding to one NVC.

Figure 11(b) illustrates the impact of object cardinality over precomputation time in LA road network (the results are similar in CA network and hence not presented). We observe that as the number of objects in the road network increases, the preprocessing time for both approaches increases. As shown, the precomputation time for VQ-tree outperforms VR-tree. The reason is that the time for hierarchically clustering polygons in VR-tree for a large datasets is relatively expensive. We also observe that the depth of VQ-tree increases with the increasing number of data objects. This is because large number of data objects yields smaller VCs and hence more splits.

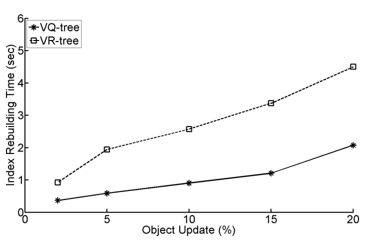


(a) Impact of network size (b) Impact of object cardinality

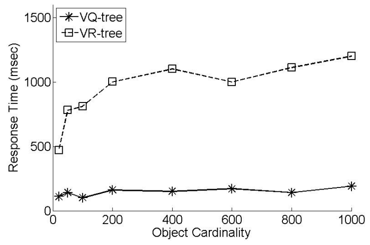
Fig. 11. Impact of network size

5.2.3 Index Reconstruction Next, we compare the index reconstruction overhead of VR-tree and VQ-tree with respect to object updates. In this set of experiments, we update the location of the randomly selected data objects and measure the index reconstruction overhead in both VR-tree and VQ-tree. Figure 12(a) shows the index reconstruction time of both index structures with varying object update ratio (i.e., the percentage of data objects whose locations changed). We observe that VQ-tree outperforms VR-tree with respect to index reconstruction. This is because the insert operations in VR-tree are expensive. When new data objects are inserted into VR-tree, besides updating leaf nodes, it is likely that updates are also required to non-leaf nodes (i.e., more than one branch of the tree maybe expanded), which leads to a large overhead during insertion. On the other hand, with VQ-tree we observe that most of the index updates take place in the leaf nodes.

5.2.4 Response Time In this experiment, we compare the performance (i.e., the response time for *contain*(q) query) of VQ-tree and VR-tree with varying object cardinality. We determine the location of the query object q uniformly at random and report average



(a) Index reconstruction



(b) Impact of object cardinality

Fig. 12. Response time vs object cardinality and Index reconstruction

of 100 queries. As we mentioned the original VR-tree proposed in [7] may yield inaccurate results. In order to provide correct results with VR-tree, we modify VR-tree by adding an additional index structure that maintains false-negative edges. Specifically, we construct a R-tree on the false-negative network edges along with their Voronoi generators. With each *contain*(q) query, we check q against this index structure. If we locate q on any of the false-negative edges, the corresponding data object is returned as the first NN. Otherwise, VR-tree continues the search based on the polygons explained in 4.2.1. Figure 12(b) plots the average response time for *contain*(q) query. The results indicate that VQ-tree outperforms VR-tree and scales better with large number of data objects. The response time of VQ-tree is approximately 12 times better than that of VR-tree with more than 200 data objects. This is because of the fact that, with VR-tree, the amount of work often depends on the size of the overlapping areas. In particular, the overlapping areas may belong to more than one NVC and hence during the search the parent node(s) of the overlapping MBRs have to be accessed repeatedly. We also implemented VR-tree using R+ tree (VR+) that minimizes the impact of overlapping areas. We observe that the performance of VQ-tree is still 7 times superior to VR+ tree.

6 Conclusion

In this paper, we study two different spatial index structures, namely the Voronoi R-tree and Voronoi Quad-tree, to index network Voronoi diagrams. These index structures enable efficient access to the network Voronoi cells containing a particular point or edge of the network. We show that previously proposed Voronoi R-tree may yield inaccurate results and fail to scale in large road networks with numerous data objects. We propose a novel approach, termed Voronoi Quad-tree, that enables disjoint decomposition of the network Voronoi diagram where network Voronoi cells are indexed with region quad-tree. The precomputation overhead of the Voronoi Quad-tree is significantly less and the Voronoi Quad-tree outperforms Voronoi R-tree in query response time by a factor of 1:4 to 12 depending on the network size and object cardinality. We intend to pursue this study in two directions. First, we plan to investigate disk organization strategies for Voronoi Quad-tree. Second, we intend to work on incremental index update techniques to avoid node reconstruction overhead due to update in the location of Voronoi generators.

References

1. H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. In *VLDB*, 2005.

2. M. Erwig and F. Hagen. The graph voronoi diagram with applications. *Journal of Networks*, 36, 2000.
3. R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 1974.
4. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.
5. H. Hu, D. Lee, and J. Xu. Fast nearest neighbor search on road networks. In *EDBT*, 2006.
6. C. S. Jensen, J. Kolářvr, T. B. Pedersen, and I. Timko. Nearest neighbor queries in road networks. In *GIS*, 2003.
7. M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, 2004.
8. M. R. Kolahdouzan and C. Shahabi. Continuous k-nearest neighbor queries in spatial network databases. In *STDBM*, 2004.
9. M. F. Mokbel, X. Xiong, and W. G. Aref. Sina: scalable incremental processing of continuous queries in spatio-temporal databases. In *SIGMOD*, 2004.
10. NAVTEQ. www.navteq.com. accessed in may 2011.
11. S. Nutanong, E. Tanin, M. E. Ali, and L. Kulik. Local network voronoi diagrams. In *SIGSPATIAL*, 2010.
12. A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. Spatial tessellations — concepts and applications of voronoi diagrams. 2000.
13. A. Okabe, T. Satoh, T. Furuta, A. Suzuki, and K. Okano. Generalized network voronoi diagrams: Concepts, computational methods, and applications. *Int. J. Geogr. Inf. Sci.*, 2008.
14. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, 2003.
15. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, 1995.
16. M. Safar. Group -nearest neighbors queries in spatial network databases. *Journal of Geographical Systems*, 2008.
17. M. Safar, D. Ibrahimi, and D. Taniar. Voronoi-based reverse nearest neighbor query processing on spatial networks. *Multimedia Systems*, 2009.
18. H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, CA, USA, 2006.
19. H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, 2008.
20. T. K. Sellis, N. Roussopoulos, and C. Faloutsos. R+-tree: A dynamic index for multi-dimensional objects. In *VLDB*, 1987.
21. Z. Song and N. Roussopoulos. K-nn search for moving query point. In *SSTD*, 2001.
22. D. Taniar, M. Safar, Q. T. Tran, J. W. Rahayu, and J. H. Park. Spatial network rnn queries in gis. *Comput. J.*, 2011.
23. Y. Tao, D. Papadias, and X. Lian. Reverse knn search in arbitrary dimensionality. In *VLDB*, 2004.
24. Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, 2002.
25. H. X., J. C.S., and S. Saltenis. The island approach to nearest neighbor querying in spatial networks. In *SSTD*, 2005.
26. K. Xuan, G. Zhao, D. Taniar, J. W. Rahayu, M. Safar, and B. Srinivasan. Voronoi-based range and continuous range query processing in mobile databases. *J. Comput. Syst. Sci.*, 2011.
27. K. Xuan, G. Zhao, D. Taniar, B. Srinivasan, M. Safar, and M. Gavrilova. Network voronoi diagram based range search. *Advanced Information Networking and Applications.*
28. M. L. Yiu, N. Mamoulis, and D. Papadias. Aggregate nearest neighbor queries in road networks. *ICDE*, 2005.
29. M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao. Reverse nearest neighbors in large graphs. *ICDE*, 2005.
30. J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based spatial queries. In *SIGMOD*, 2003.
31. G. Zhao, K. Xuan, D. Taniar, M. Safar, M. L. Gavrilova, and B. Srinivasan. Multiple object types knn search using network voronoi diagram. In *ICCSA (2)'09*, pages 819–834, 2009.
32. B. Zheng and D. L. Lee. Semantic caching in location-dependent query processing. In *SSTD*, 2001.