

Fast Object Search on Road Networks

Ken C. K. Lee
The Penn State University
University Park, USA
cklee@cse.psu.edu

Wang-Chien Lee
The Penn State University
University Park, USA
wlee@cse.psu.edu

Baihua Zheng
Singapore Management
University, Singapore
bhzheng@smu.edu.sg

ABSTRACT

In this paper, we present *ROAD*, a general framework to evaluate Location-Dependent Spatial Queries (LDSQs) that searches for spatial objects on road networks. By exploiting *search space pruning* technique and providing a dynamic object mapping mechanism, *ROAD* is very efficient and flexible for various types of queries, namely, range search and nearest neighbor search, on objects over large-scale networks. *ROAD* is named after its two components, namely, *Route Overlay* and *Association Directory*, designed to address the network traversal and object access aspects of the framework. In *ROAD*, a large road network is organized as a hierarchy of interconnected regional sub-networks (called *Rnets*) augmented with 1) *shortcuts* for accelerating network traversals; and 2) *object abstracts* for guiding traversals. In this paper, we present (i) the *Rnet* hierarchy and several properties useful to construct *Rnet* hierarchy, (ii) the design and implementation of the *ROAD* framework, (iii) efficient object search algorithms for various queries, and (iv) incremental update techniques for framework maintenance in presence of object and network changes. We conducted extensive experiments with real road networks to evaluate *ROAD*. The experiment result shows the superiority of *ROAD* over the state-of-the-art approaches.

General Terms

Spatial Search, Road Network, Index and Algorithm

1. INTRODUCTION

The proliferation of mobile devices, along with broad deployment of wireless communication networks and high precision geo-positioning technology, have been stimulating the growth of location-based services (LBSs) during the past decade. Typically, an LBS server maintains location-related information to answer user queries with respect to user-specified locations. We refer to the location-related information and user queries as *spatial objects* (or *object*, for

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the ACM. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM. *EDBT 2009*, March 24–26, 2009, Saint Petersburg, Russia. Copyright 2009 ACM 978-1-60558-422-5/09/0003 ...\$5.00

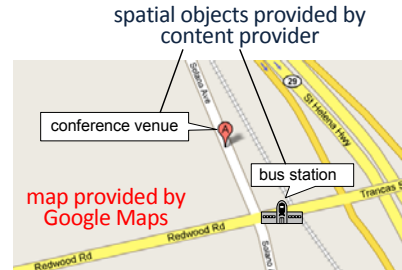


Figure 1: Spatial objects tagged on digital map

short) and *location-dependent spatial queries* (LDSQs), respectively. To many location-related applications, LDSQs often serve as fundamental data access operations. For example, for a conference event, LDSQs can help an attendee in travel planning, e.g., Q1: find the nearest bus station to the conference venue, and Q2: find hotels within 10-minute walk from the conference venue.

As objects and user trajectories are constrained by road networks, search of spatial objects should be based on *network distances*. Recently, a trend for LBS deployment has been growing quickly on the Web. Spatial objects from content providers (e.g., stores, average users etc) and digital maps from map service providers (e.g., Google Map, MapQuest, MS Virtual Earth, Yahoo! Map¹ etc.) are coupled to quickly deploy LBSs on the fly. As such, content providers embed maps from any map service provider on their web pages while tagging objects on the embedded maps. Figure 1 shows an example map from Google Map on which a bus station and a conference site (as spatial objects) are tagged by the conference webmaster (i.e., a content provider). In this model, content providers and map service providers do not necessarily maintain data from each other. This technological trend allows dynamic combination of contents and map services to facilitate content-rich LBSs.

Although existing applications based on this model can display objects tagged on a map and point-to-point directions search, common LDSQs like finding the nearest bookstores from a given location based on network distance or current traffic condition have not yet been supported. To meet the enormous Web and mobile user needs for LBSs, the support for efficient LDSQ processing is needed. Thus, there is a great demand on a system framework that can i) flexibly and efficiently accommodate diverse objects (in terms of contents, types, and formats) on maps, ii) efficiently support various LDSQs, and iii) effectively support different

¹<http://maps.google.com>, <http://www.mapquest.com>, <http://maps.live.com>, <http://map.yahoo.com>, respectively.

distance metrics such as road network distance, travel time, toll, etc to be considered for LDSQs.

However, all existing techniques proposed for processing LDSQs in spatial network databases, including *network expansion based*, *Euclidean distance bound based*, and *solution based approaches* [2, 6, 9, 13, 16, 19], do not provide the desired features. The network expansion based approaches, though supporting various types of queries, objects and distance metrics, are not efficient due to expensive network traversal involved in network distance computation. The Euclidean distance bound based approaches (which rely on heuristics derived from the physical properties of Euclidean distance) are not always applicable since Euclidean distance cannot be used to estimate some distance metrics (e.g., trip time, travel cost etc). Solution based approaches that pre-compute search results for specific queries to boost the search efficiency suffer from expensive costs of result pre-computation and storage. Besides, they adapt poorly to other query types, and to object and network changes.

In this paper, we propose a novel and efficient system framework, called *ROAD* for processing LDSQs on road networks. As we analyzed, there are two basic operations, namely, *network traversal* and *object lookup* involved in processing LDSQs on a road network. Network traversal visits network nodes and edges following a certain traversal strategy to determine the network distances of objects i.e., one of search criteria, while object lookup accesses and checks objects at traversed nodes or edges based on object attributes and search criteria. Objects collected during the course of a traversal are the answer objects. For a search that covers a large portion of a network, the overhead incurred by traversals and objects lookups would significantly deteriorate the overall search performance and thus need to be optimized.

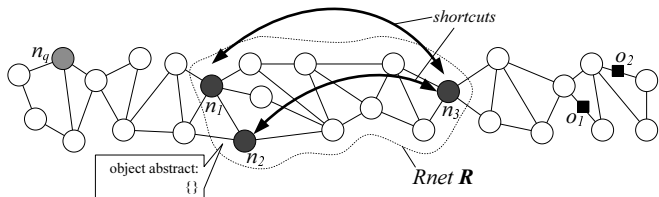


Figure 2: Basic idea behind ROAD framework

Figure 2 illustrates the problems and provides an overview of our basic ideas. As shown, o_1 and o_2 are two objects on a network. If an NN query is issued far away from these two objects, say at n_q , the search cost is expected to be higher than the same query issued somewhere close to the objects. As traversals from a query point towards the searched objects and the placements of objects are constrained by the network topology, nodes and edges (i.e., the entire network) logically form an object search space. Observing that some subspaces (i.e., small portions of the network) with no objects can be skipped from detailed examination during a search, we strategically formulate a network as a collection of regional subnets (called *Rnets*). As such, each of the Rnets captures a search subspace. The idea, aiming at exploiting *search space pruning* effective technique to effectively speed up the search performance, is to avoid detailed traversal and object lookup within Rnets and thus allow the bypass of those Rnets that do not contain objects of interest. To enable the bypass of an Rnet during traversal, two pieces of additional information are required: (i) information

about selective (i.e., shortest) paths across an Rnet that allow traversals to continue at other sides of the Rnet, and (ii) information about the content of objects inside an Rnet to provide a quick search guideline. These two requirements lead to the notions of *shortcuts* and *object abstract*, respectively, in the paper. As shown in the figure, with the aid of shortcuts and the object abstract for Rnet R , a search from n_q can bypass R and continue on the other side of the network to find objects, as R is found containing no object.

To realize the ROAD framework, two novel index structures, namely, *Route Overlay* and *Association Directory*, have been proposed. The ROAD framework is named after these two key components. The former naturally manages the physical network structure and the shortcuts, while the latter associates objects and object abstracts with the road network. This design offers many advantages. First, it provides a clean separation between network and objects. In practice, map service providers may provide shortcuts for Rnets, while content providers may map objects to the nodes, edges and Rnets on the fly. Meanwhile, flexible object and network updates can be facilitated. Additionally, diverse object types can be supported upon the same network. Second, shortcuts can be customized for different distance metrics as needed by applications. Third, as both network traversal and object lookup are seamlessly supported by ROAD, various LDSQs can be efficiently processed.

In this paper, we detail the design, implementation and evaluation of ROAD and provide a holistic solution to several important research issues, including organization of Rnets, search algorithms for LDSQs, and maintenance of the ROAD framework. Notice that the concepts of Rnets, shortcuts, object abstracts introduced for road networks in this paper are also applicable to other spatial networks like airline networks. In summary, this paper presents the following significant contributions:

1. We present *ROAD*, a system framework to support efficient processing of LDSQs on road networks. It clearly separates network and objects, exploits search space pruning technique, and supports object search based on different distance metrics.
2. We develop Rnet hierarchy and employ its nice properties to reduce index overhead, improve query performance, and facilitate incremental framework maintenance.
3. We devise efficient search algorithms for range queries and nearest neighbor queries, i.e., two of the most common types of LDSQs, upon ROAD framework. Our algorithms significantly reduce the traversal overheads, thereby rendering fast search performance.
4. We develop efficient ROAD maintenance schemes to handle object and network changes.
5. We conduct an extensive performance evaluation on ROAD. The result shows the superiority of ROAD over the state-of-the-art approaches.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 presents the core concept behind ROAD and details the ROAD framework, such as the properties of shortcuts and object abstracts, the organization of Rnets and ROAD implementation. Section 4 and Section 5 detail the query processing algorithms and ROAD framework maintenances, respectively. Section 6 reports the performance evaluations of ROAD in comparison with existing works. Section 7 concludes this paper.

2. RELATED WORK

In this section, we first discuss existing works that can be categorized into *network expansion based approaches*, *Euclidean distance bound approaches* and *solution based approaches*, on processing LDSQs on road networks. Then, we review related works on hierarchical road networks that aim at facilitating shortest path search.

Network expansion based approaches. Network expansion gradually expands the search space in a network by forming a spanning tree rooted at a given query point. It is applicable for object search and shortest path search like Dijkstra’s algorithm [4]. Iteratively, it examines the next closest unexplored node that guarantees the expansion to be minimal each time until all the nodes and edges that satisfy search criteria are visited [9, 16]. Objects of interest located on the visited nodes and edges are the result objects and the paths from the root to those objects are the shortest paths. Although the network expansion is useful for many LDSQs, it is inefficient due to an almost blind scan over the entire search space and a slow *node-by-node* expansion towards *all* directions. For a large search space, this deficiency seriously deteriorates the search performance.

Euclidean distance bound approaches. Euclidean distance is always the lower bound of physical path distance. Euclidean distance bound approaches [16, 19] employ this property as a heuristic to identify candidate objects whose Euclidean distances are not greater than a certain threshold distance. Then, false candidates whose network distances that can be determined by shortest path algorithms (e.g. A* algorithm [3]) or materialized distances (e.g. HEPV [10], HiTi [11]) are greater than the threshold are eliminated. However, the heuristic is not applicable to other network distance metrics, such as travel time or cost. It is also not very effective when paths between objects and query points are not in straight lines. As studied in [16], these approaches perform worse than network expansion for the same LDSQs.

Solution based approaches. By pre-computing and maintaining query results for potential access in the future, solution based approaches such as VN³ [13], UNICONS [2], SPIE [7] and Distance Index [6], optimize the search performance for a given type of queries. VN³ [13] employs the concept of Voronoi diagram for nearest neighbor (NN) queries on road networks. For each object, a geometric polygon is formed based on network distances from other neighboring objects and indexed in a spatial index. All points within a polygon should have the enclosed object as their NNs. With VN³, NN search is transformed to a point enclosure problem. UNICONS [2] pre-computes *k*NN objects for some selected nodes. SPIE [7] organizes a network as a set of spanning trees and pre-computes NN results on nodes in the spanning trees. NN queries can be answered by accessing pre-computed results maintained at some of the closest nodes.

Distance Index [6] pre-computes for *all* nodes the object distances and pointers to next nodes towards individual objects, and encodes them as *distance signatures*. Directed by the signatures, both range and NN queries are supported. Instead of precise distances, distance ranges are adopted such that narrow (wide) distance ranges are used to indicate objects nearby (faraway). To determine the precise distances of objects, a search chases the next pointers of nodes to reach some nodes closer to the objects, as signatures there

provide more precise distance ranges of the objects. Based on the more precise distances, objects may be collected as a query result. Figure 3 illustrates the distance signatures on objects o_1 and o_2 stored at n_q and $n_{q'}$. However, we can see both distance ranges for objects are identical, that implies redundant storage. In other words, it incurs excessive storage and pre-computation costs. Our evaluation result also reflect that it is completely impractical.

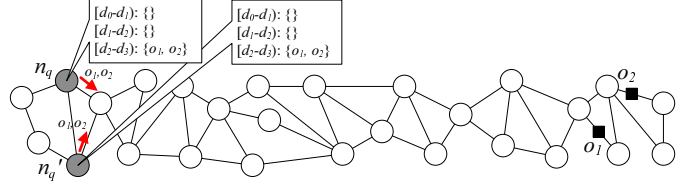


Figure 3: Distance Index

The common pitfalls of solution based approaches are their extremely high overheads incurred in pre-computation, result storage, and maintenance. More importantly, they adapt very poorly to other types of queries, and to objects and network updates.

Hierarchical road networks. Although some existing works such as HEPV [10] and HiTi[11] structure a road network in a hierarchy, ROAD is totally different from them in terms of objectives, designs and implementations. All those existing works focus on shortest path search over a large network. To alleviate memory consumption of storing all-pair shortest paths over a network, they divide a network and materialize shortest paths between the boundaries of partitions and between nodes inside each partition. By concatenating shortest paths from a source to a partition boundary node, to another partition boundary node, and so on until the destination is reached, the shortest path between source and destination is determined. For completeness, all those shortest paths must be maintained and are organized in a tree of sub-networks. Differently, our ROAD divides a network in order to facilitate search space pruning for efficient LDSQ processing. Rather than storing sub-networks in a hierarchy, we maintain a network in a flattened structure as will be discussed in next section to speed up the network expansion. Besides, some shortcuts within Rnets are not kept to save memory and maintenance cost. Furthermore, within the smallest Rnets, no precalculated shortest paths between nodes are needed.

3. THE ROAD FRAMEWORK

In this section, we present the concept, design and implementation of our ROAD framework. We first introduce Rnets, shortcuts and object abstracts, i.e., the key design in support of search space pruning in ROAD, and then discuss Rnet hierarchy formation. More, we present *Route Overlay* and *Association Directory*, the two core components in ROAD implementation.

3.1 Preliminaries

Formally, a road network can be modeled as a weighted graph \mathcal{N} consisting of a set of nodes N and edges E , i.e., $\mathcal{N} = (N, E)$. A node $n \in N$ represents a road intersection or an end point; and an edge $(n, n') \in E$ represents a road segment connecting nodes n and n' . $|n, n'|$ denotes the edge

distance, which can represent the travel distance, trip time or toll of the corresponding road segment, and its value is positive. We simply use distance in the rest of the paper. A path $P(u, v)$ stands for a set of edges connecting nodes u and v and its distance $|P(u, v)| = \sum_{(n, n') \in P(u, v)} |n, n'|$. Among all possible paths connecting node u and node v , the one with the shortest distance is referred to as the *shortest path*, denoted by $SP(u, v)$. The network distance $\|u, v\|$ between u and v is the distance of their shortest path $SP(u, v)$, i.e., $\|u, v\| = |SP(u, v)|$. For simplicity, we assume that objects reside on edges (i.e., road segments) in a network. Objects at nodes (i.e., road intersections) can be treated as they are located at the end of the edges. We denote a set of objects on edge (n, n') by $O(n, n')$ and the distance from an object $o \in O(n, n')$ to the nodes n and n' by $\delta(o, n)$ and $\delta(o, n')$, respectively. Also, we assume LDSQs to be initiated at nodes for simplicity. Each LDSQ is specified with a distance condition D and an attribute predicate A . Given a set of objects in a network, an object, o , is collected as the answer of an LDSQ if (1) its distance from a query node, n_q , denoted by $\|n_q, o\| = \min(\|n_q, n\| + \delta(o, n), \|n_q, n'\| + \delta(o, n'))$ satisfies D (e.g., $\|n_q, o\| \leq 100$) and (2) its attribute denoted by $o.a$ satisfies A (e.g., restaurant $o.type = \text{'seafood'}$). As shown, we single out the conditions of network distance from other attributes due to its importance and focus of this work.

3.2 Rnets, Shortcuts and Object Abstracts

To find objects in terms of their network distances and attributes, a search algorithm may implicitly form a search tree originated at the query node. Following the topology of the network, the portion of the network covered by a search tree conceptually represents a search space. Scanning an entire search space incurs significant traversal overheads. Skipping some search subspaces that do not contain objects of interest from detailed examinations presents an optimization opportunity. This *search space pruning* technique is expected to be very effective in road networks because spatial objects are often clustered and concentrated in some areas, e.g., hotels and resorts are likely to be in business and scenic areas, respectively. Thus, many subspaces do not contain objects of interest and can be pruned. Though well received in various database searches, to the best of our knowledge, the idea of search space pruning has not been exploited in the context of object search on road networks.

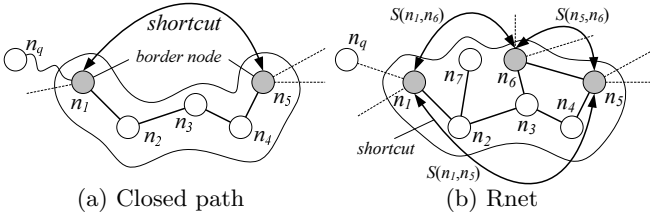


Figure 4: Closed path and Rnet

Figure 4(a) explains how search space pruning can be realized in a road network. Suppose a search tree grows from a node, n_q , to reach a node n_1 . Assume that the path covering edges (n_1, n_2) , (n_2, n_3) , (n_3, n_4) , and (n_4, n_5) represents a *closed path*, i.e., a path that has no nodes connecting to other parts of the network besides the two ending nodes (i.e., n_1 and n_5). If no object of interest presents in the closed path, a detailed traversal on the path can be skipped and the traversal has to continue at n_5 in order to explore ob-

jects thereafter. Considering this closed path as a subspace, we need to have (1) a hint about whether or what objects are on the path; and (2) an artifact at n_1 connecting n_5 , the other end of the path. Accordingly, we introduce *object abstracts* and *shortcuts*. As such, when a closed path is reached and no object of interest is indicated in an object abstract, a search can bypass the entire path via a shortcut to the other end directly. A shortcut between two ending nodes is the shortest path between them.

In road networks, closed paths are usually short; thus the performance gained by bypassing closed paths is rather limited. We, therefore, introduce a notion of Rnets, which stands for *regional sub-networks*, in a road network. Each Rnet encloses a subset of edges and it is bounded by a set of *border nodes*. Each border node is the entrance and exit of an Rnet. The formal definition of Rnet is stated in Definition 1. In an Rnet R , nodes with edges not belonging to R are border nodes. Meanwhile, a border nodes can be shared by more than two Rnets at the same time. Based on Rnets, the concepts of shortcuts and object abstracts are developed and formally stated in Definition 2 and 3, respectively. It is noteworthy that the edges that contribute to $SP(b, b')$ might not necessarily be included in $E_{\mathcal{R}}$.

DEFINITION 1. Rnet. In a network $\mathcal{N} = (N, E)$, an Rnet $\mathcal{R} = (N_{\mathcal{R}}, E_{\mathcal{R}}, B_{\mathcal{R}})$ represents a search subspace, where $N_{\mathcal{R}}$, $E_{\mathcal{R}}$ and $B_{\mathcal{R}}$ stand for nodes, edges and border nodes in \mathcal{R} , and

- (1) $E_{\mathcal{R}} \subseteq E$,
- (2) $N_{\mathcal{R}} = \{n | (n, n') \in E_{\mathcal{R}} \vee (n', n) \in E_{\mathcal{R}}\}$, and
- (3) $B_{\mathcal{R}} = N_{\mathcal{R}} \cap \{n | (n, n') \in E' \vee (n', n) \in E'\}$, where $E' = E - E_{\mathcal{R}}$. \square

DEFINITION 2. Object Abstract. The object abstract of an Rnet \mathcal{R} , $O(\mathcal{R})$, represents all the objects residing on edges in $E_{\mathcal{R}}$, i.e., $O(\mathcal{R}) = \bigcup_{e \in E_{\mathcal{R}}} O(e)$. \square

DEFINITION 3. Shortcut. The shortcut, $S(b, b')$, between border nodes b and b' ($\in B_{\mathcal{R}}$) of an Rnet \mathcal{R} bears the shortest path $SP(b, b')$ and its distance $\|b, b'\|$. \square

Figure 4(b) depicts an Rnet, R , where n_1 , n_5 and n_6 are the border nodes. When a search reaches n_1 , the entire Rnet can be bypassed with shortcuts $S(n_1, n_5)$ to n_5 or $S(n_1, n_6)$ to n_6 if the corresponding object abstract $O(R)$ indicates no object of interest.

3.3 Rnet Hierarchy

In ROAD, we structure a road network as a hierarchy of Rnets where large Rnets at the upper levels enclose smaller Rnets at lower levels. At each level, a network can be viewed as a layer of interconnected Rnets. This structure benefits various search scenarios. For objects located far away from a query node, a search range can be quickly expanded with long shortcuts in large Rnets. For objects that are close to query nodes, shortcuts in moderate-sized Rnets or even original edges can be used to reach the answer objects.

To derive an Rnet hierarchy, we first treat the entire road network as a single Rnet that has no border node and partition it into p_1 partitioned Rnets. Definition 4 states the formal definition of Rnet partitioning. We refer the original Rnet as the level-0 Rnet. The partitioned Rnets are the children of the Rnet they partitioned from. At each subsequent

level i , we partition each Rnet into p_i child Rnets. As a result, at a level x ($\in [0, l]$), the entire network is fully covered by $\prod_{i=1}^x p_i$ interconnected Rnets. For an Rnet hierarchy of l levels, there are $\sum_{h=0}^l \prod_{i=1}^h p_i$ Rnets.

DEFINITION 4. Rnet partitioning. *Partitioning of an Rnet $\mathcal{R} = (N, E, B)$ where N, E, B are a set of nodes, edges and border nodes and $B \subseteq N$, forms p child Rnets, $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_p$ where $p > 1$ and $\mathcal{R}_i = (N_i, E_i, B_i)$. Here, $N = \bigcup_{1 \leq i \leq p} N_i, E = \bigcup_{1 \leq i \leq p} E_i, B \subseteq \bigcup_{1 \leq i \leq p} B_i$. Also, the following three conditions must hold.*

1. *Edges of all child Rnets are disjointed, i.e., $\forall_i \forall_j i \neq j \Rightarrow E_i \cap E_j = \emptyset$.*
2. *Nodes in an Rnet are connected by edges in the same Rnet, i.e., $\forall_i \forall_j (n, n') \in E_i, n \in N_i \wedge n' \in N_i$.*
3. *Border nodes in an Rnet are common to its parent Rnet and some of its sibling Rnets, i.e., $B_i = N_i \cap (B \cup \bigcup_{j \in ([1, p] - \{i\})} N_j)$.* \square

As illustrated in Figure 5, a network N is first partitioned into three Rnets, namely, R_1, R_2 and R_3 , each of which is then partitioned into 2 smaller Rnets, R_{i_a} and $R_{i_b}, i \in [1, 3]$. Consequently, R_1, R_2 and R_3 form the first-level Rnets, and $R_{1_a}, R_{1_b}, R_{2_a}, R_{2_b}, R_{3_a},$ and R_{3_b} form the second-level Rnets. In the figure, n_3 is common to both R_2 and R_3 and hence it is a border node corresponding to the level-1 Rnets. Meanwhile, it is shared by both R_{2_b} and R_{3_a} and is a border node of level-2 Rnets.

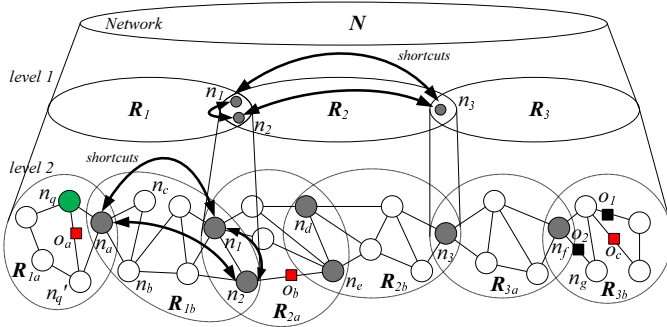


Figure 5: Example Rnet hierarchy

Network Partitioning Methods. An ideal network partitioning should generate equal-sized Rnets and minimize the number of border nodes which in turn minimizes the number of shortcuts formed and maintained. However, this ideal network partitioning is known as NP-complete [15]. In this study, we adopt geometric approach [8] and Kernighan-Lin algorithm (KL algorithm) [12]. The geometric approach first coarsely partitions a network into two by dividing a set of edges spatially such that these two result subnets have equal numbers of edges. KL algorithm is then used to fine tune the two result Rnets by exchanging edges between them until further exchanges do not reduce the number of border nodes. We set p_i to be power of 2 (i.e., $p_i = 2^x$, for x being a positive integer) and recursively apply this binary partitioning until p_i Rnets are formed. This network partitioning approach is also used in [10]. Alternatively, partitioning can

be based on network semantics. For instance, a country-wide road network can be partitioned into levels of states, counties, cities, and townships. Further, the network partitioning could be based on the distributed objects. Since ROAD is a general-purpose framework to support searches on various objects to be mapped onto the same spatial network at the query time, our current network partitioning is performed independently of objects. We will study the object-based network partitioning in our future work.

Creation of Shortcuts and Object Abstracts. After an Rnet hierarchy is formed, object abstracts and shortcuts are constructed in a bottom-up fashion. As edges in child Rnets are fully covered by their parent Rnet (see Definition 4), object abstracts of an Rnet can therefore be constructed directly from their child Rnets. Lemma 1 states this property. On the other hand, the shortcuts of a border node can be determined by adopting Dijkstra's algorithm [4] to explore paths for all other border nodes in the same Rnet. To speed up shortcut computations, shortcuts in Rnets at level i can be calculated based on those in Rnets at level $i+1$ (as stated in Lemma 2). Further, the representation of shortcuts can be based on those shortcuts in child level Rnets. Referring to our example Rnet hierarchy as shown in Figure 5, the shortcut from n_1 and $n_3, S(n_1, n_3)$ can be represented as $(S(n_1, n_d), S(n_d, n_3))$. To determine a detailed shortest path for this shortcut, $S(n_1, n_d)$ and $S(n_d, n_3)$ can be explored at nodes n_1 and n_d , respectively.

LEMMA 1. *The object abstract of a parent Rnet \mathcal{R} fully covers those of all its child Rnets $\mathcal{R}_1, \dots, \mathcal{R}_p$, i.e., $O(\mathcal{R}) = \bigcup_{1 \leq i \leq p} O(\mathcal{R}_i)$. Also, according to Definition 2, the object abstract of a smallest Rnet $R = (N, E, B)$ is $\bigcup_{e \in E} O(e)$.* \square

Proof. The proof is straightforward and is omitted. \blacksquare

LEMMA 2. *Given an Rnet hierarchy, a shortcut $S(b, b')$ between two border nodes of a level i Rnet \mathcal{R} can be derived based on those shortcuts of level $i+1$ Rnets.* \square

Proof. Suppose node b and b' are inside the level $i+1$ Rnets \mathcal{R}_b , and $\mathcal{R}_{b'}$, respectively. If $\mathcal{R}_b = \mathcal{R}_{b'}$, $S(b, b')$ must be the same as the shortcut linking b to b' of Rnet \mathcal{R}_b . Otherwise, $\mathcal{R}_b \neq \mathcal{R}_{b'}$. If \mathcal{R}_b and $\mathcal{R}_{b'}$ are not adjacent, there must be at least one level $i+1$ Rnet \mathcal{R} that bridges \mathcal{R}_b to $\mathcal{R}_{b'}$. Consequently, the shortcut $S(b, b')$ starts at \mathcal{R}_b , passes through \mathcal{R} , and reaches $\mathcal{R}_{b'}$. As the border nodes are the *only* entrances to/exits from Rnets, $S(b, b')$ must go through border nodes of $\mathcal{R}_b, \mathcal{R}$, and $\mathcal{R}_{b'}$. Consequently, the shortcuts which link border nodes must be taken. On the other hand, if \mathcal{R}_b and $\mathcal{R}_{b'}$ are adjacent, there should be at least one border node in common. Through a border node, shortcuts in both Rnets are connected. For all those cases, $S(b, b')$ of a level i Rnet can be constructed by shortcuts in level $i+1$ Rnets and the proof completes. \blacksquare

Besides, explored shortcuts in Rnets can be used to determine other shortcuts of Rnets in the same level as indicated in Lemma 3. Lemma 2 and Lemma 3 can help efficiently compute and update shortcuts (as will be discussed later). To alleviate the storage cost for shortcuts, some shortcuts $S(b, b')$ that are composed of other shortcuts in the same Rnets do not need to be maintained, as stated in Lemma 4. Hence, when a search reaches b , it can transitively reach b'

through other shortcuts in the same Rnet. Similarly, for cases that a shortcut $S(n, n')$ covers completely a reverse path of $S(n', n)$, the detail of either one can be omitted for storage space saving and the distances of those shortcuts can be retained.

LEMMA 3. *Given Rnets \mathcal{R} and \mathcal{R}' at the same level in an Rnet hierarchy, if a shortcut S of \mathcal{R} covers an edge (n, n') of \mathcal{R}' , there must be a shortcut S' corresponding to \mathcal{R}' that covers (n, n') and S must include S' .* \square

Proof. Without loss of generality, we assume that a shortcut $S(a, b)$ reaches Rnet \mathcal{R}' at node n_1 and leaves it at node n_2 , and the path P between n_1 and n_2 inside \mathcal{R}' passes by edge (n, n') . We prove this lemma by contradiction. Assume that i) no shortcut of \mathcal{R}' passes by edge (n, n') , and ii) S passes by edge (n, n') but not any shortcut of Rnet \mathcal{R}' . As S reaches Rnet \mathcal{R}' , and the border nodes are the only entrance to/exits from an Rnet, nodes n_1 and n_2 must be border nodes. As S is a shortcut, its distance $\|a, b\| = \|a, n_1\| + \|P\| + \|n_2, b\|$ must be minimized. Consequently, $\|P\| = \|n_1, n_2\|$ which means P must be the shortest path between n_1 and n_2 , i.e., the shortcut. This violates both assumptions i) and ii), and the proof is completed. \blacksquare

LEMMA 4. *Within an Rnet R , a shortcut $S(b, b')$ between border nodes b and b' can be safely discarded if there exists another border node b'' such that $S(b, b'')$ exactly covers both $S(b, b')$ and $S(b', b')$ in R .* \square

Proof. We omit the proof to save space. \blacksquare

3.4 Route Overlay and Association Directory

To facilitate network traversals that explore a network in a node-by-node fashion, we adopt a node-oriented storage scheme that associates nodes with edges and their corresponding distances to their neighboring nodes. As the network is formulated as a hierarchy of Rnets, one straightforward storage scheme is to store all Rnets where border nodes and shortcuts are nodes and edges as separate networks in addition to the original network as suggested in [10, 11]. This implementation, however, has to maintain separate structures and thus may complicate the search traversals, since search mechanisms need to switch between different networks. Based on Definition 4 that the border nodes in parent Rnets are always the border nodes in some of their child Rnets, our novel index structure, namely *Route Overlay*, that naturally flattens a hierarchical network into a plain network can effectively avoid all the shortcomings of the separate network implementation.

In Route Overlay, nodes are indexed by a B^+ -tree with unique node IDs as search keys². Each leaf entry of B^+ -tree points to a node, together with a *shortcut tree*, i.e., a specialized tree index structure that organizes shortcuts and edges to facilitate search traversals. The structure of a shortcut tree is generally similar to N -ary tree [17] except that i) non-leaf nodes that represent Rnets are associated with shortcuts and ii) number of branches associated with any node (i.e., the number of child Rnets) that is dependent on the number of child Rnets is not fixed. If a given node n is a border node, every non-leaf entry in n 's shortcut tree maintains the shortcuts from n to other border nodes, corresponding to one Rnet, for which n serves as a border node.

²Besides B^+ -tree, alternatives such as Hash index can be used.

Also, in a shortcut tree, parent Rnets are stored immediately above their child Rnets. The leaf entries store all the edges to the neighboring nodes of n . The shortcut tree for a non-border node has only one leaf node containing edges to its neighboring nodes. Figure 6 shows a Route Overlay for our network presented in Figure 5. Take n_q (a non-border node) as an example. Its shortcut tree has only one leaf node that contains edges to n_q 's neighboring nodes, e.g., n_a and n'_q . For n_a (a border node of Rnets R_{1a} and R_{1b}), its shortcut tree has two levels. The first level points to Rnets R_{1a} and R_{1b} , together with shortcuts to other border nodes, e.g., n_1 and n_2 . The second level keeps the edges to neighboring nodes, i.e., n_b, n_c, n_q and n'_q .

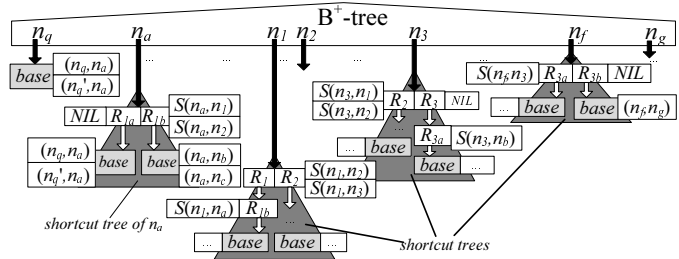


Figure 6: Route Overlay

Next, our proposed efficient object lookup mechanism in ROAD, called *Association Directory* also adopts B^+ -tree with unique node IDs or Rnet IDs as the search key. Associated with node n (n') is an object o in $O(n, n')$ together with its distances $\delta(o, n)$ ($\delta(o, n')$). Similarly, associated with R is the object abstract of an Rnet R . As an Rnet may contain a number of objects, techniques such as aggregated attribute values [20], bloom filter [1], signature [5] can be used to represent an object abstract with fewer storage overheads. Besides, those nodes and Rnets that do not have objects are not kept in the B^+ -tree to further reduce the storage cost. If the search cannot find a node (Rnet) in an B^+ -tree, no object is implied for the node (Rnet).

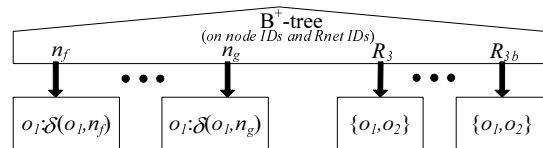


Figure 7: Association Directory

Figure 7 depicts an Association Directory for objects o_1 and o_2 in our example. In the index, an object o_1 on edge (n_f, n_g) is pointed by the nodes n_f and n_g . Moreover, objects o_1 and o_2 in Rnet R_{3b} and its parent Rnet R_3 are associated as $\{o_1, o_2\}$ with the Rnets in the Association Directory. Depending on application needs, other objects, say o_a, o_b, o_c can be placed into the same Association Directory or in a separate Association Directory. This provides flexibility of mapping various objects on the same road network. Moreover, up to the application needs, multiple Association Directories that carry different types of objects can be accessed simultaneously.

4. SEARCH ALGORITHMS

While ROAD is designed to support different types of LD-SQs, in this paper, we focus on k -nearest neighbor (k NN)

queries and range queries. A k NN query (e.g., Q1 in Section 1) returns the k objects of interest closest to n_q . A range query (e.g., Q2 in Section 1) sets a distance range and retrieves all objects of interest with their distances from n_q within the range. Our algorithms based on the idea of network expansion upon ROAD can perform searches efficiently since they navigate Rnets in detail only if those Rnets contain objects of interest; otherwise they bypass those Rnets.

We first discuss the evaluation of k NN query. At the first place, we illustrate the basic idea with a simple network that consists of a chain of nodes in Figure 8. The network is partitioned into 3 Rnets and each of them is further divided into two smaller Rnets. On this network, an NN query is issued at n_2 , and two objects o_1 and o_2 are located on edges (n_{11}, n_{12}) and (n_{12}, n_{13}) , respectively. Also, in this network, nodes n_3, n_5, n_7 and n_9 are border nodes. The search first expands from n_2 to n_1 and n_3 inside R_{1a} . The expansion is shown as a sequence of annotated arrows (arranged vertically) in the figure. Instead of following the physical edges to the right side of the network for objects, a shortcut $S(n_3, n_5)$ at n_3 , i.e., the border node of Rnets R_{1a} and R_{1b} , can be taken to bypass R_{1b} (as no object is indicated by the corresponding object abstract) to reach n_5 .

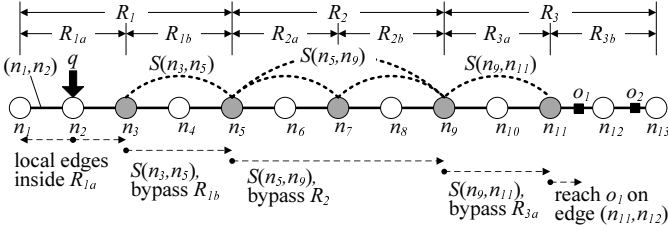


Figure 8: Example 1NN query

Next, a longer shortcut $S(n_5, n_9)$ at n_5 is taken to skip R_2 from detailed traversal. Further, the search at n_9 reaches n_{11} via $S(n_9, n_{11})$. Now, as R_{3b} contains objects, the traversal follows the original edges and the object o_1 is found after exploring n_{11} . From the figure, we can see the search only takes three jumps from n_3 to n_{11} , that significantly saves the traversal cost, compared with traversing original edges between the query node and the objects.

With the logic of network expansion as the basis, our Algorithm k NNSearch (outlined in Figure 9) incorporates shortcuts in Route Overlay and object abstracts in Associate Directory to speed up the search. In general, it iteratively expands the search in a network from n_q by visiting the closest unexplored node. This gradual expansion guarantees the first k objects satisfying search condition to be the k NN objects to the query point. We maintain a priority queue \mathcal{P} to sort pending entries in the non-descending distance order from n_q . Each entry (ϵ, d) in \mathcal{P} records a node or an object (ϵ) and its distance (d) from n_q .

The algorithm takes a Route Overlay (RO), an Associate Directory (AD), a query node (n_q), and a desired number of NNs (k) as inputs, and has all nodes and objects marked “unvisited”. To start, \mathcal{P} is initialized with $(n_q, 0)$ (line 1). Then, the algorithm repeatedly examines the head entry (ϵ, d) from \mathcal{P} until k answer objects are retrieved or the network is completely traversed (lines 2-12). The label “visited” of ϵ is used to avoid visiting (line 4). Otherwise, if ϵ refers to a node, two tasks need to be performed. **SearchObject** is first called to look up AD for objects o associated with the node ϵ and put them as $(o, d + \delta(o, \epsilon))$ to \mathcal{P} for later examination (lines

6-8). Next, Algorithm **ChoosePath** is invoked to decide subsequent nodes from ϵ to continue the network expansion (line 9) that will be discussed next. When ϵ is an object, it is collected into a result set Res (lines 10-11). Thereafter, ϵ is marked “visited” (line 12). Finally, the answer objects are output and the search completes (line 13).

Algorithm k NNSearch(RO, AD, n_q, k)

Input. Route Overlay (RO), Associate Directory (AD), query node (n_q) and the number of NNs (k)

Local. Priority queue (\mathcal{P})

Output. Result set (Res)

Begin

1. enqueue($\mathcal{P}, (n_q, 0)$); $Res = \emptyset$;
2. while (\mathcal{P} is not empty AND $|Res| < k$) do
3. $(\epsilon, d) \leftarrow$ dequeue(\mathcal{P});
4. if (ϵ is marked “visited”) then goto 2;
5. if (ϵ is a node) then
6. $O \leftarrow$ SearchObject(AD, ϵ); // look up AD
7. foreach $(o, \delta(o, \epsilon)) \in O$ do
8. enqueue($\mathcal{P}, (o, d + \delta(o, \epsilon))$);
9. ChoosePath($RO, AD, \mathcal{P}, \epsilon, d$); // see Figure 10
10. else // ϵ is an object.
11. $Res \leftarrow Res \cup \{\epsilon\}$; // ϵ is one of result objects.
12. mark ϵ visited; // this indicates ϵ visited.
13. output Res ;

End.

Figure 9: Algorithm k NNSearch

With shortcut trees organizing shortcuts and edges in accordance with the Rnet hierarchy, Algorithm **ChoosePath** (depicted in Figure 10) can quickly identify appropriate shortcuts and edges to expand the search from a node n . In brief, it examines the shortcut tree of n loaded from Route Overlay in a depth-first traversal manner (lines 2-12). For every non-leaf level, an Rnet R is checked against Associate Directory. If no object of interest is found, R , together with all its child Rnets, are bypassed. The border nodes reachable by the shortcuts are enqueued to \mathcal{P} . Otherwise (i.e., R contains objects of interest), the lookup goes down to the next lower level to examine its child Rnets in a similar fashion (lines 9-10). Once the search reaches the leaf level of the shortcut tree, all neighboring nodes connected by edges are collected (lines 11-12). If n is a non-border node, its shortcut tree contains only edges and all the corresponding neighboring nodes are put into \mathcal{P} .

Algorithm ChoosePath($RO, AD, \mathcal{P}, n, d$)

Input. Route Overlay (RO), Associate Directory (AD), a priority queue (\mathcal{P}), a node (n), distance (d);

Local. Stack (S)

Begin

1. $T \leftarrow$ LoadShortcutTree(RO, n);
2. push($S, T.root$);
3. while (S is not empty) do // search in shortcut tree.
4. $s \leftarrow$ pop(S);
5. if (s is not leaf) then
6. foreach R of s do
7. if (SearchObject(AD, R) has no object) then
8. enqueue($\mathcal{P}, (b, d + |n, b|)$) for all $S(n, b)$ of R ;
9. else
10. push all s 's children to S ;
11. else // leaf node.
12. enqueue($\mathcal{P}, (n', d + |n, n'|)$) for all edges (n, n') in s ;

End.

Figure 10: Algorithm ChoosePath

To visualize Algorithm k NNSearch based on ROAD in

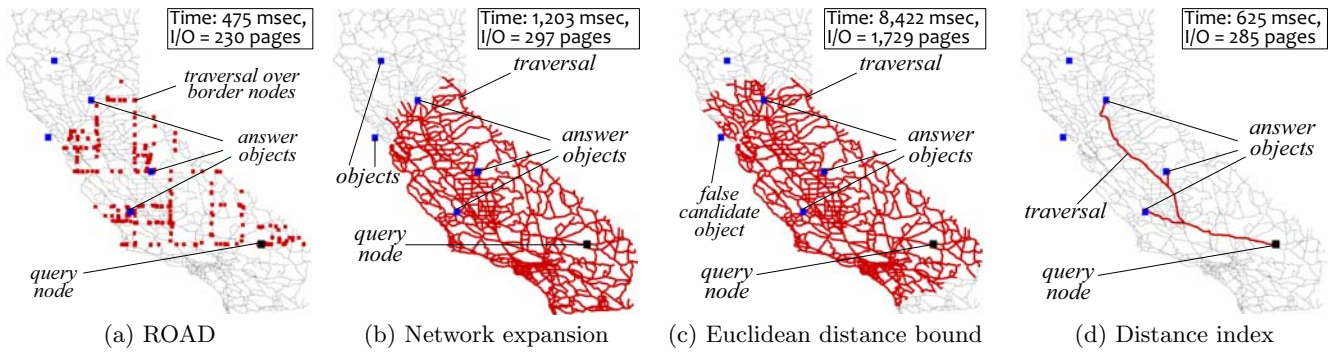


Figure 11: Illustrations on 3NN query

comparison with other existing approaches, Figure 11 shows the evaluation of a 3NN query on 5 objects based upon California road network [14] (see Section 6 for details). As indicated, our algorithm takes the shortest search time and the lowest I/O costs as it quickly expands the search range and reaches the objects via shortcuts. It outperforms other existing works that include network expansion based approaches, Euclidean distance bound approach and distance index. Network expansion based approaches span a very large network area. Euclidean distance bound approaches include false candidate objects. Distance Index incurs a high I/O cost and long search time due to loading a large number of distance signatures, although it has pre-computed paths towards the answer objects. Our algorithm constantly outperforms all those representative approaches, as evaluated in extensive experiments to be presented in Section 6.

Algorithm **RangeSearch** that supports range query is pretty similar to Algorithm **kNNSearch** with a slight difference that the search ends upon a portion of a network within a distance bound is completely traversed, instead of a specified number of objects having been found. All visited objects are the answer objects. To save space, we omit the discussion of the algorithm.

5. ROAD FRAMEWORK MAINTENANCE

In this section, we present the ROAD maintenance in presence of updates that include object changes and network changes. Owing to its clear separation between the objects and network, ROAD handles these two aspects of updates efficiently. We present the update mechanisms below.

5.1 Object Update

Object changes are handled in Association Directory, independently of Route Overlay. To insert an object located on a certain edge (n, n') enclosed by an Rnet R , we associate the object to the nodes n and n' and update the object abstracts of corresponding Rnet R and its ancestor Rnets in an Association Directory. For object deletion, we can simply remove the association of the objects from corresponding edges and from the object abstracts of corresponding Rnets in an Association Directory. On the other hand, for the changes of object attributes, we update the object abstract associated with nodes and Rnets.

5.2 Network Update

Road condition and road network structure change over time. Instead of immediately rebuilding a Route Overlay upon changes that is expensive, we propose several tech-

niques to incrementally update Route Overlay for *edge distance changes*, and *network structure changes*.

5.2.1 Change of Edge Distance

In ROAD, when the distance of an edge that represents the travel distance, trip time or cost of a road segment changes (increases or decreases), some shortcuts that represent shortest paths might become invalid and have to be updated. Here, these updates only affect Route Overlay but not objects. To save unnecessary shortcut re-computations, ROAD adopts a *filtering-and-refreshing* approach that consists of two steps. In the “filtering” step, shortcuts that may be affected by the change are identified. The identified shortcuts are then updated in the “refreshing” step. According to Lemma 2 defined in Section 3, the update of shortcuts related to level i Rnets in an Rnet hierarchy is not necessary unless shortcuts related to level $i+1$ Rnets are updated. Thus, in the following, we only explain how to re-compute shortcuts in the bottom level. The same idea can be applied to upper levels. Also, based on Lemma 3, an edge, which is not covered by shortcuts in its own Rnet, is definitely not covered by shortcuts in other Rnets at the same level. Therefore, we examine the shortcuts in an Rnet that encloses the changed edge first. If no shortcut update is incurred, the update can be safely terminated. Suppose an edge changes its distance $|n, n'|$ from d to d' , detailed update procedure is discussed below.

Edge distance increased (i.e., $d < d'$). When the distance of an edge (n, n') in an Rnet R is increased from d to d' , only those shortcuts that cover (n, n') might become invalid and need to be refreshed. In the filter step, we identify shortcuts that pass through (n, n') . Observing that a shortcut $S(b, b')$ covering (n, n') should have $\|b, b'\|$ equal to $\|b, n\| + |n, n'| + \|n', b'\|$ (where we consider $|n, n'|$ before update, i.e., d), we search affected shortcuts by finding the shortest paths from each of end nodes (n and n') to the border nodes in R and then identifying shortcuts whose distances are equal to the path passing through (n, n') . In the second phase, all the identified shortcuts are re-evaluated. If no shortcuts are refreshed, the update terminates. Otherwise, the update is propagated up to the parent level, with border nodes and shortcuts at the current level treated as nodes and edges, respectively.

Edge distance decreased (i.e., $d > d'$). When the distance of an edge (n, n') in an Rnet R is decreased from d to d' , it may contribute to paths shorter than some existing shortcuts. In this case, those shortcuts need to be identified and refreshed. In the first filtering step, we test if the distance of a path from border node b via (n, n') to

another border node b' (with $|n, n'| = d'$, the new edge distance) is shorter than the distance of the shortcut $S(b, b')$. Here we expand from n and n' to reach border nodes and to determine the distances as shown in Figure 12(a). Once $\|b, n\| + |n, n'| + \|n', b'\| < \|b, b'\|$, $S(b, b')$ between border node b and b' is identified to be affected. In the second phase, those identified paths are replaced by the new paths passing by edge (n, n') . Again, the update process will be propagated to the parent level if there are shortcuts updated.

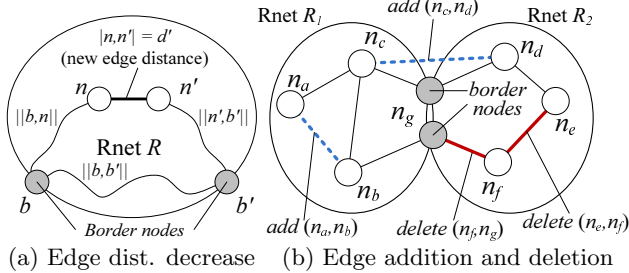


Figure 12: Network changes

5.2.2 Change of Network Structure

When new roads are constructed or existing roads are closed, the corresponding network topology is changed. We model these changes as addition or deletion of nodes and edges. As changes of nodes result in changes of edges, we treat changes of nodes as special cases of changes of edges, and only consider addition and deletion of edges below. Again, we update the network at the bottom level first and propagate the updates to the parent levels if necessary.

Addition of a new edge. A newly added edge (n, n') directly connects two nodes n and n' , assuming that n and n' belong to Rnet \mathcal{R} and \mathcal{R}' , respectively. There are two possible cases: (1) $\mathcal{R} = \mathcal{R}'$ and (2) $\mathcal{R} \neq \mathcal{R}'$. We handle them in the following.

- **Case 1:** $\mathcal{R} = \mathcal{R}'$ (i.e., both nodes are located inside the same Rnet). Adding an edge connecting two nodes (e.g., (n_a, n_b) in Figure 12(b)) can be treated as changing the distance of an edge from infinity to the edge distance. The previously discussed edge distance update mechanism can be applied here. Accordingly, the Route Overlay is updated as well to store the new edge and the new nodes (if any).
- **Case 2:** $\mathcal{R} \neq \mathcal{R}'$ (i.e., nodes are located in different Rnets). Since an edge can only be included by one Rnet (say \mathcal{R}), the node n' which does not belong to \mathcal{R} , has to be promoted to a border node between \mathcal{R} and \mathcal{R}' . In Figure 12(b), the introduction of (n_c, n_d) to R_1 gets n_d promoted to the border node. Also, the new edge (n, n') might affect some shortcuts. The update approach for the change of edge distance can be applied here. As a new border node is introduced, new shortcuts linking the new border node to other border nodes in the same Rnet have to be created.

Deletion of an existing edge. Deleting an edge (n, n') breaks the link between two nodes n and n' . Consider deleting (n_e, n_f) in R_2 in Figure 12(b). Its deletion can be managed as handling the change of its edge distance to infinity and updating affected shortcuts. In addition, it is possible that one of the end nodes of a deleted edge is a border node. If all the edges of n are within one Rnet after the deletion of

edge (n, n') , n is no longer a border node. As shown in Figure 12(b), after deleting (n_f, n_g) , n_g becomes a non-border node. Then, the shortcut trees of n and other border nodes in related Rnets in Route Overlay have to be updated.

6. PERFORMANCE EVALUATION

This section evaluates our proposed ROAD framework in terms of indexing overhead, maintenance overhead, and query performance. We applied ROAD (labeled as ROAD, hereafter) on three real road networks, namely, CA, NA and SF obtained from [14]. CA and NA consist of highways in California, USA and North America, respectively. SF is composed of streets and roads in San Francisco. In this evaluation, we simulate one type of objects based on which all the queries are evaluated, while ROAD can handle diverse objects. Objects, with number varying from 10 to 1000, are evenly distributed over those road networks³. The fewer the number of objects is in the network, the larger the search subspaces that contain no objects can be pruned. Thus, an efficient approach should be able to return the search result quickly when a small number of objects is experimented. Table 1 summarizes the evaluation parameters, their values and defaults used in the experiments.

In addition to ROAD, we implement network expansion [16], Euclidean-based approach [16, 19] and Distance Index [6] (labeled as NetExp, Euclidean, and DistIdx, respectively), in GNU C++ for comparisons. We adopt CCAM [18] to organize network nodes in storage for all the approaches. For NetExp, objects are stored with network nodes. For Euclidean, objects are indexed by an R-tree and the A* algorithm [3] is used to determine objects' network distances from query nodes. For DistIdx, distance signatures are stored with network nodes. We adopt exact object distances in the distance signature to provide the optimal search performance.

Parameter	Value (*=default)
Network	CA* (21,048 nodes, 21,693 edges) NA (175,813 nodes, 179,179 edges) SF (174,956 nodes, 223,001 edges)
No. of objects ($ O $)	10, 50, 100* , 500, 1000
Partition factor (p)	4*
No. of levels (l)	2, 3, 4* , 5, 6 for CA, and 6, 7, 8* , 9, 10 for NA and SF
Query	k NN query* and range query
No. of NNs (k)	1, 5* , 10
Search range (r)	0.05, 0.1* , 0.2 of network diameter

Table 1: Evaluation parameters

We measure the performance of all the approaches according to four commonly used performance metrics, namely,

- *Index construction time.* This measures the elapsed time to construct an index.
- *Index size.* This measures storage consumed to store an index.
- *Index update time.* This measures the time spent on updating an index in presence of object and network changes.
- *(Query) Processing time.* This represents the time duration from the time when the query is initiated to the time that a complete result is obtained.

³ROAD can benefit more from uneven object distribution that more empty subspaces can be pruned.

All indices are stored on disk. In our disk storage, the page size is fixed at 4KB. We also employ a memory cache of 50 pages with LRU replacement scheme to buffer loaded pages. In every run, a query is initialized with an empty cache. All experiments were conducted upon Linux 2.6.9 servers with Intel Xeon 3.2GHz CPU. In what follows, we evaluate the index overhead, index maintenance overhead, query performance followed by Rnet hierarchy settings.

6.1 Indexing Overhead

The first set of experiments evaluates the index construction time and index sizes of all the approaches for various number of objects and networks. We consider NetExp that has no index on objects as the baseline in this evaluation. Because of various network sizes, we fix p to 4 for all the networks, while l 's for NA and SF are set to 8 and that for CA is set to 4. We shall study the impacts of the settings of Rnet hierarchy level (l) later. Figure 13 shows the index construction time (in second) and index sizes (in metabyte) of varying number of objects on CA (in log scale). As shown in the figure, NetExp, Euclidean, and ROAD incur almost constant index construction time (in a few minutes) and index size (in a few MBs) while DistIdx increases drastically in both construction time and index size. For 1,000 objects, DistIdx takes more than 240MB for index storage and nearly half an hour to build an index! The trend keeps increasing with the increase of the number of objects. This finding reveals that DistIdx is not practical for use in realistic applications.

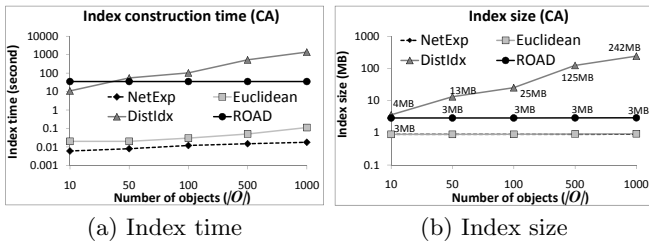


Figure 13: Index on various cardinalities

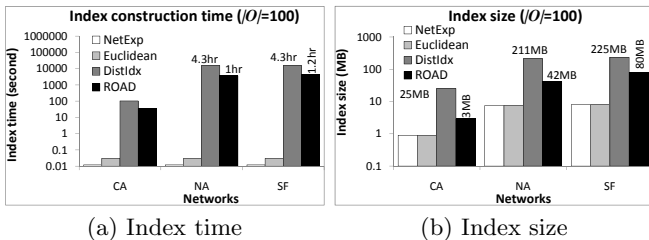


Figure 14: Index on various network

Figure 14 shows the index construction time and index size for different networks with the number of objects fixed at 100. As shown in the figure, NetExp and Euclidean incur very short index construction times and take less storage. Both DistIdx and ROAD vary with networks. However, they differ a lot in terms of efficiency. DistIdx takes more than 4 hours to build and more than 210MB to store an index for NA and SF. ROAD incurs considerably shorter construction time (about 1 hour) and less storage space (<100MB). For SF, ROAD is about $\sim 25\%$ of indexing time and $\sim 33\%$ of index size of DistIdx. Recall that the cost of DistIdx increases if more objects are included. However, the index construction cost for ROAD is only attributed to the formation of Route Overlay, which is totally independent of the number

of objects included. For a large system that needs to support a huge amount of different types of objects, this index construction time and index storage incurred by ROAD can be amortized, but the costs incurred by DistIdx that are related to the number of objects cannot. While the indexing costs of ROAD are expected to be higher than NetExp and Euclidean, as to be shown next, ROAD is actually very efficient for updates and query processing.

6.2 Maintenance Overhead

Here, we evaluate the index update time for object changes and network changes. We first evaluate the update time for object changes. In this experiment, we delete one randomly picked object from a network and then add it back at a random location. We repeat deletion/insertion for 100 times. The average performances of insertions and deletions are presented in Figure 15. DistIdx incurs several orders of magnitude higher update costs than others. For NA and SF, it takes about 2 minutes to finish *one* object deletion or addition. This is because it has to traverse entire networks to update all distance signatures. In contrast, NetExp, Euclidean and ROAD can handle update within 0.1 second for all the networks.

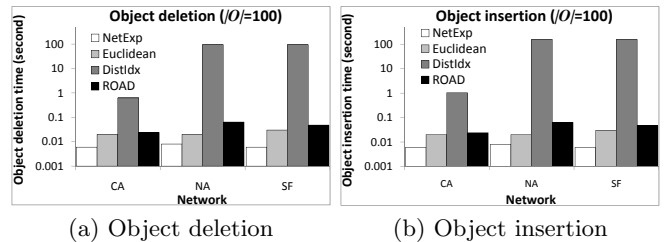


Figure 15: Object update

Similarly, we perform network change by randomly removing one edge by setting its edge distance to infinity and adding it back by recovering its original distance. The average performance of 100 trials is presented in Figure 16. The edge change almost has no observable impact on NetExp and Euclidean. However, for DistIdx, distance signatures of many nodes have to be reexamined and updated that involves a lot of disk-write operations. Differently, ROAD only needs to update affected shortcuts between some border nodes. Thus, it has considerably lower update costs than DistIdx and it takes less than 2 seconds for NA and SF. NetExp and Euclidean are very update-efficient. However, they are not query efficient as to be evaluated next.

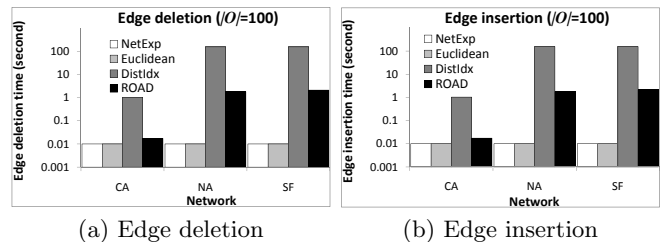


Figure 16: Network update

6.3 Query Performance

Further, we measure the query performance of all the approaches over different numbers of objects, networks and query types. We evaluated 100 queries issued at random positions and report the average performance in terms of processing time.

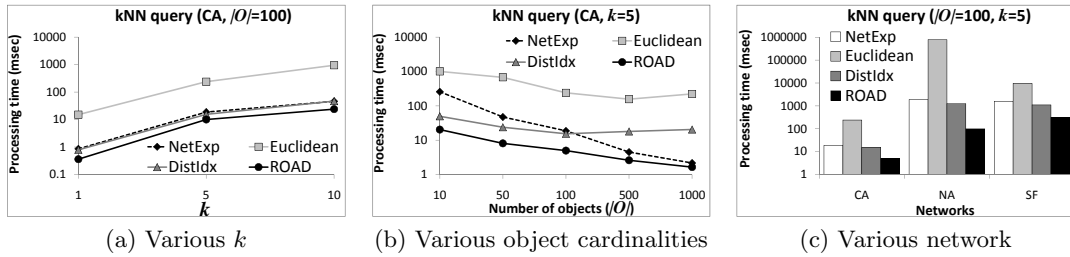


Figure 17: Query performance based on k NN query

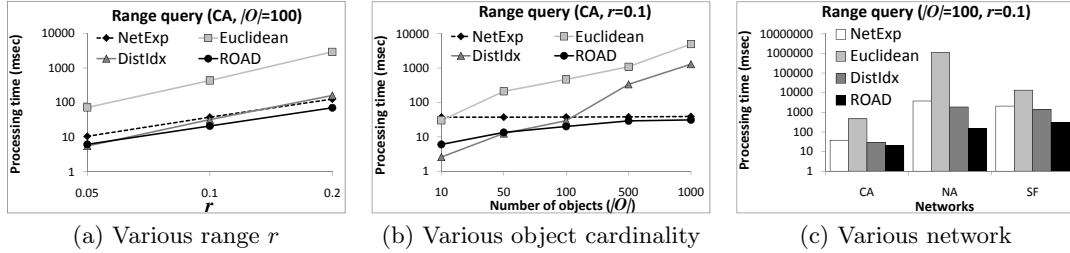


Figure 18: Query performance based on range query

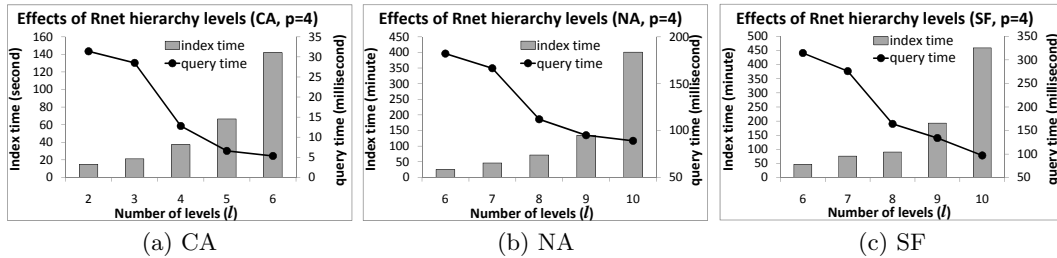


Figure 19: Impact of Rnet hierarchy level (l)

Evaluation based on k NN query. Our first experiment evaluates the query performance based on k NN query against different factors, namely, the query parameter k , the number of objects in the network and different networks. We first evaluate the query parameter k varying from 1 to 5 and to 10 while the network and the number of objects are fixed at CA and 100, respectively. The results for all approaches are plotted in Figure 17(a). From the figure, we can see that Euclidean takes the longest processing time for all evaluated k 's as it suffers from false hits and incurs redundant shortest path searches over the same portion of the network. DistIdx performs slightly better than NetExp since DistIdx has distance signatures to guide the search towards result objects that saves network traversal overhead, but on the other hand, these bulky distance signatures incur higher I/O that outweighs the performance gain. For all evaluated k 's, ROAD performs constantly the best, attributed to the effectiveness of search space pruning.

Figure 17(b) plots the results obtained by varying the object cardinality from 10 up to 1,000 while both network and k are fixed at CA and 5. In general, increases of object cardinalities will reduce the average distance of objects from a query point; thus the search range is reduced. As a result, performance (i.e., processing time) is improved (shortened). However, as shown in the figure, the performance for Euclidean and DistIdx increase initially and then drop. For Euclidean, this is because of an increase of false hit. For DistIdx, it is caused by the increased size of distance signatures. On the other hand, the performance for both NetExp and

ROAD improves continuously as we expected. It is noteworthy that the difference between them narrows since ROAD is also expansion-based; when the objects are close to query points, objects are likely to be found within the same Rnets where query points are located.

Finally, we evaluate k NN query on different approaches upon various networks where k and $|O|$ are fixed at 5 and 100, respectively. The result is depicted in Figure 17(c). As observed from the results with CA, Euclidean performs the worst. It is noteworthy that NA and SF have different numbers of edges and different node densities though similar numbers of network nodes, approximation of Euclidean distance to network distance is more appropriate for SF than NA. DistIdx perform slightly better than NetExp. Last, ROAD performs the best.

Evaluation based on range query. Our second evaluation is on range query against different approaches. We first evaluate the query parameter, r , that represents the search range. Figure 18(a) shows the results obtained by varying r from 0.05 up to 0.2 of the network diameter while we use CA and fix $|O|$ at 100. In general, processing times of all approaches increase when r increases. Among all the approaches, ROAD consistently performs better than all the others. DistIdx takes shorter processing time when small r is used. However, when r is increased, due to the high cost of loading of a number of distance signatures, the performance of DistIdx drops. At last, Euclidean performs the worst.

Next, we evaluate the impact of object cardinality ($|O|$) that ranges from 10 up to 1,000. The result is plotted in

Figure 18(b). Due to fixed range ($r = 0.1$), the network traversal cost is reasonably fixed. Thus, we can see that NetExp aligns with our expectation. However, due to different reasons, all the other approaches have their processing time increased. The processing time of Euclidean is caused by false hits and that of DistIdx is attributed to the increased size of distance signatures. Finally ROAD gains performance improvement by exploiting search space pruning. When object cardinality increases, the performance of ROAD gets closer to NetExp as ROAD performs almost the same as network expansion. Finally, the results based on various network are plotted in Figure 18(c). In general, the observation is pretty much the same as what obtained from experiments on k NN query and it can be explained similarly.

6.4 Evaluation of Rnet hierarchy

Last, we evaluate the impact of the number of levels (l) on the index overhead and query processing time. In this evaluation, we vary l for CA from 2 to 6 and that for both NA and SF from 6 to 10. To be specific, we measure the index construction time and processing time for k NN query ($k = 5$) while the numbers of objects for all the networks are fixed at 100 and p is fixed at 4. The results are shown in Figure 19. With the increase of Rnet hierarchy levels, the index time increases and query processing time drops exponentially. Although there are no absolute optimal points found, we can see a significant drop in query performance when $l = 4$ for CA and $l = 8$ for NA and SF that we used as default Rnet hierarchy levels in our experiments.

7. CONCLUSION

The rapid growth of LBSs fosters a need of efficient search algorithms for LDSQs. In the mean time, the on-going trend of web-based LBSs demands a system framework that can flexibly accommodate diverse objects, provide efficient processing of various LDSQs, and support different distance metrics. To meet those needs, we propose ROAD, a system framework for efficient LDSQ processing, in this paper. The design of ROAD achieves a *clean separation* between objects and network for better system flexibility and extensibility. It exploits *search space pruning*, an effective and powerful technique for object search. Upon the framework, efficient search algorithms for common LDSQs, namely, range and k NN queries, are devised and incremental framework maintenance techniques are developed. Via comprehensive experiments on real road networks, ROAD is shown to outperform the state-of-the-art techniques. As our future works, we are going to derive an analytical model of ROAD and to develop a prototype to evaluate its performance of the framework as well as to devise algorithms to support LDSQs other than those discussed in the paper.

8. ACKNOWLEDGEMENTS

Wang-Chien Lee and Ken C. K. Lee are supported in part by the National Science Foundation under Grant no. IIS-0534343 and CNS-0626709.

9. REFERENCES

- [1] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of ACM*, 13(7):422–426, 1970.
- [2] H.-J. Cho and C.-W. Chung. An Efficient and Scalable Approach to CNN Queries in a Road Network. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, Trondheim, Norway, Aug 30 - Sep 2, pages 865–876, 2005.
- [3] R. Dechter and J. Pearl. Generalized Best-First Search Strategies and the Optimality of A*. *Journal of ACM*, 32(3):505–536, 1985.
- [4] E. W. Dijkstra. A Note on two Problems in Connexion with Graphs. In *Numerische Mathematik*, pages 269–271, 1959.
- [5] C. Faloutsos and S. Christodoulakis. Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation. *ACM Transactions on Information Systems (TOIS)*, 2(4):267–288, 1984.
- [6] H. Hu, D. L. Lee, and V. C. S. Lee. Distance Indexing on Road Networks. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, Seoul, Korea, Sep 12-15, pages 894–905, 2006.
- [7] H. Hu, D. L. Lee, and J. Xu. Fast Nearest Neighbor Search on Road Networks. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT)*, Munich, Germany, Mar 26-31, pages 186–203, 2006.
- [8] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. Effective Graph Clustering for Path Queries in Digital Map Databases. In *Proceedings of the 5th International Conference on Information and Knowledge Management (CIKM)*, Rockville, MD, USA, Nov 12-16, pages 215–222, 1996.
- [9] C. S. Jensen, J. Kolárvr, T. B. Pedersen, and I. Timko. Nearest Neighbor Queries in Road Networks. In *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems (ACM GIS)*, New Orleans, Louisiana, USA, Nov 7-8, pages 1–8, 2003.
- [10] N. Jing, Y.-W. Huang, and E. A. Rundensteiner. Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 10(3):409–432, 1998.
- [11] S. Jung and S. Pramanik. An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(5):1029–1046, 2002.
- [12] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Systems Technical Journal*, 49(2):291–308, 1970.
- [13] M. R. Kolahdouzan and C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, Toronto, Canada, Aug 31 - Sep 3, pages 840–851, 2004.
- [14] F. Li. Real Datasets for Spatial Databases: Road Networks and Points of Interest. <http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm>.
- [15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.
- [16] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query Processing in Spatial Network Databases. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB)*, Berlin, Germany, Sep 9-12, pages 802–813, 2003.
- [17] B. R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*. 1998.
- [18] S. Shekhar and D.-R. Liu. CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 9(1):102–119, 1997.
- [19] M. L. Yiu, N. Mamoulis, and D. Papadias. Aggregate Nearest Neighbor Queries in Road Networks. *IEEE Transactions on Knowledge Data Engineering (TKDE)*, 17(6):820–833, 2005.
- [20] D. Zhang and V. J. Tsotras. Optimizing Spatial Min/Max Aggregations. *VLDB Journal*, 14(2):170–181, 2005.