ROAD: A New Spatial Object Search Framework for Road Networks

Ken C.K. Lee, Wang-Chien Lee, Baihua Zheng, and Yuan Tian

Abstract—In this paper, we present a new system framework called *ROAD* for spatial object search on road networks. ROAD is extensible to diverse object types and efficient for processing various location-dependent spatial queries (LDSQs), as it maintains objects separately from a given network and adopts an effective search space pruning technique. Based on our analysis on the two essential operations for LDSQ processing, namely, network traversal and object lookup, ROAD organizes a large road network as a hierarchy of interconnected regional subnetworks (called *Rnets*). Each Rnet is augmented with 1) *shortcuts* and 2) *object abstracts* to accelerate network traversals and provide quick object lookups, respectively. To manage those shortcuts and object abstracts, two cooperating indices, namely, *Route Overlay* and *Association Directory* are devised. In detail, we present 1) the Rnet hierarchy and several properties useful in constructing and maintaining the Rnet hierarchy, 2) the design and implementation of the ROAD framework, and 3) a suite of efficient search algorithms for single-source LDSQs and multisource LDSQs. We conduct a theoretical performance analysis and carry out a comprehensive empirical study to evaluate ROAD. The analysis and experiment results show the superiority of ROAD over the state-of-the-art approaches.

Index Terms—Location-dependent spatial query, spatial road network, indexing techniques, and search algorithms.

1 Introduction

While location-based services (LBSs) are booming in this decade, many vendors start to provide map and navigation services (e.g., Garmin, GoogleMap, MapQuest, NavTeq, Yahoo! Map) along with convenient geo-tagging tools that enable the content providers (e.g., retail stores, facilities and general users) to publish location-dependent information on digital maps [1], [2]. Here, we refer to location-dependent information (e.g., point of interest, traffic, and local events) as *spatial objects* (or *objects* for short). We define queries that search for spatial objects with respect to user-specified locations as *location-dependent spatial queries* (LDSQs). For people participating a conference as an example, some useful LDSQs include Q1: find hotels within one mile from the conference venue; Q2: locate the nearest bus station to the conference venue; and Q3: find a restaurant closest to the hotels of the conference participants.

In processing LDSQs on a road network, two basic operations, namely, *network traversal* and *object lookup*, are involved. The former visits network nodes and edges according to network proximity, while the latter accesses and checks the attributes of objects located at traversed nodes or edges against object search criteria. Objects collected during the course of a traversal form a query result. Logically, the more network traversals and object lookups are involved, the larger the query processing overhead is incurred. As shown in Fig. 1, where a network is modeled as a graph, two objects o_1 and o_2 are on one side of the network. If a nearest neighbor (NN) query is issued far away on the other side, say at n_q, the search cost is apparently higher than another NN query issued somewhere close to the objects.

As network traversals and object placements are constrained by the network topology, nodes and edges (i.e., the entire network) conceptually form an object *search space*. Observing some search subspaces (e.g., the middle portion bounded by a dashed line in Fig. 1) do not have objects of interest, we could facilitate network traversals by *pruning* those subspaces without objects of interest. This observation inspires an idea of *search space pruning*, based on which we design a novel, efficient, and extensible system framework, called *ROAD*, for processing LDSQs on road networks.

In ROAD, a network is first formulated as a set of interconnected regional subnets called *Rnets*, each representing a search subspace. On top of the Rnets, two kinds of additional information are derived: 1) selective (shortest) paths across an Rnet that enable any traversal to bypass the Rnet if it has no object of interest, and 2) the existence and/or contents of objects that are inside the Rnets to provide quick traversal guidelines. Both 1) and 2) are further elaborated into the notions of *shortcuts* and *object abstract*, respectively. Let us revisit Fig. 1. Two highlighted shortcuts, one from n_1 to n_3, and the other from n_2 to n_3, direct the NN search issued at n_q to bypass the Rnet R, and enable the traversal to continue at node n_3, since Rnet R has no object of interest, as indicated by its empty object abstract.

To realize ROAD, we propose two novel index structures, namely, *Route Overlay* and *Association Directory* (and ROAD is named after these two key components). The

- K.C.K. Lee is with the Department of Computer and Information Science, University of Massachusetts Dartmouth, North Dartmouth, MA 02747. E-mail: ken.ck.lee@umassd.edu.
- W.-C. Lee and Y. Tian are with the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802. E-mail: {wlee, yxt144}@cse.psu.edu.
- B. Zheng is with the School of Information Systems, Singapore Management University, Singapore. E-mail: bhzheng@smu.edu.sg.

Manuscript received 30 Sept. 2009; revised 19 Jan. 2010; accepted 31 July 2010; published online 17 Nov. 2010.
Recommended for acceptance by T. Sellis.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2009-09-0690.
Digital Object Identifier no. 10.1109/TKDE.2010.243.

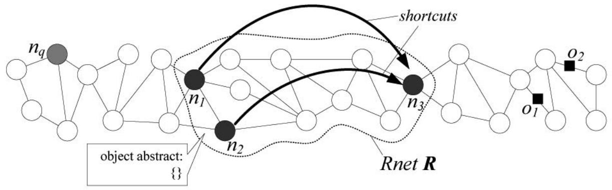


Fig. 1. Basic idea behind ROAD framework.

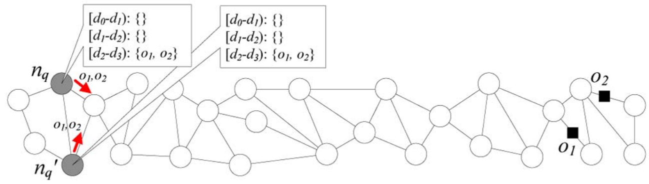


Fig. 2. Distance Index.

former manages the physical network structure (i.e., nodes and edges) and the shortcuts, while the latter manipulates the mappings of objects and object abstracts on nodes, edges, and Rnets. In this paper, we detail the design, implementation, and evaluation of ROAD, and provide a holistic solution to several important research issues that include organization of Rnets, search algorithms for various LDSQs, and framework updates. We also perform an analysis and simulation to evaluate the ROAD performance. In summary, the significant contributions presented in this paper are seven-fold:

1. We present *ROAD*, a novel system framework to support spatial object searches on road networks. ROAD cleanly separates the road network and objects, exploits the idea of search space pruning, and supports searches with different distance metrics.
2. We formulate Rnet hierarchy and explore several properties to reduce indexing overhead and improve query and update performance.
3. We devise efficient search algorithms for single-source range queries and (k)NN queries, i.e., classical types of LDSQs, upon the ROAD framework.
4. We devise efficient search algorithms for multi-source range queries and (k)NN queries to illustrate the extensibility of ROAD for different LDSQs.
5. We develop efficient update techniques for ROAD maintenance to handle object and network updates.
6. We provide a theoretical analysis on the space and time efficiency of ROAD.
7. We evaluate the performance of ROAD and compare it with 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 concepts behind ROAD. Sections 4 and 5 discuss the query processing algorithms for single-source LDSQs and multi-source LDSQs, respectively. Section 6 discusses the framework maintenance. Section 7 analyzes the ROAD performance. Section 8 evaluates ROAD compared with existing works through simulations. Finally, Section 9 concludes this paper.

2 Related Work

Existing works on processing LDSQs on road networks are categorized as *solution-based approaches* and *extended spatial database approaches*, which are reviewed below.

Solution-based approaches (such as VN^3 [3], UNICONS [4], SPIE [5], and Distance Index [6]) utilize some precomputed results to evaluate LDSQs. While VN^3, UNICONS, SPIE only cater for NN queries, Distance Index supports both range and NN queries. Distance Index precomputes for *all* nodes the distances and pointers to subsequent nodes on paths toward individual objects, and encodes them as *distance signatures*. To reduce the storage overhead of distance signatures, distance ranges, rather than precise distances, are adopted such that distances within one range share the same signature. Fig. 2 illustrates the distance signatures on objects o_1 and o_2 stored at n_q and $n_{q'}$, with $d_0 < d_1 < d_2 < d_3$. As we can observe, distance ranges maintained about objects located at nearby nodes can be very similar or even identical, thus consuming redundant storage. This, in fact, incurs impractically large storage overhead.

In general, the pitfalls of solution-based approaches include high precomputation overhead, massive storage overhead, and expensive result maintenance cost. Besides, they adapt poorly to other types of queries, and to objects and network updates.

Extended spatial database approaches incorporate road networks to existing spatial databases. Two basic search strategies were studied. The first strategy is based on the idea of *euclidean distance bound* [7], [8]. By this strategy, approaches first identify candidate objects that have euclidean distances to the query point bounded by a distance threshold. Then, they determine network distances between individual candidate object and the query point based on shortest path algorithms [9], [10] or materialized distances [11], [12]; and finally, they discard false candidates whose network distances actually are larger than the threshold. The second strategy is based on *network expansion* that gradually expands a search range on a network until all the nodes and edges that satisfy the search criteria are visited [7]. Objects on those visited nodes and edges form the result set. Although more efficient than euclidean distance bound approaches, network expansion approaches are still inefficient due to the slow *node-by-node* expansion toward *all* directions.

Distance browsing [13] has been recently proposed based on the concept of path coherence [14] that for any node n, all other nodes with their shortest paths from n via one of n's immediate neighboring nodes are spatially close. Based on this idea, shortest path quad-tree (SPQT) [13] (i.e., a spatial quad-tree) indexes all other nodes with respect to each node n. In a SPQT, each quad-cell $T_n(n')$ is a rectangular spatial area associated with one of n's neighboring nodes n' and the shortest path distance, $d_n(n')$, from n to all nodes in $T_n(n')$. Then, given a node n and a target node v, the quad-cell $T_n(n')$ that covers v can be located. As such, $d_n(n')$ can be identified as the lower bound of the network distance from n to v and the shortest path should follow node n' to approach v. As shown in Fig. 3, given an NN search issued at n_q, it first maps object locations to n_q's SPQT and then traverses to a neighboring node n_q'' as $T_{n_q}(n_q'')$ covers the objects and $d_{n_q}(n_q'')$ is the smallest. It avoids blind network traversal. However, this approach still

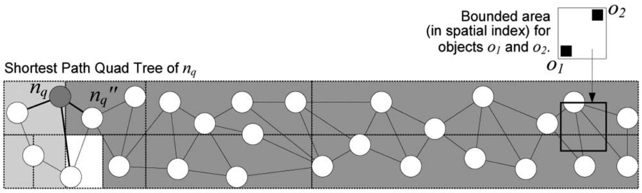


Fig. 3. Distance browsing on road network.

needs to traverse the network in a node by node fashion. Even worse, bulky SPQTs accessed for all visited nodes make this approach very I/O-inefficient. Further, computationally expensive all-pair shortest paths need to be computed in advance.

Our ROAD differs from all those existing works as it treats a road network as an object space and utilizes search space pruning—which has not been explored in this context—to enhance search performance. Also, our ROAD is scalable to networks as it organizes a road network as a hierarchy, similar to HEPV [11] and HiTi [12]. While the design and implementation of HEPV and HiTi only allow point-to-point shortest path searches, our ROAD facilitates network expansion to reach multiple destinations for objects, which is important for LDSQs.

3 The ROAD Framework

In this section, we introduce *Rnets*, *shortcuts*, and *object abstracts*, and discuss the formation of Rnet hierarchy, i.e., the key design in support of search space pruning in ROAD. Besides, we present the core of ROAD implementation, namely, *Route Overlay* and *Association Directory*.

3.1 Preliminaries

We model a road network $\mathcal{N}$ as a weighted graph that consists of a set of nodes N and edges E, i.e., $\mathcal{N} = (N, E)$. A node $n \in N$ represents a road intersection and an edge $(n, n') \in E$ represents a road segment connecting nodes n and n'. $|n, n'|$ denotes the edge weight, which represents the travel distance, or trip time, or toll of (n, n'), and we assume all edge weights are positive. For simplicity, we use distance hereafter. We also denote the shortest path distance between node n and n' by $\|n, n'\|$. Given a set of objects, $\mathcal{O}$, we consider that objects (in $\mathcal{O}$) reside on edges (i.e., road segments) in a network.[1] We denote a mapping function $L(n, n')$ on an edge (n, n') to represent a set of objects on the edge. Additionally, we use $\delta(o, n)$ ($\delta(o, n')$) to represent the distance between an object $o \in L(n, n')$ and the endpoint n (n') of the edge.

In our discussion, all LDSQs are assumed to be initiated at nodes without loss of generality.[2] In general, each LDSQ is specified with a distance condition D and an attribute predicate A. Given $\mathcal{O}$ mapped on $\mathcal{N}$ and a query node n_q, an object, o (in $\mathcal{O}$), is collected as the answer to an LDSQ if (1) its distance from n_q, denoted by $\|n_q, o\|$ (i.e., $min(\|n_q, n\| + \delta(o, n), \|n_q, n'\| + \delta(o, n')))$ satisfies D (e.g., $\|n_q, o\| \leq 10$ miles); and (2) its attributes denoted by $o.a$ satisfy A (e.g., restaurant $o.cuisine =$ 'Italian'). As D and A of each LDSQ are orthogonal, they are handled independently. To facilitate our discussion, we list the major notations in Table 1.

TABLE 1
Notations

Notations	Description
$\mathcal{N} = (N, E)$	a network $\mathcal{N}$ with nodes N and edges E
$\mathcal{O}$	a set of objects
$\mathcal{R} = (N_\mathcal{R}, E_\mathcal{R}, B_\mathcal{R})$	an Rnet $\mathcal{R}$ with nodes $N_\mathcal{R}$, edges $E_\mathcal{R}$ and border nodes $B_\mathcal{R}$ (see Def. 1)
$L(n, n')$	a subset of objects located on edge (n, n')
$L(\mathcal{R})$	an object abstract (i.e., objects located inside an Rnet $\mathcal{R}$) (see Def. 2)
$\|\|n, n'\|\|$	network distance between n and n'
$\delta(o, n)$	distance between an object o to a node n
$S(b, b')$	a shortcut S between border nodes b and b' (see Def. 3)

3.2 Rnet, Shortcut and Object Abstract

We first introduce an important notion in our ROAD, i.e., *regional subnetworks* (Rnets), as in Definition 1. Each Rnet encloses a set of edges bounded by a set of *border nodes*, which serve as the entrances and exits of the Rnet.

Definition 1 (Rnet). *In $\mathcal{N} = (N, E)$, an Rnet $\mathcal{R} = (N_\mathcal{R}, E_\mathcal{R}, B_\mathcal{R})$ captures 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. $N_\mathcal{R} = \{n \mid (n, n') \in E_\mathcal{R} \vee (n', n) \in E_\mathcal{R}\}$, *and*
3. $B_\mathcal{R} = N_\mathcal{R} \cap \{n \mid (n, n') \in E' \vee (n', n) \in E'\}$, *where* $E' = E - E_\mathcal{R}$.

As in the previous example depicted in Fig. 1, when a search reaches n_1 covered by an Rnet R, we need 1) a hint about what objects are in R to decide if a detailed examination of R is needed, and 2) an artifact at n_1 connected to the other border nodes of R (e.g., n_3) to allow the search to bypass R and to continue the traversal thereafter. Accordingly, we define *object abstracts* and *shortcuts* as in Definition 2 and Definition 3, respectively. Here, an object abstract is the summary of objects located on enclosed edges, and a shortcut is a shortest path between two border nodes.

Definition 2 (Object Abstract). *The object abstract of an Rnet $\mathcal{R}$, $L(\mathcal{R})$ indicates all the objects residing on edges in $E_\mathcal{R}$, i.e., $L(\mathcal{R}) = \bigcup_{e \in E_\mathcal{R}} L(e)$.*

Definition 3 (Shortcut). *The shortcut, $S(b, b')$, between two border nodes b and b' ($\in B_\mathcal{R}$) of an Rnet $\mathcal{R}$ is the precomputed shortest path $SP(b, b')$ and its distance is $\|b, b'\|$. Notice that the edges that contribute to $SP(b, b')$ might not necessarily be included in $E_\mathcal{R}$.*

3.3 Rnet Hierarchy

In ROAD, we structure a road network as a hierarchy of Rnets, where large Rnets at the upper levels enclose small Rnets at lower levels. At each level, a network can be viewed as a layer of interconnected Rnets. This design benefits search ranges of different sizes. To derive an Rnet hierarchy, we first treat the entire road network as a single

1. Objects at nodes (i.e., road intersections) can be treated as they are located at the end of the corresponding edges.
2. This assumption can be easily relaxed to handle LDSQs issued on edges via searching objects on the edges followed by running the queries at the ending nodes of the corresponding edges.

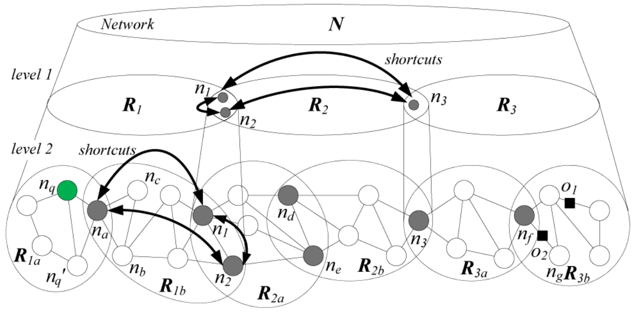


Fig. 4. Example Rnet hierarchy.

level-0 Rnet that has no border node, and divide it into p_1 partitioned Rnets. Definition 4 formally defines Rnet partitioning. 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. Then, at any level x ($\in [0, l]$) where $l+1$ is the number of levels, the network is fully covered by $\prod_{i=1}^{x} p_i$ interconnected Rnets. As a whole, there are $\sum_{h=0}^{l} \prod_{i=1}^{h} p_i$ Rnets.

Definition 4 (Rnet partitioning). *Partitioning of an Rnet* $\mathcal{R} = (N, E, B)$ *forms p child Rnets,* $\mathcal{R}_1, \mathcal{R}_2, \ldots \mathcal{R}_p$ *where* $p > 1$ *and* $\mathcal{R}_i = (N_i, E_i, B_i)$. *Here,* $N = \bigcup_{1 \le i \le p} N_i$, $E = \bigcup_{1 \le i \le p} E_i$, $B \subseteq \bigcup_{1 \le i \le p} B_i$. *Also, the following three conditions must hold.*

1. *Edges of all children Rnets are disjoint, i.e.,* $\forall 1 \le i, j \le p, i \ne j \Rightarrow E_i \cap E_j = \emptyset$.
2. *Nodes in an Rnet are connected by edges in the same Rnet, i.e.,* $\forall_i \forall_{(n,n') \in E_i}\ n \in N_i \wedge n' \in N_i$.
3. *Border nodes in an Rnet are nodes, common to its parent Rnet border nodes or nodes in its sibling Rnets, i.e.,* $B_i = N_i \cap (B \cup \bigcup_{j \in ([1,p]-\{i\})} N_j)$.

As depicted in Fig. 4, a network N is first partitioned into three Rnets, R_1, R_2, and R_3, each of which is then partitioned into two smaller Rnets, R_{ia} and R_{ib}, $i \in \{1, 2, 3\}$. Here, R_1, R_2, and R_3 form the level-1 Rnets and R_{1a}, R_{1b}, R_{2a}, R_{2b}, R_{3a}, and R_{3b} form the level-2 Rnets. Common to both R_2 and R_3, n_3 is a border node between R_2 and R_3. And, n_3 is shared by both R_{2b} and R_{3a} and thus is also a border node of R_{2b} and R_{3a}.

As the goal of ROAD is to provide a general-purpose search platform for any added-on spatial objects and various LDSQs, we adopt a network partitioning that can generate equal-sized Rnets and the smallest number of border nodes, which, in turn, minimizes the number of shortcuts formed. This network partitioning problem is, however, known NP-complete. Heuristically, we adopt both geometric approach [15] and Kernighan-Lin algorithm (KL algorithm) [16]. The geometric approach first coarsely partitions a network into two by dividing edges spatially. KL algorithm is then used to refine the partitioned Rnets by exchanging edges between them until no further exchanges can 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 for each level i. This network partitioning approach is also used in [11] as it provides a quick network partitioning in a reasonably good quality. 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.

After an Rnet hierarchy is formed, object abstracts and shortcuts are constructed in a bottom-up fashion. As edges in Rnets are fully covered by their parent Rnet according to Definition 4, object abstracts of an Rnet can be constructed directly from their child Rnets. Lemma 1 states this property.

Lemma 1. *The object abstract of a parent Rnet* $\mathcal{R}$ *fully covers those of all its child Rnets* $\mathcal{R}_1, \ldots \mathcal{R}_p$, *i.e.,* $L(\mathcal{R}) = \bigcup_{1 \le i \le p} L(\mathcal{R}_i)$.

The shortcut from a border node to another can be determined by Dijkstra's algorithm [10]. Here, we identify several unique properties of the shortcuts. First, the shortcuts in level-i Rnets can be calculated based on those in level-($i+1$) Rnets. Thus, shortcuts in high-level Rnets can be presented as a sequence of shortcuts in Rnets of the immediate low-level. Second, determined shortcuts in Rnets can be used to compute other shortcuts of Rnets in the same level. Third, to alleviate the storage cost for shortcuts, those shortcuts $S(b, b')$ that can be regenerated by other shortcuts in the same Rnets can be ignored. Please refer to [17] for the detailed statements and proofs of these properties and they are skipped for space saving.

3.4 Route Overlay and Association Directory

Based on Definition 4 that border nodes in an Rnet are always the border nodes in some of its child Rnets, we derive a novel index structure, namely *Route Overlay*, which naturally flattens a hierarchical network into a plain structure to facilitate search space expansion over a network.

In Route Overlay, nodes are indexed by a 1D index with unique node IDs as search keys. In our implementation, we use B^+-tree.[3] Each leaf entry of the B^+-tree points to one node, together with a *shortcut tree*, i.e., a specialized tree-based index structure that organizes shortcuts and edges to facilitate search traversals. If a given node n is a border node of an Rnet, all the shortcuts from n to other border nodes belonging to the same Rnet are captured by nonleaf entries of n's shortcut tree. Also, shortcut trees preserve the Rnets hierarchy by placing shortcuts of parent Rnets right above those belonging to the corresponding child Rnets. On the other hand, a leaf entry stores all the physical edges to its neighboring nodes. This shortcut tree structure generalizes the adjacency lists in many conventional network storage schemes.

Fig. 5 shows the Route Overlay for our network presented in Fig. 4. Take a nonborder node n_q as an example. Its shortcut tree has only one leaf entry that contains edges to n_q's neighboring nodes, e.g., n_a and n'_q. Conversely, for n_a (a border node of Rnets R_{1a} and R_{1b}), its shortcut tree has two levels. The root entry contains shortcuts of Rnets R_{1a} and R_{1b}. Since R_{1a} has only one border node, no shortcuts for R_{1a} are kept. Below it are two physical edges to n_q and n'_q. Besides, shortcuts to n_1 and n_2 are kept for R_{1b} at the top level. Then, physical edges to n_b and n_c at R_{1b} are placed at the bottom. As will be explained later, this shortcut tree structure can effectively facilitate search processes.

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

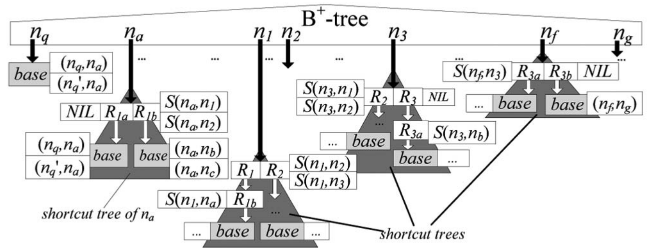


Fig. 5. Route Overlay.

Next, we propose *Association Directory*, an efficient object lookup mechanism in ROAD. It is also based on a one-dimensional index, e.g., B+-tree, with unique node IDs or Rnet IDs as the search key. Associated with node n (n') are objects o in $L(n, n')$ together with their 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, several techniques such as bloom filter [18] and signature [19] can be used to represent an object abstract with smaller 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 the Association Directory, no object is implied for the node (Rnet).

Fig. 6 depicts an Association Directory for objects o_1 and o_2 in our example. Object o_1 on edge (n_f, n_g) is pointed by the nodes n_f and n_g and it is associated with its corresponding distances to the nodes. Moreover, both Rnet R_{3b} and its parent Rnet R_3 that contain objects o_1 and o_2 are associated with $\{o_1, o_2\}$ in the Association Directory. Subject to applications, objects can be placed into one or multiple Association Directory on top of the same road network.

4 Single-Source LDSQ Algorithms

This section presents search algorithms to support single-source LDSQs that include range queries and kNN queries. A range query finds objects of interest within a specified distance range to the query point n_q (e.g., Q1 in Section 1). A kNN query returns the k objects of interest closest to n_q (e.g., Q2 in Section 1).

We first discuss the evaluation of kNN query. To illustrate the basic idea of our approach, we use a simple network that consists of a chain of nodes in Fig. 7. In the figure, the network is partitioned into three Rnets and each of them is further divided into two smaller Rnets. 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. 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 traversals are shown as a sequence of annotated arrows (as arranged vertically) in the figure.

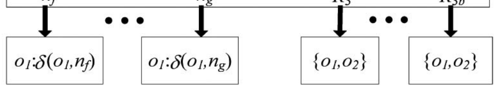


Fig. 6. Association Directory.

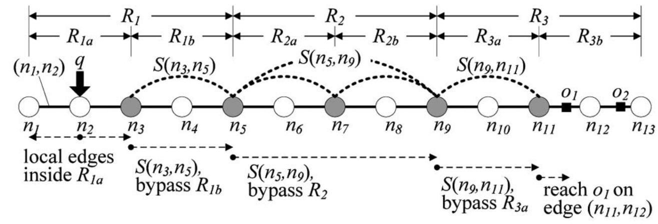


Fig. 7. Example single-source 1NN query.

Since n_3 is the border node of Rnets R_{1a} and R_{1b}, the object abstract associated with R_{1b} is checked. As the object abstract indicates no object of interest within R_{1b}, the shortcut $S(n_3, n_5)$ is taken to bypass R_{1b}. Next, the search resumes at node n_5, i.e., the border node of Rnets R_2 and R_{2a}. Again, the object abstract shows an empty result, so the shortcut $S(n_5, n_9)$ is taken to bypass R_2 and reach n_9. Similarly, the search follows the shortcut $S(n_9, n_{11})$ to reach n_{11}. Now, since R_{3b} contains objects, a traversal within R_{3b} is needed. The traversal follows the physical edges to find object o_1. Here, we can see the search only takes three jumps from n_3 to n_{11}, that significantly saves the traversal cost, compared with the gradual network expansion.

Following the basic logic of network expansion, Algorithm k**NNSearch** incorporates shortcuts in Route Overlay and object abstracts in Association Directory for speedup. It repeatedly expands the search in the network from n_q by visiting the closest unexplored node to guarantee that the first k qualified objects visited are the kNN objects. We maintain a priority queue P to sort pending entries in the nondescending distance order from n_q. Each entry (ϵ, d) in P records a node or an object (ϵ) and its distance (d) from n_q.

The algorithm takes the Route Overlay (RO), Associate Directory (AD), a query node (n_q), and a desired number of NNs (k) as inputs, and defaults all nodes and objects as "*unvisited*." To start with, P is initialized with $(n_q, 0)$. Then, the algorithm repeatedly examines the head entry (ϵ, d) from P until k answer objects are retrieved or the network is completely traversed. Since nodes and objects could be reached more than once via different paths, ϵ already marked with "*visited*" is discarded. Otherwise, the examination of ϵ begins. If ϵ refers to a node, two tasks need to be performed. AD is first accessed for objects o associated with the node ϵ, which will be put to P as $(o, d + \delta(o, \epsilon))$ for later examination. Next, Algorithm **ChoosePath** is invoked to decide subsequent nodes to continue the network expansion from ϵ that will be discussed next. If ϵ is an object, it is collected into a result set Res. After examination, ϵ is marked "*visited*." Finally, the answer objects are outputted and the search completes.

With shortcut trees organizing shortcuts and edges in accordance with the Rnet hierarchy, Algorithm **ChoosePath** can quickly identify appropriate shortcuts and edges to expand the search range from a node n. In brief, it examines the shortcut tree of the node n in the depth-first traversal order. If n is a border node, the shortcut tree must have multiple levels. For every nonleaf level, an Rnet R is checked against Association Directory. If no object of interest is found, R and all its child Rnets are bypassed. The border nodes reached by the shortcuts in that level are enqueued to P. Otherwise (i.e., R contains objects of interest), the lookup goes down to the next lower level to examine R's child Rnets in a similar fashion. When the search reaches the leaf level, all neighboring nodes connected by physical edges are collected. If n is a nonborder node, its shortcut tree contains only one level (i.e., physical edges) and all the corresponding neighboring nodes are put into P. Please refer to [17] for the pseudocodes of Algorithm k**NNSearch** and Algorithm **ChoosePath**.

Algorithm **RangeSearch** that supports range queries resembles Algorithm k**NNSearch** except that the search ends when a portion of the network within the distance bound (as specified by a query) is completely traversed. All visited objects are the answer objects. To save space, we omit the discussion of this approach.

5 Multisource LDSQ Algorithms

A multisource LDSQ finds objects with respect to m query nodes, i.e., $n_{q1}, \ldots n_{q_m}$ ($m > 1$). A multisource kNN query finds k objects whose maximum distances from all query nodes (i.e., $\max(\{d_i, i \in [1,m]\})$) are the minimum, where $d_i = \|n_{q_i}, o\|$. Q3 discussed in Section 1 is an example. A multisource range query retrieves all objects of interest within distance range r with respect to all query nodes (i.e., $\forall_{i \in [1,m]} d_i \leq r$). In the literature, [8] suggests to process multisource LDSQs as euclidean distance bound approach that first estimates candidate objects based on their euclidean distances and eliminates false candidates based on their network distances. This approach, however, covers a larger search space and incurs longer processing times than necessary.

5.1 Concurrent Network Expansion

ROAD adopts a *concurrent approach* that expands a search space from all query nodes through a best-first strategy, until all result objects are obtained or the search range is completely traversed. Hereafter, we call an expansion started from each query node, n_{q_i}, as a *subquery*, q_i. We first discuss our algorithm for multisource kNN query. According to Lemma 2, the k first visited objects are guaranteed to be the answer objects.

Lemma 2. *With concurrent expansion, the first k objects visited by all subqueries are the kNN objects.*

Proof. We prove this lemma by contradiction. Assume there are k objects, i.e., $o_1, \ldots o_k$, just visited by all the subqueries. We assume that o_i ($i \in [1,k]$) is not one of kNN as there will be another object o_j ($j \notin [1,k]$) such that o_j's maximum distance to query points is smaller than that of o_i. Suppose, the last subqueries visiting o_i and o_j are q_a and q_b, respectively. Since q_a visited o_i before q_b visits o_j, the distance of o_i from n_{q_a}, i.e., its maximum distance (see Lemma 3) should not be greater than that of o_j from n_{q_b}. This contradicts the assumption. Hence, o_j should not be a part of kNN query result. □

Lemma 3. *With concurrent expansion, the maximum distance of an object to all query nodes is determined by the last subquery that visits it.*

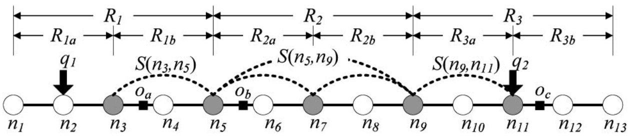


Fig. 8. Example two-source 1NN query.

Proof. The distances of an object, o, to all m query nodes are $d_1, \ldots d_m$. By concurrent expansion, $d_i \leq d_j$ if the subquery q_i reaches o earlier than q_j. Thus, if q_j is the last subquery visiting o, d_j should be the largest. □

5.2 Rnet Visited Set and Border Node Visited Set

Fig. 8 shows a two-source 1NN query based on the previous example network but with different objects. The result object is o_b on (n_5, n_6). Two subqueries, q_1 and q_2 proceed from n_2 and n_{11}, respectively. However, as shown in the figure, before they reach o_b, q_1 (q_2) may enter Rnet R_{1b} (R_{3b}) for an object o_a (o_c) based on the idea of single-source LDSQs. While o_a and o_c are not the answer objects, the traversals inside those Rnets are a waste.

Here we observe that the guidance of traversals by object abstracts alone is *inadequate* for multisource LDSQs. Instead, an Rnet is worth exploring only if it contains objects of interest and it is reached by all subqueries. In our example, R_2 is the first Rnet visited by both q_1 and q_2. According to the latter condition, we introduce two additional data structures, namely, *Rnet visited set* (RV), and *border node visited set* (BV). An RV is maintained to keep track of subqueries that have visited or reached a given Rnet. Referring to our example, we keep q_1 with R_1 and R_{1_a} and q_2 with R_3 and R_{3_a} in RV. Now, as no Rnets are visited by both q_1 and q_2, no detailed traversals within any Rnets are needed.

Conversely, Rnets R would be visited by all m subqueries at different times. Some $m-1$ subqueries have already bypassed R and need to resume their traversals in R when R is found to be reached by all m subqueries. This traversal resumption is called *backtracking*. To enable such backtracking, a BV keeps track of the border nodes of R via which each subquery q_i bypassed R. Thus, whenever a backtracking of an Rnet is triggered, subqueries resume their traversals at those border nodes.

5.3 Search Algorithm

Our Algorithm **MultiSource$k$NNSearch** (as outlined in Fig. 9) exploits all the above-discussed techniques for multisource kNN query. The algorithm maintains a priority queue P to sort pending entries in a nondecreasing distance order from respective query nodes. Every entry (ϵ, d, q_i) in P records a node or an object (ϵ), its distance from n_{q_i} (d) and the respective subquery (q_i). We also keep an Rnet visited list (RV) and a border node visited set (BV). An entry (R, q_i) in RV indicates that R has been visited by q_i. An entry (R, b, d, q_i) in BV records that a subquery q_i has reached R via the border node b, and $d = \|b, n_{q_i}\|$. Similar to RV, an object visited set, OV, records which objects have been visited by which subqueries.

Initially, all nodes and objects are marked "*unvisited by q_i*" ($i \in [1,m]$) and all the query nodes are enqueued as

Algorithm MultiSource$k$NNSearch(RO, AD, $\{n_{q_1}, \cdots n_{q_m}\}$, k)
Input. Route Overlay (RO), Association Directory (AD),
A set of m query nodes ($\{n_{q_1}, \cdots n_{q_m}\}$), #NNs ($k$)
Local. Priority Queue (P), Rnet Visited Set (RV),
Object Visited Set (OV), Border Visited Set (BV);
Output. Result set (Res)
Begin
1. **foreach** $n_{q_i} \in \{n_{q_1}, \cdots n_{q_m}\}$ **do**
2. **enqueue**(P, $(n_{q_i}, 0, q_i)$);
3. **while** (P is not empty **AND** $|Res| < k$) **do**
4. $(\epsilon, d, q_i) \leftarrow$ **dequeue**(P);
5. **if** (ϵ is marked "visited by q_i") **then goto** 3;
6. **if** (ϵ is a node) **then**
7. $O \leftarrow$ SearchObject(AD,ϵ); // *find objects with* ϵ
8. **foreach** $(o, \delta(o, \epsilon)) \in O$ **do**
9. **enqueue**(P, $(o, d + \delta(o, \epsilon), q_i)$);
10. **foreach** Rnet R containing ϵ, add (R, q_i) to RV;
11. **foreach** Rnet R visited by all subqueries **then**
12. get (R, ϵ', d', q') in BV, ϵ' is the border nodes of R;
13. **enqueue**(P, (ϵ', d', q'));
14. mark ϵ' "unvisited by q'"; // *this allows revisit to* ϵ'.
15. MultiSourceChoosePath(RO, AD, ϵ, q_i, d, P, RV, BV);
16. **else** // ϵ *is an object.*
17. adding (ϵ, q_i) to OV;
18. **if** (ϵ is visited by all subqueries) **then**
19. $Res \leftarrow Res \cup \{\epsilon\}$;
20. mark ϵ "visited by q_i";
21. **output** Res;
End.

Fig. 9. Algorithm MultiSource$k$NNSearch.

entries into P (lines 1-2). Then, the search repeatedly evaluates the head entry (ϵ, d, q_i) from P until k answer objects are retrieved or the entire network is completely traversed (lines 3-20). If ϵ has been visited by the same subquery q_i, the evaluation on ϵ is skipped (line 5). Otherwise, a detailed examination begins. If ϵ is a node, three tasks need to be performed. First, its associated objects are fetched from Association Directory and enqueued to P for later examination (lines 7-9). Second, it records all Rnets that ϵ belongs to in RV (line 10), followed by checking whether the visit of current node triggers the backtracking of any Rnet R (lines 11-14). If so, the traversals of R for all the subqueries are resumed at the border nodes b of (R, b, d', q') maintained in BV (lines 12-13). Further, we

Algorithm MultiSourceChoosePath(RO, AD, n, q, d, P, RV, BV)
Input. Route Overlay (RO), Association Directory (AD),
a node (n), a subquery (q), distance from n_q (d),
Priority Queue (P), Rnet Visited Set (RV),
Border Visited Set (BV)
Local. Stack (S);
Begin
1. $T \leftarrow$ LoadShortcutTree(RO,n);
2. **push**(S,$T.root$);
3. **while** (S is not empty) **do**
4. $s \leftarrow$ **pop**(S);
5. **if** (s is not leaf) **then**
6. **foreach** ($R \in s$) **do**
7. **if** (SearchObject(AD,R) has no object **OR**
R is not visited by all subqueries in RV) **then**
8. add shortcuts (n, b') (i.e., $(b', d + ||n, b'||, q)$) to P;
9. add (R,n, d, q) to BV;
10. **else**
11. **push** all s's children to S;
12. **else**
13. **add** all edges (n, n') in s as $((n', d + |n, n'|, q))$ to P;
End.

Fig. 10. Algorithm MultiSourceChoosePath.

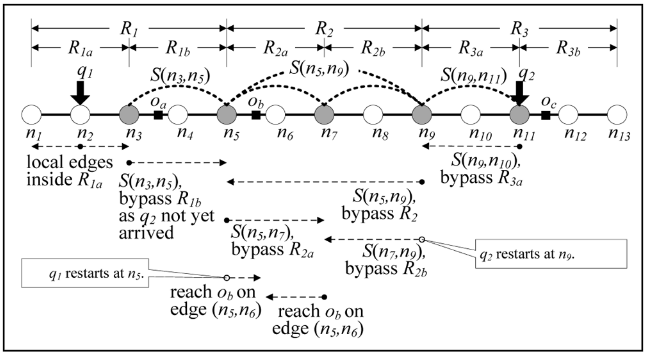


Fig. 11. Example two-source 1NN query (in detail).

restore them to "*unvisited by* q'" to allow the revisit of the nodes (line 14). Third, the search range is expanded from the current node ϵ with the aid of Algorithm **MultiSourceChoosePath** (line 15).

When ϵ refers to an object, we update OV to indicate that object ϵ has been accessed by the current subquery q_i (line 17). If it has been visited by all the subqueries, ϵ is inserted to the result set Res (lines 18-19) which is outputted after the search completes (line 21). At the end of each iteration, ϵ is marked "*visited by* q_i" (line 20).

Algorithm **MultiSourceChoosePath** (as outlined in Fig. 10) visits node n's shortcut tree in a depth-first order and identifies appropriate shortcuts and edges to expand the search range. If an Rnet R in n's shortcut tree 1) contains no object of interest as indicated by AD, or 2) has not been visited by all the subqueries as tracked in RV, the search keeps n and its distance from n_{q_i} in BV and takes the corresponding shortcuts to bypass R (lines 7-9). Otherwise, the search continues to examine R's child Rnets in the shortcut tree (lines 10-11). When the leaf level is reached, all nodes connected by physical edges are collected.

To illustrate Algorithm **MultiSource$k$NNSearch**, we revisit the example as shown in Fig. 8. First, subqueries, q_1 and q_2, expand from nodes n_2 and n_{11}, respectively. Each expansion is shown as an annotated arrow in Fig. 11. At n_2, q_1 records its visit by adding (R_{1a}, q_1) and (R_1, q_1) to RV. Similarly, at n_{11}, q_2 puts (R_{3a}, q_2), (R_{3b}, q_2), and (R_3, q_2) to RV and puts $(R_{3a}, n_{11}, 0, q_2)$ and $(R_{3b}, n_{11}, 0, q_2)$ to BV. Through local edges, q_1 reaches n_3, a border node of R_{1b}, and updates BV and RV.[4] As R_{1b} is not yet visited by all the subqueries as informed by RV, q_1 bypasses it although it contains object o_a. Then, q_2 skips R_{3a} and R_{3b} since they have no object and are not visited by all the subqueries, respectively.

Then, q_2 continues the traversal from n_9 to n_5, but q_1, has not yet reached R_2. Here, $(n_5, \|n_5, n_{11}\|, q_2)$ is pending in P for next access. Thereafter, q_1 reaches n_5, a border node of R_2. It learns from RV and AD that R_2 is visited by both q_1 and q_2 and it contains objects. Consequently, R_2 needs detailed examination and its child Rnets are visited. As R_{2a} is not visited by q_2, the shortcut is taken to bypass it and n_7 is enqueued to P for later evaluation. Meanwhile, q_2 resumes its traversal of R_2 at the border node n_9, as indicated by BV. It takes the shortcut to n_7 to visit R_{2a}. As R_{2a} is already visited by q_1, q_2 navigates inside R_{2a} and q_1

4. For brevity, we omit the descriptions of updating BV, RV, and OV hereafter.

resumes the traversal at n_5. Finally, the answer object o_b is reached by both subqueries and the search finishes.

Similar to Algorithm **MultiSource$k$NNSearch**, Algorithm **MultiSourceRangeSearch** for multisource range query traverses a network when all unexamined entries that include nodes or objects in the queue are beyond a specified distance range. Finally, the result objects are those visited by all the subqueries. We skip the algorithm details due to limited space.

6 ROAD Framework Maintenance

In this section, we present the ROAD maintenance in presence of network and object updates.

6.1 Object Update

Object changes are handled in Association Directory only. To insert an object on an edge (n, n') in Rnet R, we associate the object to nodes n and n' and update the object abstracts of R and R's ancestor Rnets in an Association Directory. For object deletion, we simply remove the association of the objects from corresponding nodes and from the object abstracts of corresponding Rnets in an Association Directory. For the changes of object attributes, we update the object abstract associated with nodes and Rnets.

6.2 Network Update

Road condition and road network structure change over time. Rather than immediately rebuilding a Route Overlay upon changes, which is expensive, we develop several techniques to incrementally update Route Overlay for *edge distance changes*, and *network structure changes*.

6.2.1 Change of Edge Distance

When the distance of an edge changes (increases or decreases), some shortcuts have to be updated. To save unnecessary shortcut recomputations, ROAD adopts a *filtering-and-refreshing* approach. In the "filtering" phase, shortcuts possibly affected by an edge change are identified. Only the identified shortcuts are updated in the "refreshing" phase. According to Rnet properties, described in Section 3 and in [17], 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. Here, we only explain how to recompute shortcuts in the bottom level. The same idea can be applied to upper levels. Similarly, 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, the distance $|n, n'|$ of an edge is changed from d to d', detailed update procedures are as follows:

Edge distance increased (i.e., $d < d'$). When an edge (n, n') in an Rnet R increases its distance from d to d', only those shortcuts that cover (n, n') might become invalid and need refreshed. In the filtering phase, 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 both ending nodes n and n' to the border nodes in R and identifying shortcuts whose distances are equal to the path passing through (n, n'). In the refreshing phase, all the identified shortcuts are re-evaluated. Updates, if any, are then propagated to the upper level.

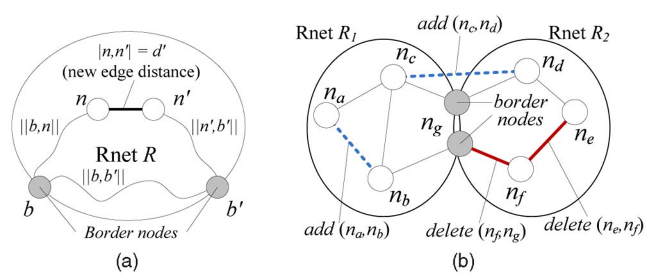


Fig. 12. Network changes. (a) Edge distance decrease. (b) Edge addition and deletion.

Edge distance decreased (i.e., $d > d'$). When an edge (n, n') in an Rnet R decreases its distance from d to d', it may contribute to paths shorter than some existing shortcuts. 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' 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 Fig. 12a. If a new formed path between b and b' via (n, n') has its distance shorter than existing $S(b, b')$, i.e., $||b, n|| + d' + ||n', b'|| < ||b, b'||$, $S(b, b')$ is said to be affected and needs to be updated. In the refreshing phase, all affected paths are replaced by new paths passing through (n, n'). Again, updates are propagated to the upper level if shortcuts are updated.

6.2.2 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. Here, 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 when necessary.

Addition of a new edge. A newly added edge (n, n') directly connects nodes n and n', assuming that n and n' belong to Rnets $\mathcal{R}$ and $\mathcal{R}'$, respectively. Two possible cases, 1) $\mathcal{R} = \mathcal{R}'$ and 2) $\mathcal{R} \neq \mathcal{R}'$, are handled as follows:

- **Case 1**: $\mathcal{R} = \mathcal{R}'$. Adding an edge connecting two nodes (e.g., (n_a, n_b) in Fig. 12b) can be treated as changing the edge distance from infinity to the current distance. Edge distance update mechanism discussed previously can be applied here. Accordingly, the Route Overlay stores the new edge.
- **Case 2**: $\mathcal{R} \neq \mathcal{R}'$. 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 Fig. 12b, the introduction of (n_c, n_d) to R_1 gets n_d promoted to a border node. Also, the new edge (n, n') might affect some shortcuts. The update approach for the change of edge distance is 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 Fig. 12b. 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 end node n of a deleted edge (n, n') is a border node. If all n's edges are within one Rnet after deleting (n, n'), n is no longer a border node. As shown in Fig. 12b, after deleting (n_f, n_g), n_g becomes a nonborder node. Then, the shortcut trees of n and existing border nodes in related Rnets in Route Overlay are updated.

7 Performance Analysis

In this section, we provide a theoretical analysis on the performance of ROAD, in terms of 1) storage cost, 2) construction time, and 3) query processing cost. Since the cost for maintaining an Association Directory is much smaller than that for Route Overlay, we focus our analysis only on the latter. To facilitate our analysis, we make an assumption that is commonly used in the literature [6], [13]. We assume the road network $\mathcal{N}$ is in form of a 2D Manhattan network in an square area A consisting of only horizontal and vertical edges, and $\mathcal{N}$ is formulated as an $(l+1)$-level Rnet hierarchy. At each nonbottom level i ($i < l$), each Rnet is partitioned into p equal-sized child Rnets. At any level i ($0 \leq i \leq l$), there are p^i Rnets.

7.1 Storage Cost for Shortcuts

First, we examine the storage cost for keeping all the shortcuts in an Rnet hierarchy. Assume Rnets of a given level i are equally sized. Each Rnet, R, covers an expected area of A/p^i. The number of border nodes for R is, therefore, approximately proportional to the length of the perimeter of R, i.e., $(4 \cdot \sqrt{A/p^i})$. Accordingly, the number of shortcuts in R is $(16 \cdot A/p^i)$. As such, at level i, p^i Rnets result in $(16 \cdot A)$ shortcuts.

As we have explained in Section 3, in an Rnet hierarchy, the shortcuts at the bottom level (i.e., level l) cover physical edges, whereas those at upper levels cover shortcuts in immediately lower level Rnets. Hence, the cost for shortcuts at level l is different from that of other levels. Consider that at the bottom level, shortcuts are simply straight paths from one side to another in an Rnet, R. The number of physical edges, in this case, is the perimeter of R, i.e., $\sqrt{A/p^l}$. On the other hand, shortcuts at upper levels cover those in child Rnets. While there are p child Rnets on the 2D area, a shortcut covers those across a row (or column) of $\sqrt{p}$ child Rnets. Thus, the storage cost for a shortcut is $\sqrt{p}$. To sum up, we express the storage cost C_{sto} for all shortcuts in

$$C_{sto} = 16 \cdot A \cdot \left(\sqrt{\frac{A}{p^l}} + \sum_{i=1}^{l-1} \sqrt{p} \right) = O\left(A \cdot \left(\sqrt{\frac{A}{p^l}} + l \cdot \sqrt{p} \right) \right). \tag{1}$$

As the storage overhead of shortcut trees highly depends on the storage cost of shortcuts, we can see from (1) that given a fixed p^l for certain desired finest Rnet sizes, a smaller l together with a correspondingly larger p (as in the term $l \cdot \sqrt{p}$) can reduce the storage overhead incurred by shortcuts. This observation is validated through experiments with real datasets presented in Section 8. Though it is used to estimate the storage cost, A can be reexpressed in terms of the number of nodes $|N|$ as nodes are assumed to be uniformly distributed within A. In the following discussion, we keep using A to formulate the cost and performance for simplicity.

7.2 Construction Time for Shortcuts

Next, we estimate the time for shortcut construction. We assume that Dijkstra's algorithm is used, which has a run time complexity of $O(V \cdot logV)$ with V denoting the number of nodes traversed. At level l (i.e., the bottom level), each Rnet R has A/p^l nodes among which $(4 \cdot \sqrt{A/p^l})$ are border nodes. By Dijkstra's algorithm, the time complexity to compute all shortcuts in R is $O(\sqrt{A/p^l} \cdot (A/p^l) log(A/p^l))$. Hence, the time complexity for deciding all that shortcuts at level l having p^l Rnets is $O(A\sqrt{A/p^l} \cdot log(A/p^l))$.

On the other hand, the time complexity for computing a shortcut in an Rnet, R, at upper level i is $O(\sqrt{A/p^{i-1}} \cdot log\sqrt{A/p^{i-1}})$, because there are $O(\sqrt{A/p^{i-1}})$ border nodes in R's child Rnets. Again, there are $(4 \cdot \sqrt{A/p^i})$ border nodes in R. Hence, the time complexity for shortcut computation at level i with p^i Rnets is $O(A \cdot \sqrt{p} \cdot log\sqrt{A/p^{i-1}})$. In summary, (2) formulates the time complexity C_{spt} for shortcut computation:

$$\begin{aligned} C_{spt} &= O\left(\frac{A\sqrt{A}}{\sqrt{p^l}} \cdot log\frac{A}{p^l} \right) + \sum_{i=1}^{l-1} O\left(A \cdot \sqrt{p} \cdot log\sqrt{\frac{A}{p^{i-1}}} \right) \\ &= O\left(\frac{A\sqrt{A}}{\sqrt{p^l}} \cdot log\frac{A}{p^l} + A \cdot \sqrt{p} \cdot \sum_{i=1}^{l-1} log\sqrt{\frac{A}{p^{i-1}}} \right). \end{aligned} \tag{2}$$

As indicated in (2), the computation time for shortcuts in the bottom level is the predominant factor to the total Rnet hierarchy construction time, consistent with what we observed in our implementation.

7.3 LDSQ Processing Time

Further, we estimate the processing time for LDSQs. Here, we only consider single-source LDSQs. In ROAD, a query typically involves two phases, namely, 1) an *expansion phase* that expands a search range from a local smallest Rnet where a query is issued to larger Rnets that cover target objects, and 2) a *drill-down phase* that narrows down the search from large Rnets to the smallest Rnets that contain required objects. Assume that search expands up to Rnets in level t, expecting that some level-t Rnets cover an object. By means of Dijkstra's algorithm alike expansions, the processing time in terms of node visits in expansion phase, therefore, is $O(\frac{A}{p^l} log\frac{A}{p^l} + \sum_{i=t}^{l-1} \sqrt{\frac{A}{p^{i-1}}} log\sqrt{\frac{A}{p^{i-1}}})$. Here, the first term $\frac{A}{p^l} log\frac{A}{p^l}$ denotes the time complexity for expanding physical edges and the second term does that for shortcuts in upper levels. Symmetric to an expansion, drill down phrase results in the same time complexity.

Next, we estimate t, the level of Rnet hierarchy that a search space needs to cover. Let $|\mathcal{O}|$ be the number of objects

TABLE 2
Evaluation Parameters

Parameter	Value	Default
Network ($\mathcal{N}$) (nodes,edges)	CA (21,048, 21,693) NA (175,813, 179,179) SF (174,956, 223,001) PRS (327,402, 451,760)	NA
# objects ($\lvert\mathcal{O}\rvert$)	100, 1000, 10,000, 100,000,	10,000
Distribution	10 - 1,000 clusters, Uniform	100 clusters
Partition factor (p)	2, 4, 16	4
# levels (l)	2, 4, 8	4 for CA 8 for others
Query	single-source LDSQs, multi-source LDSQs	single-source kNN
# NNs (k)	1, 10, 100, 1,000	10
Search range (r)	0.1 of net. diameter	0.1
# sources (m)	2, 4, 6	2

evenly distributed in a network. Consider an NN query. We can expect that at the lowest level t, each Rnet with an area $\frac{A}{p^t}$ covers an NN object for a query. Hence, as $p^t = |\mathcal{O}|$, $t = log_p|\mathcal{O}|$. Putting all of them together, we obtain the time complexity C_{NN} in performing a single-source NN search, as in

$$C_{NN} = O\left(\frac{A}{p^l} log \frac{A}{p^l} + \sum_{i=log_p|\mathcal{O}|}^{l-1} \sqrt{\frac{A}{p^{i-1}}} log \sqrt{\frac{A}{p^{i-1}}}\right). \quad (3)$$

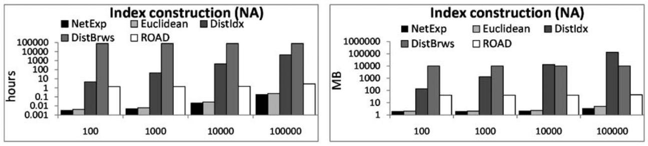
From (3), we can see that if p^l is fixed, the first term is invariant but the second term will increase for l while p is reduced accordingly. We validated this observation in our experiments. Also, when $|\mathcal{O}|$ is very large, we can anticipate that the second term will become very small, thus making NN searches very efficient. Meanwhile, for a single-source range query with a searched area of a, t is determined as $log_p \frac{A}{a}$ such as the area of an Rnet at level t, i.e., A/p^t sufficiently covers a.

8 Performance Evaluation

This section evaluates our proposed ROAD framework in terms of both indexing overhead and query performance. We applied ROAD (labeled as ROAD, hereafter) on four real road networks, namely, CA, NA, SF, and PRS.[5] CA and NA consist of highways in California and North America, respectively. SF and PRS correspond to streets and roads in San Francisco and Paris, respectively. We generated 100 to 100,000 objects following either uniform distribution or clustered distribution over these road networks. To simulate clustered distribution, we select a set of nodes as cluster centroids and distribute equal numbers of objects within 10 nodes around them. Table 2 lists all the evaluation parameters, their values and defaults used in the experiments.

We also implemented network expansion [7], euclidean distance bound approach [7], [8], Distance Index [6] and Distance Browsing [13] (labeled as NetExp, euclidean, DistIdx, and DistBrws, respectively), all in GNU C++ for comparison. NetExp serves as the baseline approach in our evaluation. We adopt CCAM [22] to organize network nodes in disk storage for all the approaches. For NetExp,

5. CA, NA and SF are obtained from [20] and PRS is from [21].



(a) (b)

Fig. 13. Index construction (varying no. of objects on NA). (a) Index construction time. (b) Index size.

objects are stored with network nodes. For euclidean, objects are indexed by an R-tree and A* algorithm [9] is used to determine objects' network distances from query nodes. For DistIdx and DistBrws, distance signatures and shortest path quad-trees (SPQT) are stored with network nodes, respectively. For DistIdx, we adopt exact object distances in the distance signature to provide its optimal search performance.

Four performance metrics are measured in this evaluation, i.e.,

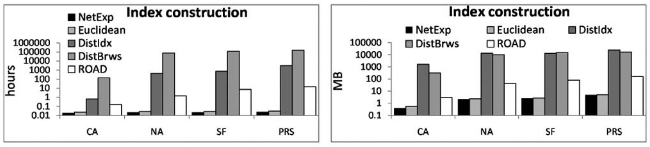
1. *index construction time*: the time to construct an index;
2. *index size*: storage used to store an index;
3. *query processing time*: time duration from the time a query is initiated to the time a complete result is obtained; and
4. *index update time*: the time spent in maintaining the underlying indices when a update (either object update or network update) is processed.

All experiments were conducted upon Linux 2.6.9 servers with Intel Xeon 3.2 GHz CPU and 4 GB RAM. Unless explicitly stated, each experiment set presented below evaluates one parameter while using the default values for other parameters.

8.1 Index Construction

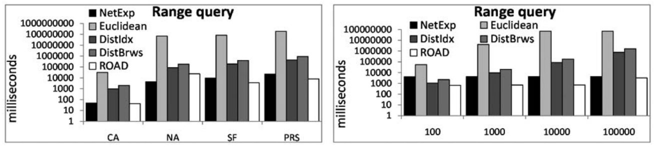
The first experiment set evaluates the index construction time and index sizes for all the approaches with various number of objects and networks. Here, we use the default p and l for ROAD and leave the evaluation of their impacts in the final set of experiments. Fig. 13 shows the index construction time (in hours) and index sizes (in megabyte) for various number of objects on NA. Since the construction time is not affected by the object distribution, the experiment results are not presented here. In the figure, NetExp and euclidean incur the smallest index construction times (in a few minutes) and index size (in a few MBs). ROAD takes around 1 hour construction time and about 20 MB storage space. In contrast, due to the bulky SPQTs, DistBrws takes an extremely long construction time (over 100,000 hours) and a huge storage (over 10 GB), though it is almost invariant to the number of objects. As for DistIdx, both the construction time and index size increase drastically as the number of objects evaluated increases. When 100,000 objects are experimented, DistIdx consumes 100 GB storage size and takes more than 10,000 hours to build the index![6] This experiment result reveals that both DistBrws and DistIdx are impractical in real applications and the ideas of query

6. In our implementation, we construct indices for DistBrws and DistIdx using multiple computers as the construction of SPQTs and distance indices for different nodes can be divided. The total machine time used is reported here.



(a) (b)

Fig. 14. Index construction (CA, NA, SF, PRS). (a) Index construction time. (b) Index size.



(a) (b)

Fig. 16. Single-source range query performance. (a) Networks. (b) No. of objects (NA).

precomputation and materialization of shortest paths between nodes or toward objects are not appealing.

Fig. 14 shows the index construction times and index sizes for different networks with the number of objects fixed at 10,000 and the number of clusters fixed at 100, i.e., 100 objects per cluster. As shown in the figure, NetExp and euclidean incur the shortest index construction times and smallest storage overhead. However, they both are not query efficient as will be discussed next. On the other hand, DistIdx, DistBrws and ROAD incur different index construction time and index size as networks change but ROAD always outperforms the other two. For example, when the largest network PRS is evaluated, DistIdx takes over 1,000 hours to build the index and 10 GB to store it; while DistBrws takes over a month to build the index and more than 15 GB to store it. Differently, ROAD incurs significantly shorter construction time (less than 1 hour) and consumes less storage (about 18 MB). Compared with DistIdx, ROAD only requires around 0.6 percent of its index construction time and 0.03 percent of its index size.

8.2 Query Performance

The second set of experiments evaluate the search performance of ROAD and other approaches in answering single-source LDSQs and multisource LDSQs on the following factors:

1. networks,
2. numbers of objects,
3. object distributions,
4. query parameters, and
5. the number of sources for multiple-source LDSQs.

In the experiments, we generate 100 random queries and report the average query processing time.

8.2.1 Experiments on Single-Source kNN Query

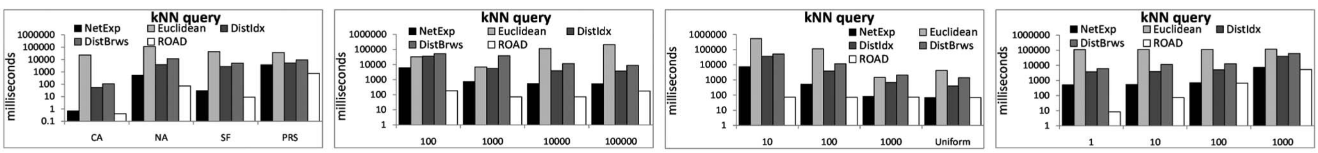
First, we conduct evaluations for single-source kNN queries. As depicted in Fig. 15a, euclidean performs the worst because of exhaustive shortest path searches for a possibly large number of candidate objects, consistent with the observations made in [7]. Further, both DistIdx and DistBrws perform worse than NetExp and ROAD due to the excessive accesses to distance signatures and shortest path quad-trees and slow node-by-node network traversals. As expected, ROAD consistently performs the best. For clustered objects, ROAD can effectively bypass those Rnets with no object of interest. The improvement of ROAD over NetExp ranges from 1.6 (for CA) to 5.1 (for PRS).

Then, we evaluate the impact of object cardinalities and object distributions. With fewer objects and/or fewer clusters, more subspaces with no object of interest can be pruned so that ROAD can achieve a better performance. When we increase the number of objects from 100 up to 100,000 with the number of clusters fixed at 100, all the approaches (except euclidean) have their search performance improved, as shown in Fig. 15b. This is because a larger object cardinality implies a higher density. The average distance between query points and their kth NN object becomes shorter, which, in turn, cuts down the network traversal cost for kNN searches. ROAD consistently demonstrates the best performance. For example, it only requires 33 percent of NetExp's processing time when 100,000 objects are evaluated. Besides, we evaluate the impact of objects distribution via varying the number of clusters from 10 up to 1,000 and examining the uniform distribution. Fig. 15c plots the experiment results. When objects are scattered in the network, the average distance from query points to objects is also shortened. As such, the performance of all the approaches is improved. When 10 clusters are experimented, ROAD takes only 1 percent processing time of NetExp.

Finally, we examine the impact of query parameter k (from 1 to 1,000) and the result is plotted in Fig. 15d. While all the approaches take more time when k is increased, ROAD consistently performs the best due to its strong pruning power. When k is 1, ROAD takes 1.7 percent processing time of NetExp.

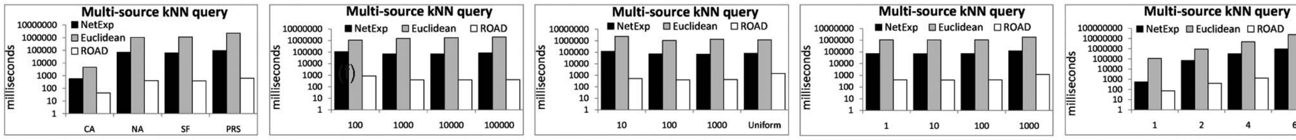
8.2.2 Experiments on Single-Source Range Query

Second, we examine the performance of different approaches for single-source range queries. As observations are very similar to those for kNN queries, we only report the performance over different networks and different object cardinalities for space saving, shown in Fig. 16a and 16b, respectively. From the figures, we can see that



(a) (b) (c) (d)

Fig. 15. Single-source kNN query performance. (a) Networks. (b) No. of objects. (NA) (c) Distributions (NA). (d) k (NA).



(a) (b) (c) (d) (e)

Fig. 17. Multisource kNN query performance. (a) Networks. (b) No. of objects (NA). (c) Distributions (NA). (d) k (NA). (e) No. of sources (NA).

ROAD consistently outperforms all the others and it benefits more from a larger network with a smaller number of objects. Again, euclidean performs the worst as it has to examine a large number of candidate objects. Also, compared with NetExp, both DistBrws and DistIdx do not improve the search performance since they both suffer from the massive access overhead for large networks and large numbers of objects.

8.2.3 Experiments on Multisource kNN Query

Third, we study the performance of the various approaches for multisource kNN queries. Fig. 17 reports results over different settings of

1. networks,
2. numbers of objects,
3. object distributions,
4. query parameter k, and
5. numbers of sources.

As there are no approaches on top of DistIdx and DistBrws presented in the literature supporting multisource LDSQs, we ignore them in this and next experiments. In the first four experiments, we fix m at two (i.e., two-source kNN queries), as shown in Fig. 17a, 17b, 17c, and 17d, and the last experiment studies the impact of m with k set to one (i.e., multisource NN queries), as shown in Fig. 17e. As observed from the results, ROAD consistently performs better than NetExp and euclidean. This is because NetExp has to explore all the subnetworks (i.e., edges and nodes) around query points; while euclidean has to invoke multiple network traversals to determine the network distances of candidate objects. Differently, ROAD can effectively prune away some search spaces that have no result objects.

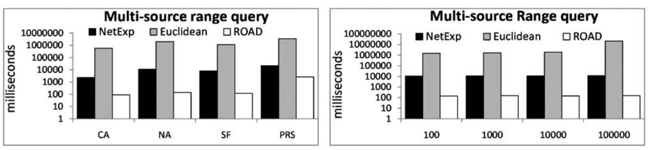
8.2.4 Experiments on Multisource Range Query

Next, we evaluate the performance when multisource range queries are issued. To save space, we focus on the impacts of networks and numbers of objects only with the number of sources fixed at two and the search range fixed at 0.1 of the network diameter. The results are reported in Fig. 18a and Fig. 18b respectively. By pruning away those Rnets without objects of interest, ROAD outperforms the other approaches. When different networks are evaluated, ROAD, on average, takes only 4 percent to 12 percent processing time of NetExp. When the number of objects changes, ROAD takes consistently 13 percent processing time of NetExp in NA with a fixed number of clusters (i.e., 100). This is because range queries request to explore all the nodes/edges within the search range, that is independent of the number of objects. As the search range is fixed, the search performance does not change even when the number of objects varies. Again, euclidean performs the worst due to exhaustive candidate object distance searches.

8.3 Index Update

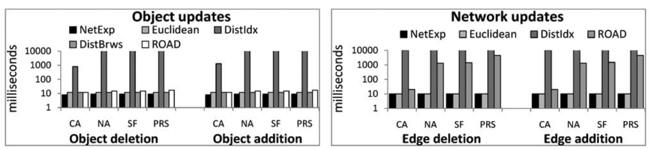
Further, we evaluate the index update cost upon object changes and/or network changes. First, we simulate object changes by removing 100 randomly chosen objects from 10,000 installed objects and then inserting them back one by one. Each deletion and addition involves only one single object. We measure the time taken and report the average performance of deletions and insertions in Fig. 19a. As shown, the update cost incurred by DistIdx is several orders of magnitude higher than that of others, as DistIdx has to update the massive distance signatures. For NA, SF, and PRS, it takes about 10 and 18 minutes to finish one object deletion and addition, respectively. In contrast, NetExp, euclidean, DistBrws, and ROAD can handle an update within 10 milliseconds for all the networks since the objects are logically independent from networks in these approaches.

Conversely, we simulate network changes by setting 100 edge distances to infinity and later recovering their original edge distances. Each edge update involves one edge only. The average performance of 100 trials is presented in Fig. 19b. DistBrws is not examined as no efficient shortest path quad-tree update mechanism is reported. The edge change has almost unobservable impacts on NetExp and euclidean. However, for DistIdx, the distance signatures of many nodes need reexamination and update, resulting in large processing times. In contrast, ROAD only needs to update affected shortcuts of certain border nodes of Rnets and it has considerably much lower update costs (within 5 seconds) than DistIdx (about 20 minutes).



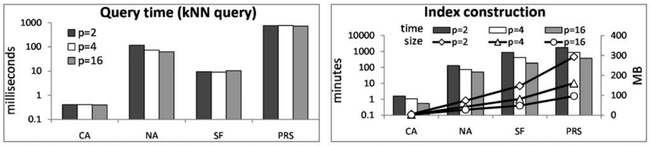
(a) (b)

Fig. 18. Multisource range query performance. (a) Networks. (b) No. of objects (NA).



(a) (b)

Fig. 19. Index updates. (a) Object update. (b) Edge update.



(a) (b)

Fig. 20. Combinations of p and l. (a) Query time (kNN). (b) Index overhead.

8.4 Evaluation on p and l

Last but not least, we examine the impacts of factors p and l on the Rnet hierarchy formation, which in turn affects the performance of ROAD. In this experiment, we try different $\langle p, l\rangle$ pairs with p^l fixed at 256 (i.e., 2^8, 4^4, and 16^2) for CA and 65,536 (i.e., 2^{16}, 4^8, and 16^4) for NA, SF, and PRS. The results in terms of query processing times for single-source kNN queries ($k = 10$) and indexing overhead in terms of index construction times and sizes are plotted in Fig. 20.

We can observe from the figure that ROAD performs similarly in terms of query processing times under different $\langle p, l\rangle$ pairs. Meanwhile, a smaller l (with a corresponding larger p) results in a smaller index and a shorter construction time. This finding suggests the design of ROAD should adopt a smaller l and a larger p, given a fixed number of finest Rnets targeted (i.e., p^l). Both observations are consistent with our performance analysis in the previous section.

From this evaluation, we can see the efficiency of ROAD to support single-source and multisource LDSQs. It outperforms competitive approaches, namely, NetExp, euclidean, DistIdx, and DistBrws, owing to its effective search space pruning capability that is not explored by any of existing approaches. Meanwhile, ROAD provides moderate and very practical construction and maintenance cost compared with the state-of-the-art approaches, DistIdx and DistBrws.

9 Conclusion

The on-going trend of web-based LBSs demands a system framework that can be extended to accommodate diverse objects, provide efficient processing of various LDSQs, and support different distance metrics. In response to these needs, we propose ROAD, a new system framework for LDSQ processing, in this paper. The design of ROAD achieves a *clear separation* between objects and network for better system extensibility. It also exploits *search space pruning*, a powerful technique for efficient object search. Upon the framework, efficient search algorithms for single-source and multisource LDSQs are devised. Via a comprehensive performance evaluation on real road networks, ROAD is shown to significantly outperform the state-of-the-art techniques.

Recently, various LDSQs, such as continuous queries [4], skyline queries [23] and optimal location queries [24], were researched. However, existing works addressed them based on the solution-based approaches or extended spatial database approaches and thus suffered from the shortcomings of those approaches. In the future, we are going to extend our ROAD framework to support those emerged LDSQs.

Acknowledgments

In this research work, Wang-Chien Lee, Ken C. K. Lee, and Yuan Tian were supported in part by US National Science Foundation (NSF) grants CNS-0626709, IIS-0534343, IIS-0328881 and Taiwan National Science Council grant 98-2219-E-001-001; and Ken C. K. Lee was partly supported by UMass Dartmouth Healey Grant 2011.

References

[1] Garmin, "POI Loader," http://www8.garmin.com/products/poiloader, 2010.
[2] Google, "Map Maker," http://www.google.com/mapmaker, 2010.
[3] M.R. Kolahdouzan and C. Shahabi, "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases," *Proc. 30th Int'l Conf. Very Large Data Bases (VLDB),* pp. 840-851, 2004.
[4] H.-J. Cho and C.-W. Chung, "An Efficient and Scalable Approach to CNN Queries in a Road Network," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 865-876, 2005.
[5] H. Hu, D.L. Lee, and J. Xu, "Fast Nearest Neighbor Search on Road Networks," *Proc. 10th Int'l Conf. Extending Database Technology (EDBT),* pp. 186-203, 2006.
[6] H. Hu, D.L. Lee, and V.C.S. Lee, "Distance Indexing on Road Networks," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 894-905, 2006.
[7] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query Processing in Spatial Network Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 802-813, 2003.
[8] M.L. Yiu, N. Mamoulis, and D. Papadias, "Aggregate Nearest Neighbor Queries in Road Networks," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 6, pp. 820-833, 2005.
[9] R. Dechter and J. Pearl, "Generalized Best-First Search Strategies and the Optimality of A^*," *J. ACM,* vol. 32, no. 3, pp. 505-536, 1985.
[10] E.W. Dijkstra, "A Note on two Problems in Connexion with Graphs," *Proc. Numerische Mathematik,* pp. 269-271, 1959.
[11] 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 Trans. Knowledge and Data Eng.,* vol. 10, no. 3, pp. 409-432, 1998.
[12] S. Jung and S. Pramanik, "An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps," *IEEE Trans. Knowledge and Data Eng.,* vol. 14, no. 5, pp. 1029-1046, 2002.
[13] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable Network Distance Browsing in Spatial Databases," *Proc. 2008 ACM SIGMOD Conf.,* pp. 43-54, 2008.
[14] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Efficient Query Processing on Spatial Networks," *Proc. 13th Ann. ACM Int'l Workshop on Geographic Information Systems,* pp. 200-209, 2005.
[15] Y.-W. Huang, N. Jing, and E.A. Rundensteiner, "Effective Graph Clustering for Path Queries in Digital Map Databases," *Proc. Fifth Int'l Conf. Information and Knowledge Management (CIKM '96),* pp. 215-222, 1996.
[16] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Systems Technical J.,* vol. 49, no. 2, pp. 291-308, 1970.
[17] K.C.K. Lee, W.-C. Lee, and B. Zheng, "Fast Object Search on Road Networks," *Proc. Int'l Conf. Extending Database Technology (EDBT),* pp. 1018-1029, 2009.
[18] B.H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Comm. ACM,* vol. 13, no. 7, pp. 422-426, 1970.
[19] C. Faloutsos and S. Christodoulakis, "Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation," *ACM Trans. Information Systems (TOIS),* vol. 2, no. 4, pp. 267-288, 1984.
[20] F. Li, "Li's Collections of Real Road Network Data," http://www.rtreeportal.org, 2010.
[21] NAVTEQ, "Development Resources," http://developer.navteq.com, 2010.
[22] S. Shekhar and D.-R. Liu, "CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations," *IEEE Trans. Knowledge and Data Eng.,* vol. 9, no. 1, pp. 102-119, 1997.

[23] K. Deng, X. Zhou, and H.T. Shen, "Multi-source Skyline Query Processing in Road Networks," *Proc. 23rd IEEE Int'l Conf. Data Eng. ICDE*, pp. 796-805, 2007.

[24] Y. Du, D. Zhang, and T. Xia, "The Optimal-Location Query," *Proc. Int'l Symp. Spatial and Temporal Databases (SSTD '05)*, pp. 163-180, 2005.

Ken C.K. Lee is currently an assistant professor at the Department of Computer and Information Science, University of Massachusetts, Dartmouth. In Fall 2009, he received the PhD degree from the Pennsylvania State University, University Park. He received the BA and MPhil degrees from the Hong Kong Polytechnic University.

Wang-Chien Lee is an associate professor of computer Science and engineering at Pennsylvania State University (PSU), University Park. He received the BS degree from the Information Science Department, National Chiao Tung University, Taiwan, the MS degree from the Computer Science Department, Indiana University, and the PhD degree from the Computer and Information Science Department, the Ohio State University. Prior to joining PSU, he was a principal member of the technical staff at Verizon/GTE Laboratories, Inc. Dr. Lee leads the Pervasive Data Access (PDA) Research Group at PSU to pursue cross-area research in database systems, pervasive/mobile computing, and networking.

Baihua Zheng received the BS degree from Zhejing University and the PhD degree in computer science from Hong Kong University of Science and Technology. Currently, she is an assistant professor in the School of Information Systems at Singapore Management University. Her research interests include mobile and pervasive computing, and spatial databases.

Yuan Tian received the BE degree in electrical engineering from Tsinghua University, China. She is currently working toward the PhD degree at the Department of Computer Science and Engineering, the Pennsylvania State University, University Park. Her research interests include spatial database, location-based services, and social network.