3

Navigation Systems: A Spatial Database Perspective

Shashi Shekhar, Ranga Raju Vatsavai, Xiaobin Ma, and Jin Soung Yoo
Univeristy of Minnesota

CONTENTS

3.1 INTRODUCTION

Navigation systems that guide objects moving from one place to another have progressed recently with the rapid advances in positioning, communication, and spatial data storage and processing technologies. The easy availability of satellite-based global positioning systems has revolutionized all forms of automated navigation. Other positioning technologies such as handsets that use user input and networks that use one of the many location-determining methods are also showing continued advances. The proliferation of such location-aware devices provides us with opportunities to develop a diverse range of location-based applications, many of which will use user location-specific information.

Location-based services (LBS) provide the ability to find the geographical location of a mobile device and then provide services based on that location [OpenLS]. The Open GIS Consortium (OGC) recently initiated the OpenLS standard, which addresses the technical specifications for LBS, to enhance a range of personal, governmental, industrial, and emergency mobile applications. Location-based systems and geographic information systems (GIS) share many common features. At the heart of the OpenGIS Location Services (OpenLS) standard lies the GeoMobility server, which comprises abstract data types (ADTs) and the core services through which a service provider can provide location application services and content to any service point on any device. The core services are location utilities services, directory services, presentation service, gateway service, and route determination service.

These location-based application services require a spatial database (SDB) server, which provides effective and efficient retrieval and management of geospatial data. Spatial database systems serve various spatial data (e.g., digital road maps) and nonspatial information (e.g., route guidance instruction) on request to the client. SDB servers provide efficient geospatial query-processing capabilities such as find the nearest neighbor (e.g., gas station) to a given location and find the shortest path to the destination. The SDB system acts as a back-end server to the GeoMobility server. Thus SDB servers play a crucial role in implementing efficient and sophisticated navigation system applications. This chapter introduces navigation systems from a spatial database perspective.

Section 3.2 briefly reviews the history of navigation systems and provides a generic architecture of a typical navigation system based on

OpenLS specifications. In the subsequent sections, each component of this architecture is presented in detail. Section 3.3 presents various components of SDBs using a digital road map as an example. Section 3.4 introduces gateway service, and Section 3.5 addresses location utility service. Section 3.6 presents the components of directory service. Section 3.7 addresses route determination service and Section 3.8, presentation service. The chapter concludes with a discussion on future research needs.

3.2 NAVIGATION SYSTEMS

A modern navigation system is an integrated collection of position and orientation sensors and computing and communication hardware and software used to facilitate the movement of people, vehicles, and other moving objects from one place to another. It includes methods for determining position, course, and distance traveled. The platform could be anything from land-based vehicles to space-based satellites. So while navigation is the process that guides the movement of an object between two points in space, navigation systems are the hardware and software components that facilitate automated and intelligent navigation. As such, navigation systems cover a broad spectrum of integrated technologies that allow accurate determination of the geographic coordinates of the (moving) objects, their velocity, and height.

The history of navigation is as old as human history, although early navigation was limited to following landmarks and memorizing routes. Historical records show that the earliest vehicle navigation dates back to the invention of the south-pointing carriage in China around 2600 B.C. A brief discussion of other historic vehicle navigation systems can be found in [Zha97]. Well-known navigation devices that were extensively used in early navigation are the magnetic compass and the odometer. The 17th-century discovery of chronometer by John Harrison provided accurate local time at sea, which helped in solving the long-known problem of estimating longitudes. The use of navigation devices in automobiles began in the early 20th century. Many modern-day automobiles are equipped with devices that are capable of determining the current location and then dynamically displaying and updating the current position on digital road maps.

Over the centuries, various kinds of technologies have been tried for navigation. The discovery of global positioning systems (GPS) has changed the face of modern navigation forever. In modern vehicle navigation, the second-generation guidance system developed by the U.S. Department of Defense in the mid-1980s, known as the Navigation Satellite Timing and Ranging (NAVSTAR) global positioning system, is becoming widely used. The positional accuracy of civilian GPS receivers has been improved to +/− 10 meters. Submeter accuracy can also be obtained through differential GPS. Navigation inside confined spaces, such as buildings, can be achieved through indoor location-sensing devices. The commonly used sensors for indoor navigation are infrared and short-range radios. Some example indoor navigation systems are Active Badges [WHFG92], Active Bat [HHS + 99], ParcTAB [AGS + 93], and the Cricket system [PCB02].

This ability to accurately determine the position of moving objects gave rise to new services known as location-based services. LBS uses accurate and real-time positioning systems and GIS to determine the location of a moving object. The information generated by these systems is sensitive to the current position of the user and can be used to advise users about current conditions such as weather and traffic. Thus navigation systems are the backbone of the location-based services.

The OGC recently initiated the OpenLS standard to address the technical specifications for LBS. The core of LBS applications is the back-end SDB server, which provides efficient storage, management, and processing capabilities for geospatial data. The limitations of earlier navigation systems, which were confined to simple positioning devices and paper-based maps (e.g., road maps, navigation charts), have been diminished with the availability of accurate digital road maps and digital communication systems. The SDB server provides dynamic information on demand to aid automated navigation. Thus navigation systems along with SDBs provide us with opportunities to develop innovative applications ranging from a simple trip plan to complex mobile object monitoring and management systems.

The general architecture of a modern navigation system is shown in Figure 3.1. The components can be broadly classified into four subsystems: the SDB server, the GeoMobility server, communication systems, and the location-aware clients. Client-side components include position-aware devices that range from personal digital assistants (PDAs) and cellular phones to cars, ships, airborne vehicles, and laptops. The client can be totally independent; in that case, the devices can also include small (static) SDBs (e.g., CD/DVD-ROMs); however, in many

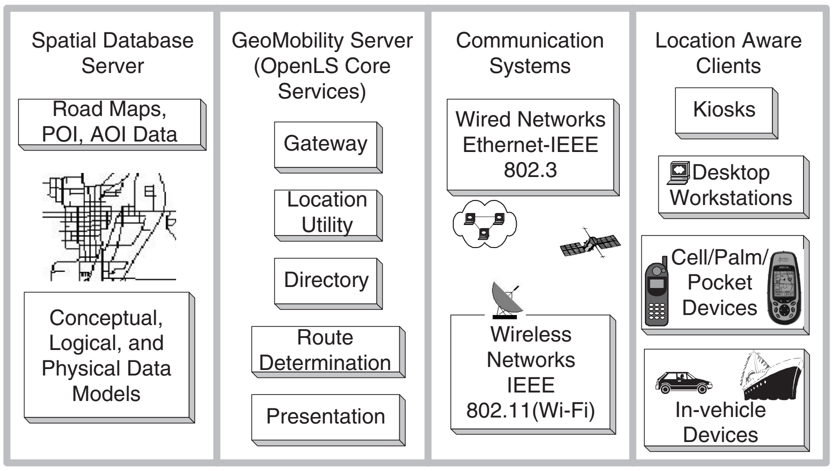


Figure 3.1 Architecture of a modern navigation system.

location-based service applications, the client depends on a server for various services and communicates with the server through wireless and Internet network technologies.

The server-side components include a Web server, a large SDB server, and the application server. The client and server interact through wireless communications. Client-side devices use various visual interfaces (e.g., graphical user interface [Bon93; Mac96; ST91], voice recognition [YLM95; Rab95]) to interact (query and presentation) with the server. Building applications that integrate heterogeneous technological pieces in a viable way is impossible without the help of standards, and OpenLS [OpenLS] is such a standard. Several other standards are similar or address specific issues; for example, Location Interoperability Forum's (LIF) standard addresses location determination methods, and ISO TC/204 deals with navigation data formats. The discussion in this chapter is based on the OpenLS standard and describes how the various subsystems work together to form various navigation system applications.

3.2.1 *Spatial Database Server*

An SDB [Guting94; SCR+99; SC02; RSV01] management system aims at the effective and efficient management of data related to a space in the physical world (e.g., geographic or astronomical space). An SDB server is an essential component for building efficient navigation system applications. It provides conceptual, logical, and physical data modeling facilities to build and manage spatial databases. It serves various spatial (e.g., digital road maps) and aspatial information (e.g., route guidance instructions) on request to the client. It also provides various geospatial query-processing capabilities, such as find the nearest neighbor (e.g., restaurant) to a given location and find the shortest route between two points. It acts as a back-end SDB server to the GeoMobility server. Commercial examples of SDB management systems include Oracle Spatial [SDC], DB2 Spatial Extender [DB2Spatial]), and ESRI's Spatial Database Engine [ESRI].

3.2.2 Open Location Services and GeoMobility Server

The OGC is an international consortium for developing publicly available geoprocessing specifications. Most of these specifications have been adopted by the industry, and as a result an extremely successful interoperable geospatial infrastructure now exists. The OGC recently initiated an Open Location Services Initiative [OpenLS], which aims at the development of interface specifications that facilitate the use of location and other forms of spatial information in the wireless Internet environment. The purpose of the Initiative is to produce open specifications for interoperable location application services that will integrate spatial data and processing resources into the telecommunications and Internet services infrastructure [OpenLS]. The OpenLS specification allows the deployment of interoperable location-based products and services that will have a far-reaching impact on industry and society.

The *GeoMobility server* is an OpenLS platform through which content/service providers can deliver and service location-based applications. The core services are Location Utilities Service, Directory Service, Presentation Service, Gateway Service, and Route Determination Service.

- *Location Utilities Service*. The OpenLS utilities specification provides two services, geocoding and reverse geocoding, and an abstract data type named as Address. The geocoder service is a network-accessible service that transforms a description of a location into a normalized description of the location with point geometry. The reverse geocoder service maps a given position into a normalized description of a feature location.
- *Directory Service*. The directory service provides a search capability for one or more points of interest (e.g., a place, product, or service with a

fixed position) or area of interest (e.g., a polygon or a bounding box). An example query is ''Where is the nearest Thai restaurant to the EE/CS department?''

- *Presentation Service*. This service deals with visualization of the spatial information as a map, route, and/or textual information (e.g., route description).
- *Gateway Service*. This service enables obtaining the position of a mobile terminal from the network.
- *Route Determination Service*. This service provides the ability to find a best route between two points that satisfies user constraints. This service and other network analysis capabilities are described in Section 3.7.

3.2.3 Communication Systems

Telecommunications have undergone significant changes in the last 30 years. In the 1970s, analog communications gave way to digital communications and circuit-switching technology gave way to packet-switching technology. In the early 1980s, the original ARPANET began to evolve into the current Internet. In its early days, the Internet comprised a handful of small networks at universities and defense establishments. Explosive growth of the Internet began during the late 1980s, at the same time personal computers revolutionized the home computing environment.

The Internet can be viewed as the interconnection of thousands of Local Area Networks (LANs) and Wide Area Networks (WANs). Ethernet is the most widely installed LAN technology to connect multiple computers together to enable applications such as file sharing, electronic mail, and Internet access. WANs are simply the interconnection of two or more LANs using some form of telecommunication medium. Asynchronous Transfer Mode (ATM) has been widely adopted for WAN interconnections. Most modern WAN protocols, including TCP/IP and X2.5, are based on packet-switching technologies. ATM combines the best of both worlds (i.e., the guaranteed delivery of circuit-switched networks and the efficiency of packet-switched networks).

The most recent advance in telecommunications is wireless telephony, commonly known as cell phones. Cell phone usage grew exponentially in the United States during the 1990s. More history of communications can be found in [ComSoc]. Today, wireless communication plays an important role in navigation systems. It is what makes user mobility over large geographic areas possible. Both analog and digital wireless systems are used in current communication systems. However, some wireless communication applications such as paging may still need access to wired networks (e.g., Public Switched Telephone Network), and the Internet readily provides countless access points for wireless subnetworks. Some navigation systems such as short-range beacons (used for vehicle to roadside communications) also need wired networks to transfer information from the beacon heads to the navigation management center. Navigation devices communicate with the roadside beacon acceptor, which then transfers information to the navigation center. Location-based services are thus dependent on both wired and wireless networks.

3.2.4 *Location-Aware Clients*

Client-side devices in the architecture of a modern navigation system consist of three basic components: a position and orientation module, a computing module consisting of display and storage, and a communication module. Each client-side module may not necessarily be equipped with all three modules but still can be part of a location-based application. For example, a PDA without a positioning module can utilize the gateway service to obtain its current location. Client-side devices vary widely in nature and function; example devices include but are not limited to PDAs, cell phones, laptops, and land, sea, and airborne vehicles. Client devices may additionally be equipped with visual display units (e.g., touch screens) and voice recognition systems. Stand-alone (or thick) clients can store SDBs locally, either on CD-ROMs, DVDs, or hard disks; however, many location-based clients may need to access GeoMobility Servers.

3.3 SPATIAL DATABASES

3.3.1 *Digital Road Maps*

Location-based services depend heavily on digital road maps, postal addresses, and point-of-interest data sets. These maps are indispensable for any location-based utility that involves position- (e.g., street address) or route-based queries. Current road navigation systems use digital road maps available on CDs or DVDs, but many applications (e.g., emergency services) require dynamic updates from back-end SDB servers. Traditionally, digital road maps were available through governmental departments (e.g., state departments of transportation); however, due to commercial demand, several private-sector companies have begun to offer digital road maps with additional points of interest and areas of interest information. Table 3.1 summarizes various digital road map sources along with important characteristics. Given a wide variety of heterogeneous digital road map databases, it is imperative to understand data quality before building any LBS application.

Table 3.1 Digital road map sources.

Source	Provider	Coverage	Comments
TIGER [TIGER]	U.S. Department of Commerce, Census Bureau	USA	Aggregated from many sources, e.g., USGS, State DOTs, etc. Accuracy inadequate for OLSs in many areas
State base map [MNDOT]	State department of transportation, e.g., MN/DOT	Minnesota, USA	Digitized from 1:24000 USGS paper map
Navtech [NAVTECH]	Navigation Technologies Corporation	USA, North America, Western Europe	Cleaner version of TIGER file Best accuracy for urban areas
Etak [ETAK]	Tele Atlas	USA, Canada, Western Europe, Hong Kong, Singapore	Best accuracy for urban areas
GDT [GDT]	Geographic data technology	USA, Canada	Better accuracy for nonurban areas
Digital Map 2500 [GSI93]	Geographical Survey Institute (GSI), Japan	Japan	The spatial data framework (SDF2500) includes roads and railways for city planning and as well for Japan as a whole
Philips-Digital Map Data [PDMD03]	Graticule	Great Britain, Europe	Street data specialized for navigation available at different scales

Data quality refers to the relative accuracy and precision of a particular GIS database. These facts are often recorded as a part of metadata. Digital road maps are an important category of geospatial data. The purpose of a geospatial data quality report is to provide detailed information for a user to evaluate the fitness of geospatial data for a particular use. To provide a data quality report based on geospatial data standards, a digital data producer is urged to include the most rigorous and quantitative information available on the components of data quality. In fact, data quality is a part of the geospatial metadata defined by the Federal Geographic Data Committee (FGDC) [FGDC01]. The metadata standard documents the content, quality, condition, and other characteristics of data so that geospatial digital data users can evaluate the data fitness for their purpose. This standard provides a common set of terminology and definitions for the documentation of spatial data, including information on identification, data quality, spatial data organization, spatial reference, entities and attributes, distribution information, and metadata references. There are several map accuracy standards, including the well-known National Map Accuracy Standard (NMAS) [NMAS] and the American Society for Photogrammetry and Remote Sensing (ASPRS) standard [ASPRS]. There are four components of data quality standards:

1. *Lineage*. Refers to the narrative of source materials (e.g., USGS quad sheets) used and procedures (e.g., map projection, map generalization) applied to produce the product.

2. *Positional Accuracy*. Defines expected error in position of features (e.g., landmark points). For example, an NMAS-compliant map guarantees 90% of features within 40 feet of their true position at 1:24,000 scale. New standards (e.g., ASPRS) revised this accuracy for well-defined points.

3. *Attribute Accuracy*. Defines expected error in attributes (e.g., road names). For example, a road map may claim 90% accuracy for a road name attribute.

4. *Completeness*. Defines the fraction of real-world features depicted on a map. For example, a road map includes 99% of available streets.

For current digital road maps, positional accuracies, which are of the most concern in navigation systems, vary greatly for different map sources. Following is a summary of accuracy claims from different sources:

- TIGER: mean error = 281 feet (90 m), Median error = 166 feet (50 m). 90th percentile from 110 m to 440 m across different sources
- Basemap: 40 feet at 1:24,000 scale, 166 feet at 1:100,000 scale
- NavTech: 97% accuracy, percent error = linear combination of 13 component errors
- Etak: 40 feet at 1:24,000 scale (cover 70% population), 166 feet at 1:100,000 scale (cover 25% population), 90% of tested points less than threshold
- GDT: In enhanced regions, 5 m to 7 m

3.3.2 *Data Model of Digital Road Maps*

This section presents techniques related to the data modeling of a location-based application. The focus is a digital road map. Database applications are modeled using a three-step design process [EN01]. In the first step, all of the available information related to the application is organized using a high-level *conceptual data model*. The second step, also called the *logical modeling* phase, is related to the actual implementation of the conceptual data model in a commercial database management system (DBMS). The third and final step, modeling of the *physical design*, deals with the nuts and bolts of the actual computer implementation of the database applications.

CONCEPTUAL DATA MODEL

At the conceptual level, the focus is on the data types of the application, their relationships, and their constraints. The actual implementation details are left out at this step of the design process. Plain text combined with simple but consistent graphic notation is often used to express the conceptual data model. The Entity Relationship (ER) model is one of the most widely used conceptual design tools, but it has long been recognized that it is difficult to capture spatial semantics with ER diagrams. The first difficulty lies with geometric attributes, which are complex, and the second difficulty lies with spatial relationships. Several researchers have proposed extensions to the existing modeling languages to support spatial data modeling. The pictogram-enhanced ER (PEER) model

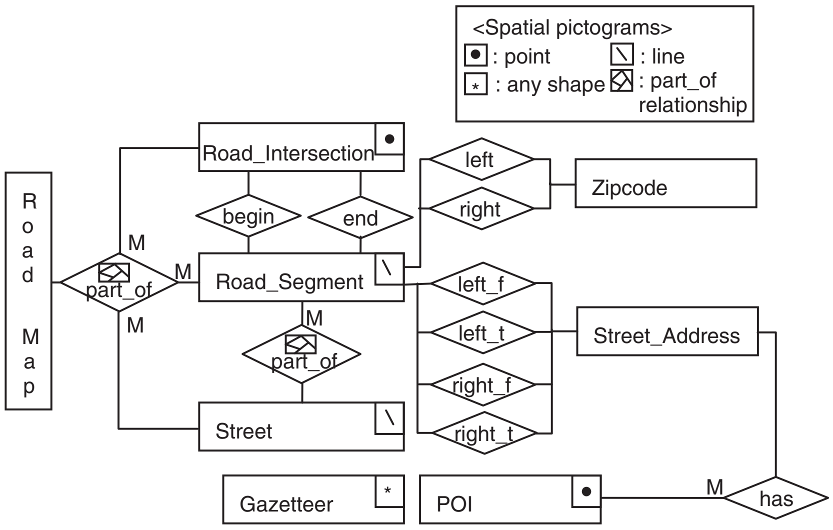


Figure 3.2 A PEER diagram for a digital road map.

proposed in [SVCB99] is used to show the conceptual model of a digital road map.

Figure 3.2 shows a PEER diagram for a digital road map. Spatial networks (e.g., road maps) are modeled as graphs, where vertices are points embedded in space. *Graph* consists of a finite set of *vertices* and a set of *edges*. In a digital road map, vertices represent road intersections and edges represent road segments, which are lines connecting two intersections. Sometimes labels (e.g., name) and weights (e.g., miles, travel time) are attached to each vertex and edge to encode additional information. A road segment is modeled with (a range of) street addresses, which is commonly used in geocoding (i.e., assigning a coordinate to a given address such as "511 Washington Ave") and reverse geocoding (i.e., finding the address given a coordinate), as suggested in [VW01]. The street addresses are divided into left-side addresses and right-side addresses. Each side keeps two end addresses: from and to. The zip code information of a street address is used for searching a map when the exact address is unknown. The left-side and right-side zip codes are also attached to the road_segment. Two edges are *adjacent* if they share a common node. A sequence of adjacent edges constitutes a *path*. At the conceptual level, a path is modeled as a street. This diagram also includes Point of Interest (POI) and Gazetteer entities for supporting directory service of the OpenLS standard.

LOGICAL DATA MODEL

The logical modeling phase is related to the actual implementation of the conceptual data model in a commercial DBMS. Data are organized using an implementation model without any regard to actual storage details. Examples of implementation models are hierarchical, network, relational, and object-oriented models. A hybrid object-relational data model is also gaining popularity and being implemented in current commercial SDBs. [SVCB99] provided the grammar-based translation scheme for mapping a pictogram-extended ER model onto an object-relational model. This mapping uses OGC simple feature specification for SQL [OGC98]. The SQL functions (methods) specified by the OGIS specification fall into three categories: (1) basic functions on the Geometry data types, (2) operators for testing topological relationships, and (3) functions that support spatial analysis. The OGIS standard specifies the data types and the operations on these data types that are essential for spatial applications such as GIS.

Although relational database management systems (RDBMS) provide a fixed set of data types, object-relational database management systems (ORDBMS) support recently standardized SQL3, which allows user-defined data types. This mechanism allows user-defined complex spatial data types such as point, line, and polygon. The actual mapping between a PEER model and OGIS/SQL3 logical model is guided through the definition of grammar and the translation rules. In general, entity pictograms translate into appropriate data types in SQL3, and the relationship pictograms translate into spatial integrity constraints.

Table 3.2 shows a relational schema for the digital road map example. There are six tables: Road_Map, Road_Intersection, Road_Segment, Street, POI, and Gazetteer. The Road_Map table is represented as an adjacency_list graph in order to support routing algorithms (see Section 3.7). The relationships of left-side and right-side zip codes, and the four street address relationships for geocoding, are placed as attributes in the road_segment relation. In addition, commercial database companies have introduced the notion of providing application-specific packages, which provide a seamless interface to the database user. For example, Oracle provides a *Spatial Data Cartridge* package [SDC] for GIS-related applications.

PHYSICAL DATA MODEL

In the physical data modeling phase, issues related to storage, indexing, and memory management are addressed. Physical database design is

Table 3.2 Relational schema for a digital road map.

Table Name	Primary Key	Attributes	Secondary Indices
Road_Map	Map_Id	Cover_area, nodes with	CCAM on road map
(A nested table)		adjacency lists	
Road_Intersection	Intersection_Id	Coordinate	
Road_Segment	Segment_Id	Begin_intersect_Id,	
		End_intersect_Id,	R-Tree
		Shape,	B+tree
		Street_Id, Distance,	B+tree
		Left_ zipcode, Right_ zipcode,	
		Left_from_street_addr,	
		Left_to_street_addr,	
		Right_from *street*addr,	
		Right_to_street_addr	
Street	Street_Id	Street_name, Street_type, Direction,	
		Speed, Oneway	
POI	POI_Id	Type,	B+tree
		Name, Address,	R-Tree
		Coordinate	
Gazeter	Type	Name,	
		Address	

critical to ensure reasonable performance for various queries written in an elegant but high-level logical language such as SQL, which provides no hints about implementation algorithms or data structures. Historically, physical database design techniques such as B+ tree index are credited for the large-scale adoption of relational database technology by providing reasonable response time for SQL queries of many kinds. Well-known file organizations are hashed files and ordered files; however, ordered file organization cannot be used directly for spatial objects (e.g., location of a city) because no total order is defined on points in a multidimensional space. This situation has given rise to several mapping techniques such as Z-order and Hilbert curves, also known as space-filling curves. Even though there is no ideal mapping technique, the mapping of points in multidimensional space into one-dimensional values will enable the use of the well-known B+ tree indexing structure.

A fundamental idea in spatial indexing is the use of approximations. This allows index structures to manage an object in terms of one or more spatial keys, which are much simpler geometric objects than the object itself. The prime example is the *bounding box* (the smallest axis-parallel rectangle enclosing the object). For grid approximations, space is divided into cells by a regular grid, and the object is represented by the set of cells that it intersects. Well-known spatial indexing structures are R-tree and several variants of it. An R-tree is a height-balanced tree that is the natural extension of a B-tree for *k*-dimensions. Objects are represented in the R-tree by their minimum bounding rectangle (MBR). Figure 3.3 shows a set of spatial objects (MBRs) in a two-dimensional space and an R-tree for the set of MBRs. These well-known indexing methods provide efficient query processing involving point and range queries.

For the digital road map example, B+ tree can be used on street address attributes of road_segment for geocoding (see Table 3.2). For example, to transfer from a given address, "511 Washington Ave, Minneapolis, MN" to a coordinate, first find the road segment using secondary indices on street addresses and then search for a coordinate of the given address through connected road segments. For reverse geocoding (i.e., finding the street address given a coordinate), we can use an approximation method using a spatial index (e.g., R-tree in which the road segment is indexed in terms of one spatial key). The nearest road segment object to a query point gives the street address. Similarly, in the POI table, B+ tree can be used for text-based searches (e.g., name) and R-tree, which is defined on point coordinates, might be used for supporting proximity queries (point query, range query, and nearest neighbor query).

Several location-based services, especially those that deal with network databases (e.g., road networks), have to deal with efficient network computations. Figure 3.4 shows three different representations of a graph.

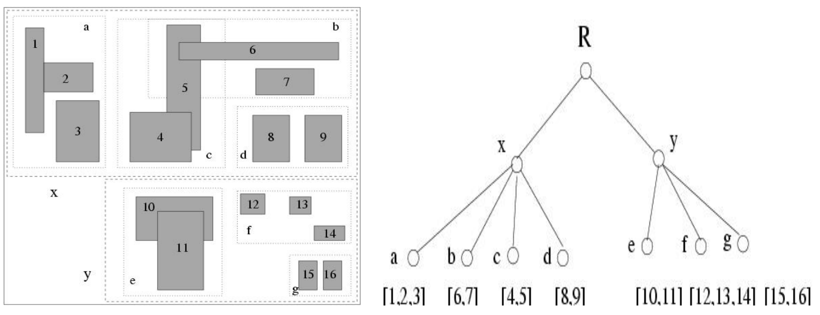
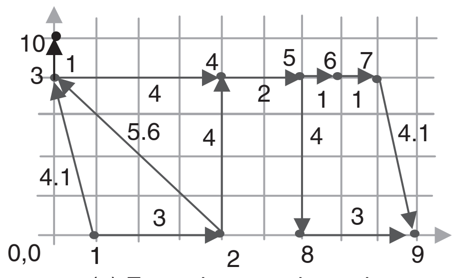


Figure 3.3 A collection of spatial objects and its R-tree hierarchy.



(a) Example network graph

Destination

Source	1	2	3	4	5	6	7	8	9	10
1	0	1	1	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	1
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	1	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

(b) Adjacency-matrix

1	→....	2	3
2	→	3	4
3	→	4	
4	→	5	
5	→	6	8
6	→	7	
7	→	9	
8	→	9	
9	→	N	
10	→	N	

(c) Adjacency-list

Node (R)

ID	X	Y
1	1	0
2	4	4
3	0	4
4	4	4
5	6	4
6	7	4
7	8	4
8	6	0
9	9	0
10	0	5

Edge (S)

Src	Dst	Dist
1	2	3
1	3	4.1
2	3	5.6
2	4	4
3	4	4
3	10	1
4	5	2
5	6	1
5	8	4
6	7	1
7	9	4.1

(d) Node and Edge Relations

ID	X	Y	Succ.	Pred.
1	1	0	(2,3)	
2	4	4	(3,4)	(1)
3	0	4	(4,10)	(1,2)
4	4	4	(5)	(2,3)
5	6	4	(6,8)	(4)
6	7	4	(7)	(5)
7	8	4	(9)	(6)
8	6	0	(9)	(5)
9	9	0		(8)
10	0	5		(3)

(e) Denormalized Node Table

Figure 3.4 Three different representations of a graph.

The *Adjacency-matrix* and the *Adjacency-list* are two well-known main-memory data structures for implementing graphs. In the *Adjacency-matrix,* the rows and the columns of a graph represent the vertices of the graph. A matrix entry can be either 1 or 0, depending on whether there is an edge between the two vertices, as shown in Figure 3.4(b). The *Adjacency-list* structure is efficient for queries that involve enumerating the vertices of a graph: Find all neighbors of *v*. The *Adjacency-list* data structure is an array of pointers. Each element of the array corresponds to a vertex of the graph, and the pointer points to a list of immediate neighbors of the vertex, as shown in Figure 3.4(c); however, main-memory data structures are not suitable for database applications because the database is usually too big to fit in main memory at one time.

Directed graphs can be implemented in a relational model using a pair of relations for the nodes and edges of the graph. The *Node* (R) and the *Edge* (S) relations are shown in Figure 3.4 (d). A denormalized representation is shown in Figure 3.4 (e). The directed graph representation is often used to speed up shortest path computation. This representation of a node table contains coordinates, a list of successors, and a list of predecessors. This representation is used to model the digital road map example (Road_Map table in Table 3.3).

[SL97] have proposed a new spatial access method called the Connectivity-Clustered Access Method (CCAM) for general network databases. CCAM clusters the nodes of the network via graph partition. In contrast with the previous topological ordering-based approach, CCAM assigns segments to the data page by a graph partitioning approach, which tries to maximize the connectivity residue ratio. Each data page is kept at least half full whenever possible. In addition, an auxiliary secondary index is used to support the Find(), get-a-Successor(), and get-Successors() operations. B+ tree with Z-order can also be used to

Table 3.3 Query types for directory service.

Type	Subtype	Query Attribute	Query Example
Attribute Query	Unique Attribute Query	A unique identifier (e.g., the name (of restaurant), address)	Where is the Red Dragon Chinese restaurant?
	Property Attribute Query	Some property or attribute (e.g., the type of restaurant, named reference category, keyword list)	Where are the Chinese restaurants?
Proximity Query	Point Query	Pointed location (e.g., highlighted location)	Where am I?
	Range Query	Spatial region within some distance of some other location (e.g., within boundary)	Which Chinese restaurants are within 500 meters of my hotel?
	Nearest Neighbor Query	Some point location (e.g., nearest (to my hotel))	Where is the nearest Chinese restaurant to my hotel?

index road maps. Other access methods, such as the R-tree or Grid File, can alternatively be created as secondary indexes in CCAM to suit an application.

3.4 GATEWAY SERVICE

Gateway service is the interface between the Open Location Services Platform and Mobile Positioning Servers through which the platform obtains near real-time position data for mobile terminals.

Positioning and orientation devices are vital to any navigation system. This chapter is concerned only with land-based navigation systems, so here positioning means the determination of the coordinates of a vehicle, person, or any moving object on the surface of the earth. There are three types of positioning systems commonly in use: stand-alone, satellite-based, and terrestrial radio-based [Zha97].

3.4.1 *Stand-Alone Positioning Systems*

Deduced (or "dead") reckoning (DR) is the typical stand-alone technique to determine "current position" with reference to a "starting position," and was commonly used by sailors before the development of celestial navigation. In order to determine the current position, DR incrementally integrates the distance traveled and the direction of travel relative to the known start location. In earlier times, direction used to be determined by magnetic compass, and the distance traveled was computed by the time of travel and the speed of the vehicle. In modern land-based navigation, however, various sensor devices can be used to compute accurate direction and distance traveled by the vehicle. Example sensors are differential odometers, gyroscopes, magnetic compasses, and transmission pickup sensors.

3.4.2 *Satellite-Based Positioning Systems*

The Navigation Satellite Timing and Ranging (NAVSTAR) global positioning system is the well-known satellite-based positioning technology that is widely used in modern vehicle navigation. GPS consists of three parts: (1) the space segment (which is a constellation of 24 operational satellites), (2) the user segment (GPS receivers), and (3) the control segment (consisting of monitoring stations, ground antennas, and the coordinating master control station). The 3D coordinates (latitude, longitude, and altitude) of a GPS receiver can be calculated from the simultaneous observation of three or more satellites with a positional accuracy of 10 meters. Using differential GPS, which combines signals from satellites and ground-based sources with known fixed locations, a positional accuracy of submeter can be achieved. Following is a sample format of GPS data (GPGGA) from the Trimble GPS receiver, where $GPGGA is the message id ($GP) followed by time, position, and fix related data (GGA), and UTC represents coordinated universal time. This structure is confined to the National Marine Electronics Association's NMEA-0183 Version 2.30 format [NMEA].

$GPGGA	*UTC*	*Latitude*	*Lat. Dir. (S/N)*	*Longitude Long. Dir. (E/W)*	*Data Quality*	…

3.4.3 (Terrestrial) Radio-Based Positioning Systems

Radio-based positioning systems are designed for specific applications (e.g., offshore navigation) and are generally managed by government and military/naval agencies. Terrestrial positioning systems commonly employ direction or angle of arrival (AOA), absolute timing or time of arrival (TOA), and differential time of arrival (TDOA) techniques to determine the position of a vehicle. The radio navigation systems commonly operate in three frequencies: low (< 300 kHz), medium (300 kHz–0.3 MHz), and high (0.3–10 GHz) frequency. Well-known radio navigation systems are DECCA (operated by some European governments), Omega (developed by the U.S. Navy Submarine Service), and LORAN-C (operated by the U.S. Coast Guard).

Indoor navigation systems generally use infrared and short-range radios. The mobile networking community uses a technique known as Cell Identification (Cell-ID).

3.5 LOCATION UTILITY SERVICE

The OpenLS utilities specification provides two services: geocoder and reverse geocoder, and an abstract data type named Address. The geocoder service is a network-accessible service that transforms the description of a location into a normalized description of the location with point geometry. Conversely, the reverse geocoder service maps a given positioning into a normalized description of a feature location.

3.5.1 Geocoding

Geocoding is the process of assigning an *x*, *y* coordinate (e.g., latitude, longitude) to a given address. Once such point geometry is computed, the given address can be displayed on a map. "Address interpolation" is a well-known geocoding technique. Given a street segment with start and end coordinates, and an associated address range (e.g., a tuple from the Road_segment table defined in the Logical Data Modeling section), we can interpolate the (approximate) location of any given address that falls within the given range by simply dividing the length of the road segment by the number of houses. In case of ambiguities, the approximate location can be computed as the centroid of the zip code.

3.5.2 Reverse Geocoding

As the name suggests, reverse geocoding is exactly the opposite of geocoding (i.e., find the address given an *x*, *y* coordinate). Reverse geocoding occurs virtually all the time; find an address (e.g., a landmark, a restaurant) given my current location. But because the coordinates predicted by the GPS receiver contain errors, we need to identify the most likely segment of the road network given the predicted location. This task is commonly known as *map matching*. Map-matching techniques can be broadly classified as geometric, probabilistic, and fuzzy.

GEOMETRIC

The geometric techniques utilize only the predicted location(s) and the road segments. The well-known geometric techniques are point-to-point matching, point-to-curve matching, and curve-to-curve matching. In point-to-point matching, the objective is to find the closet node n_i to the measured position p (e.g., the location predicted by GPS). Generally, the Euclidean distance is used to find the distance between p and n_i. The number of nodes n_i is quite large in a road network; however, this number can be reduced using a range query with a suitable window size and the appropriate spatial access method (e.g., R-tree, CCAM). In point-to-curve matching, the objective is to find the closest curve from the measured point. Here we find the minimum distance between a

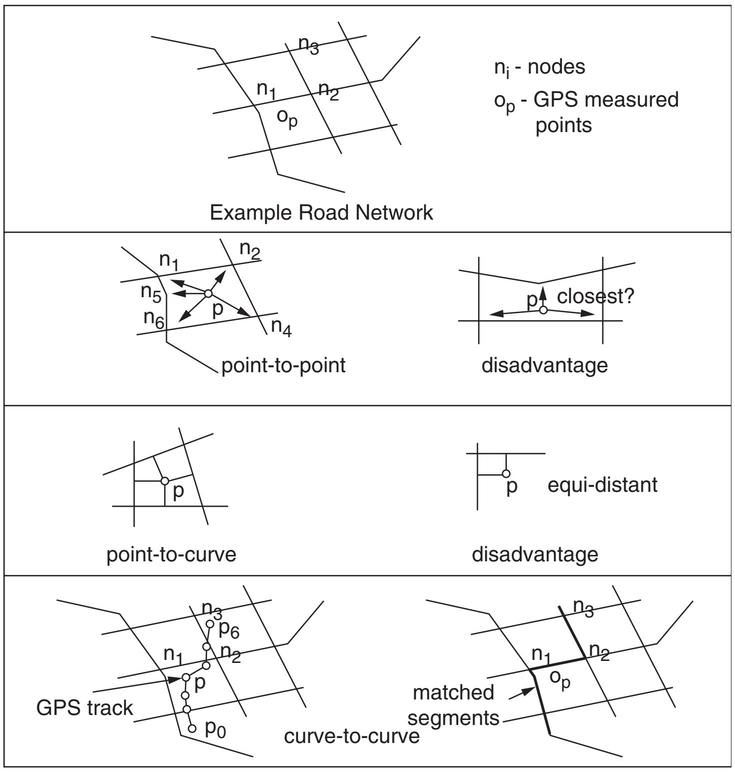


Figure 3.5 Geometric map-matching techniques.

p and the line segments l_i. Both of these methods have limitations. A more accurate geometric method, curve-to-curve matching, uses the piecewise linear curve (generated by connecting the points predicted by a GPS) to find the closest line segment. These methods are summarized in Figure 3.5, and more details on these methods can be found in [BK96].

PROBABILISTIC

Using sensor-specific error models, the probabilistic algorithms first compute a confidence region along the measured track (e.g., GPS track). A map overlay (or spatial join) of this estimated region with the road network layer gives the road segment on which the vehicle is traveling. If more than one road segment is found within the estimated region, however, then the most probable road segment is estimated using various checks (e.g., road network topology, history). More rigorous probabilistic models for map matching can be found in [PSS01;KJL00].

FUZZY LOGIC

Expert rules, such as amount of distance traveled and directional changes in the heading of the vehicle, are assigned fuzzy membership functions. A map-matching module then evaluates the sensor (GPS) measurements and road networks to find the matching segment and position of the vehicle. More details on probabilistic and fuzzy map matching can be found in [Zha97].

3.6 DIRECTORY SERVICE

The directory service of location-based services provides a search capacity for one or more Points of Interest (POI). A POI is a place, product, or service with a fixed position, typically identified by name rather than by address and characterized by type. A POI may be used as a reference or a target point in many query types (see Table 3.3). The query types can be divided into two types: (1) attribute queries, based on nonspatial attributes, and (2) proximity queries, based on spatial attributes. An attribute query is subdivided into a unique attribute query or a property attribute query. The unique attribute query amounts to a pinpoint White Pages query, which constrains the request by the identifier. The property attribute query is a normal Yellow Pages query constrained by nonunique attributes. Attribute queries are well supported by the query-processing methods of traditional relational databases.

Proximity queries are based on spatial objects and are divided into three types: point queries, range queries, and nearest neighbor queries.

3.6.1 *Point Query (PQ)*

Given a query point P, find all spatial objects O that contain it:

$$PQ(p) = \{O \mid p \in O.G \neq \phi\}$$

where $O.G$ is the geometry of object O. The spatial query-processing method of a point query can be used by a spatial index. First, in the filter step, the spatial objects are represented by simpler approximations such as the MBR. Determining whether a query point is in an MBR is less expressive than checking if a point is in an irregular polygon. The spatial operator, contain, can be approximated using the overlap relationship among corresponding MBRs. In the refinement step, the exact geometry of each element from the candidate set is examined.

3.6.2 *Range Query (RQ)*

Given a query polygon *P*, find all spatial objects *O* which intersect *P*. When the query polygon is a rectangle, this query is called a *window query*.

$$RQ(p) = \{O | O.G \cap P.G \neq \phi\}$$

If records are ordered using a space-filling curve (say Z-order), then the range of Z-order values satisfying the range query is determined. A binary search is used to get the lowest Z-order within the query answer. The data file is scanned forward until the highest Z-order satisfying the query is found. Range query can also be processed in a top-down recursive manner using spatial index structures (e.g., R-tree). These methods work in the same manner as in the point query. The query region is tested first against each entry (MBR, child-pointer) in the root. If the query region overlaps with MBR, then the search algorithm is applied recursively on entries in the R-tree node pointed to by the child-pointer. This process stops after reaching the leaves of the R-tree. The selected entries in the leaves are used to retrieve the records associated with the selected spatial keys.

3.6.3 *Nearest Neighbor Query (NNQ)*

Given a query point *P*, find the spatial object *O* with the smallest distance to *P*:

$$NNQ(p) = \{O | dist(O.G, P.G) \leq dist(O'.G, P.G)\}$$

Here $O'$are all other spatial objects except *O*. The most common type of nearest neighbor search is the point *k*-Nearest Neighbor (KNN) query, which finds the *k* point objects that are closest to a query point. Conceptually, one strategy for the nearest neighbor query applies to a two-pass algorithm. The first pass retrieves the data page *D* containing query object *P* to determine *dist*, the minimum distance between any objects in *D* to *P*. The second pass is a range query to retrieve objects within distance *dist* of *P* for determination of the nearest object. This approach reuses the spatial index algorithm for spatial selection (e.g., point query and range query). Most of the current research on KNN query is based on utilizing different spatial index structures such as R-trees or Quad-trees. The representative algorithm in a branch-and-bound manner was proposed originally by [RKV95]. The algorithm

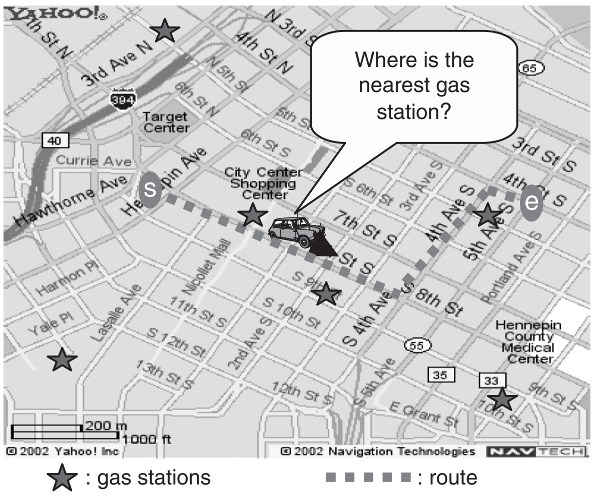

Where is the nearest gas station?
s
e
★ : gas stations
■ ■ ■ ■ ■ : route
© 2002 Yahoo! Inc
© 2002 Navigation Technologies

Figure 3.6 An example of a nearest neighbor query.

traverses a spatial index tree in a depth-first manner to visit entries with a minimum distance from a query point. A similar technique for moving query point is described in [SR01].

Figure 3.6 shows a typical example of NNQ on a road network. Consider a situation in which a mobile user is on his way to a destination and wants to find the nearest gas station close to the route. Some recent studies have proposed NNQ techniques to solve this problem. Figure 3.7 shows four different ways to find the nearest neighbor on a road network. In the figure, the circle represents a current location on the route and the large diamond represents the nearest neighbor found by each method. Each method generates a different nearest neighbor. In Figure 3.7(a), the route is regarded as several consecutive line segments. [TPS02] proposes a continuous nearest neighbor search method for a query point (current location of the user) that is moving on a trajectory. This approach generates a result consisting of a set of <point, interval> tuples in which a point is the nearest neighbor in the corresponding interval. Figure 3.7(b) shows a neighborhood query generation model that takes account of the current position and the past/future trajectories of the moving points [IKK02]. Most NNQ methods use Euclidean distance as the distance measure. In LBS, however, especially those that deal with spatial network databases (e.g., road networks), the Euclidean distance may not properly approximate the real road-distance. Figure 3.7(c) presents [FW02]'s method for searching the nearest neighbor from a

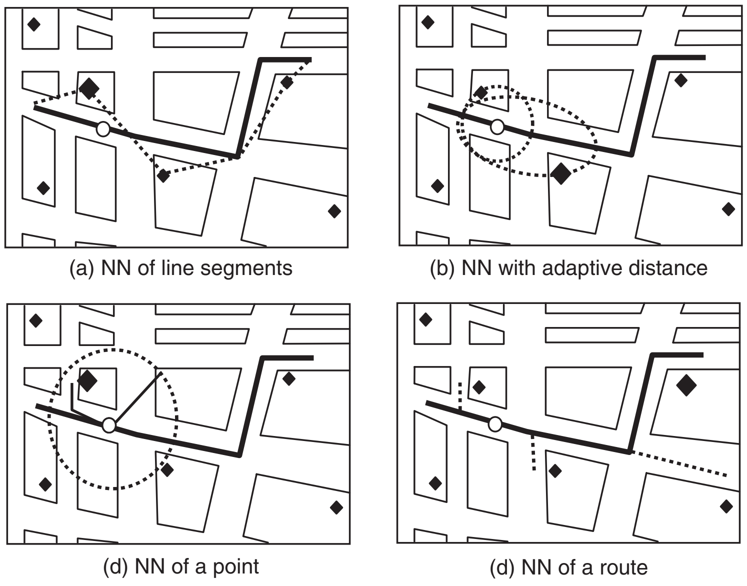


Figure 3.7 Nearest neighbor methods illustrated on a road network.

query point on the road network. The algorithm consists of two interactive steps: a filtering step using Euclidean distance and a refinement step using road-distance.

Figure 3.7(d) shows another variation of NNQ [SY03], which tries to find the nearest neighbor having a minimum detour length from the predetermined route. The nearest neighbor in Figure 3.7(d) is very different from the other nearest neighbors in Figure 3.7. One approach for this problem uses an approximation method of finding the closest pair [CMTV00] between two spatial data sets (spatial object points and intersect points of the route), where each set is indexed by an R-tree, and rechecked it with the road-distance. Another approach uses an ‘‘Allocate’’ operation, which divides the road network into service areas of a given set of spatial objects. For all points P in the service area of a spatial object O_i, road_distance(P,O_i) $<=$ road_distance(P,O_j), $\forall_{j \neq I}$, only those service areas that intersect with a given route are considered for determining the nearest neighbor.

Most queries are composed from a fixed set of basic operations. These basic relational operations form the building blocks for the composition of all complex queries. Query processing maps high-level queries into a composition of basic relational operators and then optimizes them.

3.7 ROUTE DETERMINATION SERVICE

Route determination services address finding route and navigation information between locations. Route determination should support the following two services. The first deals with the determination of a new route; given a start location, end location, optional waypoints, and a set of route criteria, find the best path. Possible criteria are fastest, shortest, easiest, pedestrian, public transportation, avoid locations/areas, avoid highways, avoid tollways, avoid U-turns, and avoid ferries. The second service deals with the determination of alternate routes. The new (alternate) route should have minimal overlap with the existing route. After determining the route, returned combined information are route summary information, route maneuver and advisory information, route geometry, maps of the route and maneuvers, and turn-by-turn instructions, in presentation format.

3.7.1 Path-Query Processing

Path-query processing is an important ingredient in spatial network applications. Support for navigation, route planning, and traffic management essentially reduces to providing *path options* based on some application-dependent criterion. For example, a well-known graph operation is determining the "shortest" path between two points *A* and *B* on a road network where the "shortest" criterion could be based on distance, travel time, or some other user-specified constraint. Underlying the computation of all path queries are *graph traversal* algorithms, which search for paths by traversing from one node to another along the edges of a graph. As we have seen before, searching for paths is a recursive operation, and therefore the adjacency lists of nodes have to be repeatedly transferred from secondary storage to the main memory buffer. Graph traversal algorithms form the backbone of all path computation algorithms. Examples of well-known graph traversal algorithms are breadth-first, depth-first, and Dijkstra's and best-first A*. The description of breadth-first and depth-first search algorithms can be found in any basic data structures textbook. Memory-bounded and hierarchical search algorithms are described in the next subsection. Examples showing how all of these algorithms work are provided at the end of this section.

DIJKSTRA'S ALGORITHM

Dijkstra's algorithm can be used to solve the single-source (partial transitive closure) problem. Given a source node v, Dijkstra's algorithm will compute the shortest path from the source node v to all other reachable nodes using a best-first search where frontier nodes are ranked by their path lengths to the source. Dijkstra's algorithm is a classic shortest-path search algorithm that can be found in many books, including [SC02].

BEST-FIRST A* ALGORITHM

Best-first search has been a framework for heuristics that speed up algorithms by using semantic information about a domain. A* (A-star) is a special case of the best-first search algorithm. The cost function from node s to d is of the form $\mathrm{cost}(s, d) = g(s, v) + h(v, d)$, among which *cost*$(s, d)$ is total cost, $g(s, v)$ is the cost from s to v, and $h(v, d)$ is the estimated cost from v to d. It uses an estimator function $h(v, d)$ (also known as f-cost) to estimate the cost of the shortest path between nodes *v* and *d*. The A* search without estimator functions is not very different from Dijkstra's algorithm. The pseudo-code is shown in Figure 3.8. The procedure terminates when it finds destination node d as the best node.

```
procedure A*(G(V,E),v,d,f);
{
    var: integer;
    foreach  u in V do {if (v,u) is edge then g(v,u) = edge(v,u) else g(v,u) = inf;
    g(v,v) = 0; path(v,u):= null}
    frontierSet := [v]; exploredSet := emptySet;
    while not_empty(frontierSet) do
    {
              select w from frontierSet with minimum(g(v,w)+ h(w,d));
              frontierSet := frontierSet - [w]; exploredSet := exploredSet + [w];
              if(u = d) then terminate
              else {
                        fetch( w.adjacencyList);
                        foreach < u, g(w,u)> in w.adjacencyList
                        if g(v,u) > g(v,w) + edge(w,u) then
                        {
                                  g(v,u) := g(v,w) + edge(w,u);
                                  path(v,u) := path(v,w) + (w,u);
                               if frontierSet ∪ exploredSet ∈ u then
                                            frontierSet := frontierSet + [u];
                        }
                 }
}
}
```

Figure 3.8 Best-first A*.

This procedure can terminate quickly if the shortest path from s to d has fewer edges. It does not have to examine all nodes to discover the shortest path, as in the case of many other algorithms (e.g., Dijkstra). Furthermore, the estimator can provide extra information to focus the search on the shortest path to the destination, reducing the number of nodes to be examined. The best-first A* search algorithm is complete and optimal.

MEMORY-BOUNDED SEARCH ALGORITHMS

Previously introduced algorithms assume that the system has unlimited memory that can hold all information used in these search algorithms. In reality, however, many systems have memory limitations, so we need algorithms that work with given memory bounds.

Let us consider a search tree of maximum depth m and branching factor b. Let us also assume that a solution (destination node) can be found at depth d. The time and space (memory) requirements for simple breadth-first and depth-first search algorithms are $O(b^d)$, $O(b^d)$ and $O(b^m)$, $O(bm)$, respectively. It is easy to see that the memory requirements are much higher as the problem size increases for breadth-first as compared to depth-first search algorithms. Although the depth-first search has modest memory requirements, it may get stuck going down the wrong path. This pitfall can be avoided by limiting the depth of the search path. Finding a good depth limit is not an easy task, however, and this limitation has led to the development of the iterative deepening search algorithm. The iterative deepening search algorithm combines the benefits of depth-first and breadth-first algorithms; it is optimal, complete, and has modest memory requirements $O(bd)$. We now present two algorithms, IDA* and SMA*, that are designed to work with modest memory requirements.

IDA*

The IDA* algorithm is a logical extension of the iterative-deepening search algorithm. The algorithm is similar to best-first A* presented previously; however, instead of a best-first search strategy, we use an iterative-deepening search. The algorithm [RN95] shown in Figure 3.9 proceeds in the same manner as depth-first; however, it uses a cost function (f-cost) to limit the search depth, rather than a fixed depth-limit. The f-cost of a node is given by $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the start node to node n and $h(n)$ is the estimated cost of

```
function IDA* (problem) returns a solution sequence
  inputs: problem, a problem
  local variables: f-limit, the current f-cost limit
    root, a node
    root ←MAKE-NODE(INITIAL-STATE[problem])
    f-limit ← f-COST (root)
  loop do
    solution,f-limit ← DFS-CONTOUR(root,f-limit)
    if solution is non-null then return solution
    if f-limit = ∞ then return failure; end

function DFS -CONTOUR (node, f-limit) returns a solution sequence and a
  new f- COST limit
  inputs: node, a node
      f-limit, the current f-COST limit
  local variables: next-f , the f-COST limit for the next contour, initially ∞
  if f-COST [node] > f-limit then return null, f-COST [node]
  if GOAL-TEST [problem] (STATE[node]) then return node, f-limit
  for each node s in SUCCESSOR (node) do
    solution, new-f ←DFS-CONTOUR (s, f-limit)
  if solution is non-null then return solution, f-limit
    next-f ← MIN (next-f,new-f);end
  return null, next-f
```

Figure 3.9 IDA* algorithm.

the path from the node *n* to the destination node. First, each iteration expands all of the nodes inside the contour for the current f-cost. If a solution is not found, then the search extends over to the next contour line (f-cost). Once the search inside a given contour is complete, a new iteration is started, using a new f-cost for the next contour.

IDA* is complete and optimal under the same conditions as A*; however, it requires memory proportional to the longest path it explores. Although bd, branching factor time depth, is a good estimate of the storage requirement in most cases, IDA* suffers from duplicate computations because it remembers only the current cost between iterations.

SMA*

The simplified memory-bounded A* algorithm tries to avoid the duplicate computations of IDA* by remembering as much history as the memory permits and not just the f-cost, as in the case of IDA*. If there is no memory left and the algorithm still needs to generate a successor, the most unpromising node (i.e., the shallowest and highest f-cost node) is dropped from the queue. The SMA* algorithm is optimal and complete if enough memory is available; otherwise, it returns the best

```
Algorithm SMA* (start):
OPEN = {start};
USED ← 1;

loop
        if empty(OPEN) return FALSE;
        best ← deepest least-f-cost leaf in OPEN;
        if ( d = best ) return TRUE;
        u ← next-successor(best);
        f(u) ← max( f(best), g(u) + h(u));
        if (completed(best)), BACKUP(best);
        if (S(best) all in memory, remove best from OPEN.
        USED ← USED + 1;

        if (USED > MAX) then
                delete shallowest, highest-f-cost node in OPEN;
                remove it from its parent's successor list;
                insert its parent on OPEN if necessary;
                USED ← USED - 1;
        endif
        insert u in OPEN.
end of loop

Procedure BACKUP(n)
if n is completed and has a parent then
        f(n) ← least f-cost of all successors;
        if f(n) changed, BACKUP(parent(n)).
```

Figure 3.10 The SMA* search algorithm.

solution that can be found using the given memory. A simplified SMA* algorithm [Rus92] is shown in Figure 3.10. SMA* uses a binary tree of binary trees data structure to store the current node (OPEN) sorted by f and depth, respectively. MAX is a global variable used to represent the maximum number of nodes that can be fit into memory, and the USED variable is used to keep track of how many nodes are currently in memory. Each node contains its g, h, and f-costs, as well as the minimum f-cost of its examined successors. S(n) denotes n's successor list. A node with no unexamined successors is called complete.

HIERARCHICAL STRATEGIES

Hierarchical algorithms decompose a large spatial graph into a boundary graph and a collection of fragment graphs, each of which is much smaller than the original graph. Hierarchical graphs are particularly useful in reducing input/output (I/O) costs and main-memory buffer requirements for processing queries on graphs that are too large to fit inside the main-memory buffers.

The basic idea of a hierarchical algorithm for computing a shortest path is to decompose the original graph into a set of smaller-fragment graphs and a summary graph called a *boundary* graph. Proper construction of the *boundary* graph allows an optimality, preserving decomposition of the shortest path query on the original graph into a set of shortest path queries on the smaller graphs.

The hierarchical graph has a two-level representation of the original graph. The lower level is made up of a set of fragments of the original graph. The higher-level graph consists of the boundary nodes and is called the boundary graph. Boundary nodes are defined as the set of nodes that have a neighbor in more than one fragment, i.e.,

$$N_i \in BN \Leftrightarrow \exists E_{i,j}, E_{i,k} | FRAG(k) \neq FRAG(j)$$

Edges in the boundary graph are called boundary edges, and the boundary nodes of a fragment form a clique (i.e., they are completely connected). The cost associated with the boundary edge is the shortest-path cost through the fragment between the boundary nodes. A boundary edge is associated with a fragment identifier. A *boundary path* is the shortest path through the boundary graph.

The hierarchical algorithm is composed of three steps: (1) finding the relevant boundary-node pair in the boundary graph, (2) computing the boundary path, and (3) expanding the boundary path. The first step in determining the shortest path is to compute the boundary node through which the shortest path leaves the source's fragment and enters the destination's fragment. If both the source and destination are boundary nodes, then the algorithm is trivial. If the source is an internal node and the destination is a boundary node, then the boundary node through which the shortest path leaves the source's fragment is found by querying the fragment graph for the cost of the path from the source to all boundary nodes of that fragment, and by querying the boundary graph for the cost of the shortest path from all boundary nodes of the source's fragment to the destination. The source-boundary-destination path with the lowest aggregate cost determines the appropriate boundary node.

The case where the source is a boundary node and the destination is an internal node is similar, but the roles of the source and destination are reversed. When both the source and destination are internal nodes, the appropriate boundary node pair is found by querying the fragment graphs to determine the cost of the shortest path from the internal nodes to all boundary nodes of the fragment. Next, the boundary graph is queried to compute the shortest-path cost between all pairs of boundary nodes. The path with the lowest aggregate cost determines the boundary-node pair. Once the appropriate boundary-node pair has been determined, the boundary graph is queried to determine the shortest path between those boundary nodes. The final step is to expand the boundary path by querying the fragments for the shortest path through them. Adjacent nodes in the boundary path form source/destination pairs on which the shortest-path query can be run on in a fragment. For more details and related techniques, see [JHR95; JHR98; JP02; JSQ00].

We now briefly analyze how each algorithm described in the previous section performs on the example graph shown in Figure 3.4(a). Let us assume that the source node is 1 and the goal node is 9. The estimated cost h from a given node *n* to the goal node 9 is shown in Table 3.4.

In the first iteration, Dijkstra's algorithm explores edges (1,2) and (1,3) and then selects node 2 (as cost(1,2) < cost(1,3)) and set cost $g(1,2) = 3$. In the next iteration, it picks node 3 (as the [cost(1,2) + cost(2,4)] > [cost(1,3) + cost(3,10)] and [cost(1,2) + cost(2,3)] > [cost(1,3) + cost(3,10)]). Applying the same logic, it examines other nodes from adjacency lists and puts them into the frontier set. This process continues ÿmpro the algorithm finds the shortest path from node 1 to node 9.

On the other hand, best-first A* uses an improved heuristic cost function, which also considers the cost between the current node and the

Table 3.4 Estimated cost to goal node (9).

Node (*n*)	***h*(*n*)**
1	8
2	5
3	9.8
4	6.4
5	5
6	4.5
7	4.1
8	3
10	10.3

Table 3.5 Summary of path-finding results.

Algorithm		Solution
Dijkstra		1,2,4,5,6,7,9
A*		1,2,4,5,6,7,9
IDA*	f-limit = 16	1,2,4,5,6,7,9
SMA*	Mem = 5	No solution
	Mem = 6	1,2,4,5,8,9
	Mem = 7	1,2,4,5,6,7,9
Hierarchical (with A*)		1,2,4,5,6,7,9

destination node. At first iteration, it picks up node 2 (because the [g(1,2) + h(2,9)] < [g(1,3) + h(3,9)]). It then examines nodes 3 and 4, which are neighbors of node 2. The cost through node 4(g(1,4) + h(4,9) < g(1,3) + h(3,9)) is minimum to reach node 9. So as compared to Dijkstra's, A* will not expand node 3 at this iteration. Applying the same logic, best-first A* examines all necessary nodes in the following iterations. Finally, it will find the shortest path from node 1 to node 9. In the case of the IDA* algorithm, we first set the contour line to be the f-limit of h(1,9), which is 8. In the first iteration, we find a new f-limit by finding the minimum f-cost that is greater than the current f-limit. This means the search expands from node 1 to node 2 because g(1,2) + h(2,9), which gives the new contour line, is less than g(1,3) + h(3,8). Applying similar logic, in the next iteration we find the minimum cost of the path through 1-2-4 as the new f-limit. Finally, we get path 1-2-4-5-6-7-9 as the optimal path.

To illustrate how SMA* works, we have chosen three memory bounds of 5, 6, and 7 nodes, respectively. If we can store information for only one node (i.e., node 1) and node 1 is not the destination node, we stop. When we have enough memory to store information for two nodes, then we can expand the search from node 1 to nodes 2, 3, and none of these are destination nodes. Because we cannot find a path to the destination node, we stop. As summarized in Table 3.5, we cannot find a solution up to a memory bound of 5; however, for memory bound 6, we do find a solution, although not an optimal one. For a memory bound of 7, we find an optimal path.

In the case of the hierarchical strategy, we first cut the graph between nodes 4 and 5 to get two fragments, {1, 2, 3, 4, 10} and {5, 6, 7, 8}. The boundary graph consists of nodes 4 and 5, and the edge (4,5). Now finding the optimal path reduces to finding the optimal paths in these two subgraphs and that pass though the boundary nodes; that is, path(1,4) + path(4,5) + path(5,9). Using A* in the first fragment results in the optimal path of 1-2-4, and in the second fragment results in 5-6-7-9. The global optimal path is obtained by combining these subpaths. Table 3.5 summarizes the results of applying all of these algorithms on the network graph shown in Figure 3.4(a).

3.8 PRESENTATION SERVICE

Presentation services display road maps and overlay routes, points of interest, object locations, and/or text information such as route descriptions on a road map. Currently, most presentation services are provided based on a visual interface framework; however, in the future a voice-based user interface will likely be adopted, especially in in-vehicle navigation systems, to help drivers who are already overloaded with driving tasks. Apart from easy-to-use visual and audio interfaces, presentation service requires efficient route-guidance algorithms to dynamically process and present the guidance instructions.

Route guidance is the process that guides travelers along a route either by prepared printouts of the desired route in pretrip guidance or by output of an en route guidance module in real time. In either case, a route-planning module and a positioning system are required. When using prepared printouts or maps for pretrip guidance, an explicit route-planning module is not required, but some route-planning function should be executed before the traveling route has been acquired. Nowadays, en route guidance is a desirable feature for in-vehicle navigation systems. These display simple (visual or auditory) icons to advise drivers of forthcoming actions (e.g., right/left-turn ahead) in real-time. The idea is to convey route information to drivers that is relevant to the next few minutes of driving based on current position and without distracting drivers from driving tasks. Various guidelines for designing in-vehicle information systems with applications to route guidance can be found in [GLPS93].

There are two kinds of en route guidance models: a centralized model and a distributed model. In a centralized model, the traveler communicates with a management center, which traces the traveler's location, speed, and other information. It is the management center's responsibility to compute the route and broadcast this information to the traveler. In a distributed model, the route computation is performed by the guidance unit at the hand of the traveler; such guidance units require high computation ability.

In en route guidance, static route guidance assumes that the travel cost, which includes travel distance, travel time, and minimum turn, is static. In real situations, however, the travel cost varies at different times. Dynamic route-guidance systems consider the changing situation, calculate the travel cost on-the-fly with dynamic information, and recommend new routes to the traveler.

3.9 CONCLUSION

Earlier navigation systems, which were limited to simple positioning devices and static paper maps (e.g., road maps, navigational charts), have evolved into much more sophisticated navigation systems comprising satellite-based precise positioning systems (GPS)-enabled portable digital assistants. These devices have local memories to support small digital maps and have wireless communication ports for getting dynamic spatial information from remote back-end SDB servers. Spatial databases play a central role in modern location-based applications. Location-based services will not achieve their full potential unless there is a cohesiveness between disparate components and conformance with open standards. Recent industry trends show that key players in this sector (ESRI's ArcIMS, Intergraph's IntelliWhere, MapInfo, Cquay, Webraska) are developing interfaces to standard open platforms, such as OpenLS.

The current portable PDAs have limited memories and display units. These limitations dictate the need for efficient main memory spatial processing algorithms and intelligent user interfaces. Emergency applications, which require real-time dynamic spatial data from remote SDB servers, are limited by the limited bandwidth provided by present wireless communication devices. In order to reduce the amount of information transferred over networks, we need efficient compression techniques. Additional research is needed to progressively transmit the data based on importance. These research needs are summarized in Table 3.6.

Table 3.6 Research need in modern navigation systems.

Navigation System Component		Research Needs
Server	Gateway	Indoor location sensing 100% coverage of location sensing despite GPS shadows
	Location utility	Improving map accuracy Improving effectiveness of map matching using additional information such as long-term and short-term histories
	Directory	Nearest neighbor (e.g., facility) to a route (segment)
	Route determination presentation	Alternate paths Safe visual and audio interfaces Cartographic generalization Adaptive (client-specific) result generation
Client	PDAs	Improving memories, display sizes, processing power Computing under limited resources Smart caching, prefetching
Communications		Improving bandwidths Efficient map compression algorithms Progressive transmissions

ACKNOWLEDGMENTS

This work was partially supported by Army High Performance Computing Research Center contract number DAAD19-01-2-0014 and NSF Grant EIA-0224392. The content of this work does not necessarily reflect the position or policy of the government, and no official endorsement should be inferred. We also gratefully acknowledge the support received from the Minnesota Department of Transportation. We would like to thank spatial database research group members for their valuable comments and input. We would also like to express our thanks to Kim Koffolt, whose comments have improved the readability of this paper.

References

[AGS$^+$93]	N. I. Adams, R. Gold, B. N. Schilit, M. M. Tso, and R. Want. "An Infrared Network for Mobile Computers." In *Proc. USENIX Symp. On Mobile and Location Independent Computing*. Cambridge, Massachusetts, 1993.
[ALK]	ALK Incorporation, *http://www.alk.com*
[ASPRS]	American Society for Photogrammetry and Remote Sensing, "ASPRS Interim Accuracy Standards for Large-Scale Maps," *http://www.asprs.org/asprs/resources/standards.html*
[Bon93]	G. Bonsiepe. "Interpretations of Human User Interface." *Visible Language*, 24(3):262–285, 1993.
[CMTV00]	A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. "Closest Pair Queries in Spatial Databases." ACM SIGMOD, 2000.
[ComSoc]	IEEE Communications Society. "Communications History." *http://www.ieee.org/organizations/history_center/comsoc/techhist.html*
[DB2Spatial]	IBM. "DB2 Spatial Extender for Linux, UNIX and Windows." *http://www.306.ibm.com/software/data/spatial/*
[BK96]	D. Bernstein and A. Kornhauser. "An Introduction to Map Matching for Personal Navigation Assistants." The New Jersey TIDE Center's technical report, 1996, *http://www.njtide.org/reports/mapmatchintro.pdf*
[EN01]	R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*, with E-book, 3rd edition. Addison-Wesley, New York, 2001.
[ESRI]	Environmental Systems Research Institute, ArcSDE, *http://www.esri.com/software/arcgis/arcinfo/arcsde/index.html*

[ETAK] Tele Atlas, Products and Services, *http://www.na.teleatlas.com*

[FGDC01] Federal Geographic Data Committee, *http://www.fgdc.gov,* Geospatial Metadata, April 2001

[FW02] J. Feng and T. Watanabe. "Fast Search of Nearest Target Object in Urban District Road Networks." Pan-Yellow-Sea International Workshop on Information Technologies for Network Era (PYIWIT), 2002.

[GDT] Geographic Data Technology, Product Catalog, *http://www.geographic.com*

[GLPS93] P. Green, W. Levison, G. Paelke, and C. Serafin. *Preliminary Human Factors Design Guidelines for Driver Information Systems*. The University of Michigan, Ann Arbor, MI, Transaction Research Institute, Technical Report No. UMTRI-93-21.

[GSI93] Geographical Survey Institute, "Digital Map Series: Digital Map 2500 (Spatial Data Framework)," 1993, Japan.

[Guting94] R. H. Guting. "An Introduction to Spatial Database Systems." *VLDB Journal*, 3:357–399, 1994.

[HHS+99] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. "The Anatomy of a Context-Aware Application." In *Proc. 5th Annual ACM/IEEE Intl. Conf. On Mobile Computing and Networking* (Mobicom), 1999.

[IKK02] Y. Ishikawa, H. Kitagawa, and T. Kawashima. "Continual Neighborhood Tracking for Moving Objects Using Adaptive Distances." *IDEAS*, 2002.

[JHR95] N. Jing, Y. W. Huang, and E. Rundensteiner. "Hierarchical Path Views: A Model Based on Fragmentation and Transportation Road Type." In *Proc. 3rd ACM Workshop on Geographic Information Systems*, November 1995.

[JHR98] N. Jing, Y. W. Huang, and E. Rundensteiner. "Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation." *IEEE Transactions on Knowledge and Data Engineering*, 10(3), May/June 1998.

[JP02] S. Jung and S. Pramanik. "An Efficient Path Computation Model for Hierarchically Structured Topological Road Maps." *IEEE Transactions on Knowledge and Data Engineering*, 14(5), Sept/Oct 2002.

[JSQ00] G. R. Jagadeesh, T. Srikanthan, and K. H. Quek. "Heuristic Techniques for Accelerating Hierarchical Routing on Road Networks." *IEEE Transactions on Intelligent Systems*, 3(4), December 2000.

[KJL00] W. Kim, G. I. Jee, and J. G. Lee. "Efficient Use of Digital Road Map in Various Positioning for ITS." In *Proc. Position Location and Navigation Symposium, IEEE*, 2000.

[Mac96] V. Machiraju. *A Survey on Research in Graphical User Interfaces*. Department of Computer Science, University of Utah, August 1996.

[MNDOT] Minnesota Department of Transportation, Basemap, *http://rocky.dot.state.mn.us/basemap/*

[NAVTECH] Navigation Technologies, *http://www.navtech.com*

[NMAS] U.S. Geological Survey. "National Map Accuracy Standards." *http://rockyweb.cr.usgs.gov/nmpstds/nmas.html*

[NMEA] National Marine Electronics Association, The 0183 Standard, *http://www.nmea.org*

[OGC98] OGC, "Simple Features Specification (for OLE/COM, CORBA, SQL)", *http://www.opengis.org/techno/specs.htm*

[OpenLS] OGC, "Open Location Services Initiative (OpenLS)", *http://www.openls.org/about.htm*

[PCB02] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. "The Cricket Location-Support System." In *Proc. Sixth ACM Intl. Conf. On Mobile Computing and Networking*, 2002.

[PDMD03] Graticule, "Philips – Digital Map Data," 2003, *http://www.graticule.com*

[PSS01] J. Pyo, D. H. Shin, and T. K. Sung. "Development of a map matching method using the multiple hypothesis technique." In *Proc. 2001 IEEE Intelligent Transportation Systems Conference*, pages 23–27, 2001.

[Rab95] L. R. Rabiner. "The Impact of Voice Processing in Modern Telecommunications." *Speech Communications*, 17(3–4):217–226, November 1995.

[RKV95] N. Roussopoulos, S. Kelleym, and F. Vincent. "Nearest Neighbor Queries." In *Proc. 1995 ACM SIGMOD Intl. Conf. On Management of Data*, pages 71–79, ACM Press, 1995.

[RN95] S. Russell and N. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[RSV01] P. Rigaux, M.O. Scholl, and A. Voisard. Spatial Databases: With Applications to GIS. Morgan Kaufmann Publishers, 2001.

[Rus92] S. Russell. "Efficient Memory-Bounded Search Methods." In *Proc. 10th European Conf. On Artificial Intelligence*, pages 1–5, Wiley.

[SC02] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, Englewood Cliffs, NJ, 2002.

[SCR+99] S. Shekhar, S. Chawla, S. Ravada, A. Fetterer, X. Liu, and C. T. Lu. "Spatial Databases—Accomplishments and Research Needs." *IEEE Transactions on Knowledge and Data Engineering*, 11(1):45–55, 1999.

[SDC] Oracle, "Oracle Spatial," *http://otn.oracle.com/products/oracle9i/datasheets/spatial/spatial.html*

[SL97] S. Shekhar and D. R. Liu. "CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations." *IEEE Transactions on Knowledge* *and Data Engineering*, 9(1), January 1997.

[SR01] Z. Song and N. Roussopoulos. "K-Nearest Neighbor Search for Moving Query Point," In *Proc. 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, 2001.

[ST91] J. W. Sullivan and S. W. Tyler (Eds.). *Intelligent User Interfaces*. ACM Press, New York, 1991.

[SVCB99] S. Shekhar, R. R. Vatsavai, S. Chawla, and T. E. Burk. "Spatial Pictogram Enhanced Conceptual Data Models and Their Translation to Logical Data Models." Integrated Spatial Databases, Digital Images, and GIS, *Lecture Notes in Computer Science*, Vol. 1737, 1999.

[SY03] S. Shekhar and J. S. Yoo. "Processing In-Route Nearest Neighbor Queries: A Comparison of Alternative Approaches." In *Proc. 11th ACM Intl. Symp. On Advances in Geographic Information Systems*, ACM-GIS 2003.

[TIGER] U.S. Census Bureau, **T**opologically **I**ntegrated **G**eographic **E**ncoding and **R**eferencing system, *http://www.census.gov/geo/www/tiger/index.html*

[TPS02] Y. Tao, D. Papdias, and Q. Shen. "Continuous Nearest Neighbor Search." In *Proc. 28th Very Large Data Bases Conference,* 2002.

[VW01] M. Vazirgiannis and O. Wolfson. "A Spatiotemporal Model and Language for Moving Objects on Road Networks." In *Proc. 7th Intl. Symp. On Spatial and Temporal Databases*, SSTD, 2001.

[WHFG92] R. Want, A. Hopper, V. Falcao, and J. Gibbons. "The Active Badge Location System." Technical Report 92.1, 1992, ORL, 24a Trumpington Street, Cambridge, CB2, 1QA.

[YLM95] N. Yankelovich, G. A. Levowt, and M. Marx. "Designing Speech Acts: Issues in Speech User Interfaces." CHI 95 Conference on Human Factors in Computing Systems, Denver, CO, May 7–11, 1995.

[Zha97] Y. Zhao. *Vehicle Location and Navigation Systems*. Artech House, 685 Canton Street, Norwood, MA, USA, 1997.