Roads Belong in Databases

Jagan Sankaranarayanan Hanan Samet
Center for Automation Research
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland, College Park, MD 20742
{jagan, hjs}@cs.umd.edu

Abstract

The popularity of location-based services and the need to perform real-time processing on them has led to an interest in queries on road networks, such as finding shortest paths and finding nearest neighbors. The challenge here is that the efficient execution of operations usually involves the computation of distance along a spatial network instead of "as the crow flies," which is not simple. This requires the precomputation of the shortest paths and network distance between every pair of points (i.e., vertices) with as little space as possible rather than having to store the n^2 shortest paths and distances between all pairs. This problem is related to a 'holy grail' problem in databases of how to incorporate road networks into relational databases. A data structure called a road network oracle is introduced that resides in a database and enables the processing of many operations on road networks with just the aid of relational operators. Two implementations of road network oracles are presented.

1 Introduction

The growing popularity of online mapping services such as Google Maps, Microsoft Bing, and Yahoo! maps has led to an interest in responding to queries in real time, such as finding shortest routes between locations along a spatial network as well as finding nearest objects from a set S (e.g., gas stations, markets, and restaurants) where the distance is measured along the shortest path in the network. Elements of S are usually constrained to lie on the network or at the minimum to be easily accessible from the network. The online nature of these services means that responses must be generated in real time. The challenge in performing queries on road networks is that operations involve the computation of distance along a spatial network (i.e., network distance) instead of "as the crow flies," which is not simple.

Operations on road networks [3, 4, 12–22] are expensive because computing distances between two objects (e.g., postal addresses) on the road network requires the invocation of a shortest path algorithm [1, 5, 6, 8, 11, 24]. A popular shortest path algorithm is Dijkstra's algorithm [5], which if invoked between a source vertex q and a destination vertex v, ends up visiting every vertex that is closer to q via the shortest path from q than v.In particular, it is not uncommon for Dijkstra's algorithm to visit a very large number of the vertices of the network in the process of finding the shortest path between vertices that are reasonably far from each other in terms of

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

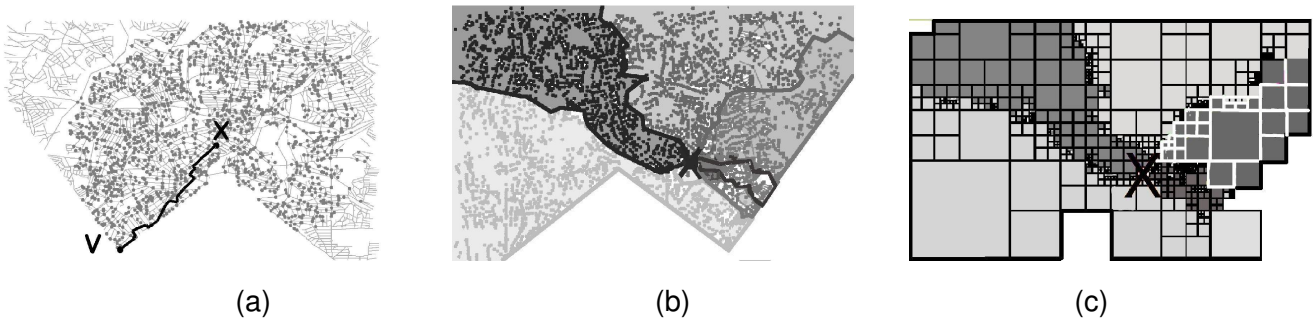

Figure 1: (a) Map of Silver Spring, MD where the highlighted vertices are those visited by Dijkstra's algorithm in determining the shortest path from X to V, (b) partition into regions r_i such that the shortest path from X to a vertex in r_i passes through the same vertex among the six vertices adjacent to X (i.e., the shortest-path map of X), and (c) leaf blocks in the shortest-path quadtree for the regions of the same partition.

network hops. For example, Figure 1(a) shows the vertices that would be visited when finding the shortest path from the vertex marked by X to the vertex marked by V in a spatial network corresponding to Silver Spring, MD. Here we see that in the process of obtaining the shortest path from X to V of length 75 edges, 75.4% of the vertices in the network are visited (i.e., 3,191 out of a total of 4,233 vertices).

Methods such as the *transit node routing* of Bast et al. [1] and the landmark approach of Goldberg and Harrelson [8] as well as [6] can significantly speedup shortest path computations on large road networks, up to several orders of magnitude compared to Dijkstra's algorithm. In spite of these newer approaches being much faster than Dijkstra's algorithm, they still involve searches on graphs during runtime. That is, when the user is waiting for an answer, network distance computation still involves searching a graph space denoted by the road network for a shortest path so that network distances can be obtained. Such a model for performing operations on road networks is unsuitable for building a real-time large scale system because of the computational complexity of computing network distances in road networks.

Therefore, it is not surprising that web services allow users to compute shortest paths, but if presented with any task that is a bit more complicated (such as finding k nearest neighbors), then they resort to using the Euclidean distance (i.e., as the crow flies); but the error due to this approximation is generally unacceptable. If operations on road networks must compute the network distances on the fly, then requiring that the result be obtained in real time (or almost real time) precludes the use of conventional algorithms that are graph-based (e.g., nearest neighbor finding methods such as the INE and IER methods [13] and improvements on them [4]), which usually incorporate Dijkstra's algorithm [5] in at least some parts of the solution [14]. We suggest precomputing of the shortest paths between vertices in a road network as the only way of building scalable services that perform real-time operations on road networks.

Road Networks in Databases: A source of frustration for database researchers with road networks is that they cannot be integrated easily into a relational database system. Understanding why this is important requires a bit more explanation. When we think of road networks, we view them as general graphs as there is an easy transformation from a road network to a general graph equivalent G. The transformation is achieved by casting road intersections as vertices, road segments as edges, and distances in miles or time taken to travel a road segment in the road network as edge weights in G. Furthermore, additional constraints on road networks such as one ways, no left hand turns etc., can be suitably handled by including directions with edges, or making small changes to the topology of the graph equivalent G. Now, operations on road networks are cast as combinatorial operations (i.e., graph operations) on G. From a database point of view, casting operations on road networks as graph operations on G is not a good strategy. In particular, combinatorial operations on G cannot be cast in terms of relational operators (i.e., select, project, join etc.), which constitute the basic operators of any relational database system. As a consequence, operations on road networks cannot be performed in the context of a

database system, which has the unfortunate implied consequence that operations on road networks cannot be expressed using SQL.

Now, why is expressing operations on road networks using SQL important? Relational database systems allow operations to be written in SQL, which is an English-like language. SQL is quite expressive which means that people can build applications quickly, without having to concern themselves with how the queries are actually processed. This translates easily for a database system as queries written in SQL can be easily optimized by a query optimizer, since any query in SQL can be rewritten exclusively in terms of relational operators. Given that the database knows how these few operators work (also maintain statistics to aid them), it is fairly straightforward to optimize queries written in SQL. Being able to express operations using SQL is especially critical to road networks given that many branches of science and engineering deal with road networks and run expensive operations on them.

We argue that integrating road networks into a database makes good sense from a systems point of view. Suppose that we want to develop an application that finds all restaurants within 10 miles of a given postal address. If we were to design this algorithm, then we would store the restaurants as a relation in a database system. When a query point q is given, we would have to first query the database to obtain the set of restaurants that are likely to be within 10 miles of q, process this query in an external graph processing module, and then possibly store the result in the database. It would make for a much cleaner system design if we could perform all the operations in the context of a database system, which is really our goal here.

In Defense of Precomputation: As we pointed out earlier, integrating road networks into a relational database, requires a reassessment of how we deal with them. In this context, precomputation gives us an opening to transform road networks from a graph data structure to an alternate representation that can potentially make redundant the "graph" (or topological) aspect of most operations on road networks. The key idea is that by precomputing the shortest paths and distances between every pair of vertices in a road network, we can build a data structure residing in a database that can perform operations on road networks using SQL.

The question that we explore next is whether precomputation a bad strategy. We argue that it is not, pointing out that although the precomputation task is a big computational problem, it is not impossibly large. Of course, there is no denying that precomputation can be massively expensive for large road networks, but it can be achieved with a sufficient investment of time and hardware resources. We first argue that such a representation is feasible to compute. In particular, given a graph $G(V, E)$, where $n = |V|$, and $m = |E|$, Dijkstra's algorithm using a Fibonacci heap [7] takes $O(n^2 \log n + nm)$ time to compute the shortest path between all pairs of vertices in a spatial network. When $m = O(n)$, as in road networks, the time complexity of Dijkstra's algorithm would be $O(n^2 \log n)$. Empirical studies [25] have indicated that Dijkstra's algorithm is no where close to being the fastest algorithm for computing the all-pairs shortest paths on road networks. Moreover, recent developments in the shortest paths algorithm literature have shown better theoretical bounds on the computational time. In particular, Henzinger *et al.* [10] present a linear time shortest path algorithm for planar graphs, while Thorup [23] provides a linear time shortest path algorithm for general graphs with integer edge weights. Moreover, a host of other techniques such as parallel processing and the use of sophisticated hardware such as Graphics Processing Units (GPU) [9] could further speed up the precomputation of all shortest paths of a graph.

The real problem with precomputation is that the resulting precomputed representation is an impossibly large. Consider a spatial network $G(V, E)$ containing n vertices. Storing the n^2 shortest paths can require $O(n^3)$ space, when we assume that each shortest path can potentially have $O(n)$ vertices. Given that n is around 24 million vertices for the road network of USA, the anticipated size of the shortest path representation is around 13 billion trillion pieces of information, which is indeed impossibly large to store. Actually, one immediate reduction in the storage requirement can be achieved by just storing an intermediate vertex w ("next hop") on the shortest path from source s to destination t. We can then obtain the entire shortest path between s and t, by repeatedly finding the next vertex in the shortest path between s and w, and w and t, and so on. This reduces

the total amount of space necessary to $O(n^2)$, which comes at the cost of making shortest path retrieval into an iterative process taking $O(k)$ time, where k is the length of the shortest path. Even such a representation is quite large requiring 576 trillion pieces of storage for the road network of USA. In this article, we show that we can substantially reduce this precomputed input into a much compact representation that is just linear in n [19, 21].

It is important to note that precomputation is actually a good idea arguing from the point of view of *decoupling*, which provides a cost justification to any precomputation strategy. In particular, suppose that we precompute and store the shortest path and network distance between every pair of vertices in the road network of Manhattan, NY. Such a representation can potentially be used by several datasets (possibly, millions of user generated datasets) pertaining to postal addresses in Manhattan for query processing. Moreover, road networks are usually static structures, while datasets of objects may be updated frequently. For example, when dealing with a set of mobile hosts on a road network, the current positions of the objects are frequently updated, while the road network would largely remain static. So, precomputation is largely a one-time affair. In effect, precomputation enables us to *decouple* the *data* from the *underlying domain* which allows for the datasets and the network representation to be largely independent of each other.

2 Road Network Oracle

Our precomputed input representation of a road network containing n vertices is of size $O(n^3)$. Our goal here is to convert it to a database friendly representation that is much smaller in size. The Road Network Oracle (or simply *Oracle*) O of a road network G is a data structure that completely encapsulates all the n^2 shortest paths and network distances between every pair of vertices in G. The obvious storage choice for O in a relational database system is a *database relation*. In the following, we provide a basic blueprint for an oracle without making too many assumptions. Later in Section 3, we show how we can construct oracles for road networks.

The oracle O allows for two operations at the minimum, but of course support for additional functionalities is clearly preferable. Given a source vertex u and a destination vertex v, O provides an intermediate vertex in the shortest path between u and v. Next, given u and v, O also provides the network distance between u and v or more likely an approximation of it. Now, the key restriction that we place on O is that its size should be small so that both these operations can be be performed in a reasonable amount of time using database indices in O.

Without loss of generality, the scheme of oracle O can be deduced by observing that the input representation contains $O(n^2)$ information, which consists of an intermediate vertex in the shortest path between every pair of vertices, as well as the network distances between them. Of course, storing all of this information is not an option here, which means that we need to find a way to obtain a compact equivalent representation by removing *redundancies* from the input. One way to do this by aggregating a set of source vertices A and a set of destination vertices in B, such that the shortest paths from a source vertex in A to a destination vertex in B share a common vertex Ψ. Now, instead of recording the same intermediate vertex Ψ for every pair of source and destination vertices in A to B, respectively, we can simply represent all the individual vertices in A by their group identity (i.e., A), and all the individual vertices in B by their group identity (i.e., B). Similarly, network distances between source vertices in A to destination vertices in B, that are more or less of the same value can also be succinctly captured by just storing just a single network distance for the shortest paths from vertices in A to B.

The schema of a relation that captures the shortest paths is given by: $O(AB, \Psi)$, where AB represents the group identity of the vertices in the road network belonging to A and B, such that Ψ is the common intermediate vertex on the shortest paths between them. Another relation records the network distances separately using another relation with the schema: $O(AB, d_{apx})$, where again AB is the group identity of the source and destination vertices whose network distances are approximated with d_{apx}. Of course, for the sake of simplicity, if we assume (not without merit) that the vertices that make up AB in the relation storing the shortest paths are also the same ones that make up AB in the relation storing the network distances, then we can combine these two relations into a single relation with the schema: $O(AB, \Psi, d_{apx})$. The oracle O should capture all the $O(n^2)$

shortest paths and network distances. Given any pair of vertices p and q in the road networks, there should be exactly one tuple in O such that AB contains p and q. However, the groups AB should be maximal so that the number of tuples in the schema should be at a minimum. Finally, the group AB to which the pair (p, q) belongs should be unique so that it can be found quickly with the aid of the B-tree on the attribute AB in O.

Operations on Road Networks using SQL Given an implementation of $O(AB, \Psi, d_{apx})$, we now show how we can efficiently perform query processing on it using SQL. In particular, we demonstrate how many operations on road networks can be expressed using SQL (and relational operators) in the context of a database system. Our example assumes the following setup. Let R be a relation of restaurants with schema *(id, type, price)*, where *id* uniquely identifies restaurants on the road network, *type* is the kind of the cuisine served by the restaurant, and *price* is the average cost. We also define another relation Q of movie theaters given by the same schema *(id, movie_id)* where *id* uniquely identifies the movie theater on the road network, and *movie_id* is a movie playing in the theater. Given a source p and destination q, let $Z(p, q)$ map them to a group key AB such that A contains p, and B contains q so that one can find a tuple in O with the aid of B-tree on $O.AB$ such that the value of AB equals $Z(p, q)$. We present the following queries on a spatial network.

Approximate Network distance: Given a source p and destination q, obtain an approximate network distance between them.

SELECT $O.d_{apx}$ FROM O WHERE $O.AB = Z(p, q)$

Region Search: Given a query location q, obtain all restaurants in R that are within 10 miles of q which serve *Italian* cuisine. Of course, this query would make more sense if the oracle can give ϵ-guarantees on the quality of the network distance answers.

SELECT R.id, $O.d_{apx}$ FROM R, O WHERE $O.AB = Z(q, R.\text{id})$ and R.type = "Italian" and $O.d_{apx} \leq 10$ miles

k-Nearest Neighbor Search: Given a query location q, determine the k closest restaurants in R to q which serve *Italian* cuisine.

SELECT R.id, $O.d_{apx}$ FROM R, O WHERE $O.AB = Z(q, R.\text{id})$ and R.type = "Italian" ORDER BY $O.d_{apx}$ LIMIT k

Distance Join: Find the k closest pairs of restaurants in R and movie theaters in Q, such that Q is playing a movie of movie_id m.

SELECT R.id, Q.id, $O.d_{apx}$ FROM R, Q, O WHERE $O.AB = Z(R.\text{id}, Q.\text{id})$ AND Q.movie_id = m ORDER BY $O.d_{apx}$ LIMIT k

For a discussion on how the query optimizer can optimize queries on O, the reader is referred to [20].

3 Oracle Implementations

Now that we have laid out the design of an oracle, we can try to implement a few oracles. First of all, the input $O(n^2)$ can be stored as an oracle $O(uv, \Psi, d_G(u, v))$, where u is a source vertex, v is a destination vertex such that Ψ is an intermediate vertex in the shortest path from u to v and $d_G(u, v)$ is the network distance between u and v. The main drawback of this oracle is that it contains $O(n^2)$ tuples, which renders it infeasible owing to its large size.

We now present the design of two oracles both of which are based on the observation that road networks also contain additional information in the form of the spatial positions of road intersections (vertices) and road segments (edges). Shortest paths are related to the spatial information by the observation that the shortest paths between spatially proximate source vertices and spatially proximate destination vertices will often pass through

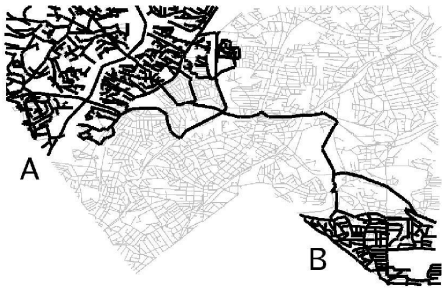

Figure 2: Example illustrating the path coherence in a road network of Silver Spring, MD. The 30,000 shortest paths (given in a darker shade) between every pair of vertices in A and B pass through a single vertex.

a common vertex, which is termed *path coherence*. Moreover, when the proximate source vertices are far from the proximate destination vertices we have the situation that the shortest paths between them have many common segments.

Figure 1(b) shows a colored map obtained by applying a coloring algorithm to the vertices based on the shortest path from a source vertex u in the road network G of Silver Spring, MD. Now, each vertex in G (other than u) is assigned a color based on which outgoing edge of u forms the first link in the shortest path from u. For example, u has six outgoing edges and hence, every vertex in G is assigned one of the 6 possible colors based on which of the 6 outgoing edges of u form the first link in the shortest path from u. We see that the resulting colored map has contiguous colored regions, which indicates that two destination vertices v, w in G that are close to one another, but spatially far from u share common segments in the shortest path from u.

An oracle, termed the *SILC Oracle*, which captures the above scenario is obtained as follows. Given a source vertex q, we perform such a coloring operation on all the vertices in the road network so that all the vertices except q have a unique color. Next, we can construct a quadtree representation as shown in Figure 1(c) on the colored vertices that keeps subdividing a block until all the vertices are of the same color, which means that they share the same first edge from q. We record the identity of these blocks using a pointerless quadtree representation which in essence is a pair of numbers where one number corresponds to the path from the root of the tree to the block and the other number corresponds to the depth of the block. These two numbers are concatenated to form an integer and the result is known as the block's *Morton block representation*. Let B represent one of these Morton blocks. Next, the shortest-path quadtree of a vertex is stored in a relation $O(q, B, \Psi, d_{apx})$, where q corresponds to a source vertex, B is the Morton block representation of one of the blocks in the quadtree, Ψ is the first link in the shortest path from q to the vertices contained in B, and d_{apx} is a network distance that approximates the network distances from q to all the vertices in B. We compute the shortest-path quadtree for each vertex in the road network and store the tuples in O. The shortest-path quadtree reduces the storage requirements from $O(n^2)$ to $O(n^{1.5})$ [15]. O is indexed using a two-dimensional B-tree on q, B. Given a source vertex u and a destination vertex v, O can retrieve the intermediate vertex and the approximate network distance in $O(\log n)$ time. The only drawback of this oracle is that d_{apx} is not of a bounded approximation, which means that for some cases the errors can be large.

The next oracle, termed the *path-distance oracle* [19,21], is obtained by utilizing the path coherence between sets of source and destination vertices such that they share common vertices in the shortest paths between them. Figure 2 shows a configuration of source vertices A and destination vertices B such that every shortest path from a vertex $u \in A$ to a vertex $t \in B$ passes through a particular set of vertices, resulting in storing partial path information of 30,000 shortest paths using $O(1)$ storage. The triple (A, B, u) is said to form a *Path-Coherent Pair (PCP)* if and only if all the shortest paths from source vertices in A to destination vertices in B have at least one vertex or one edge in common u as shown in Figure 2. In particular, we can show [19] that a given road network of size n, can be broken into $O(n)$ such groups of $O(1)$ size that capture all of the $O(n^2)$ shortest paths of the network. This partitioning of the vertices into appropriate subsets of source and destination vertices

is achieved by appealing to the Well-Separated Pair (WSP) decomposition [2], and conditions under which it is satisfied for a spatial network are specified in [21]. Moreover, as the shortest paths between A and B share large segments, the network distances between source vertices in A and destination vertices in B can be approximated by a single network distance value d_ϵ that provides ϵ-guarantees on the quality of the answers [19–21]. The end result is that given a road network G containing n vertices, we can break it up into $O(s^2n)$ sets (s is a small value > 0) of the form (AB, Ψ, d_ϵ) such that AB encapsulates the set of source and destination vertices, Ψ is an intermediate vertex common to all the shortest paths from A to B, and d_ϵ approximates all the network distances between A and B. This oracle satisfies all the conditions that we laid out earlier in Section 2 as it is linear in size, with no redundancy and can retrieve both the intermediate vertex and ϵ-approximate network distance quickly.

4 Concluding Remarks

In this paper we presented an alternate way of performing operations on road networks using road network oracles, which reside in a database system and enable operations on road networks using SQL. Our approach of converting road networks into oracles provides an opportunity to move away from traditional methods of working with road networks towards a way that is scalable, efficient, database friendly, and being able to support Internet-scale real time operations. We presented a few possible implementations of the oracle that use the spatial information in a road network to provide group memberships to vertices in the road network. The path-distance oracle [19–21] can be shown to be linear in n as well as being able to provide an intermediate vertex in the shortest path as well as an ϵ-approximate network distance between any pair of vertices drawn from the road network [21]. An interesting challenge is exploring whether there are other schemes of providing vertices in a road network with group memberships (á la spatial information) so that given source and destination vertices, one can find a tuple containing the pertinent shortest path and distance information using a B-tree. Another open problem is whether dynamic oracles can be designed so that they can deal with updates as is the case when road segments or vertices become closed or road segments become one way, as well as other dynamic traffic conditions such as congestion which may make some routes more acceptable. Finally, there is the issue of how to optimize operations involving the oracles so that operations can be optimized effectively in the context of a relational database system. In particular, how can a query optimizer be taught more strategies to perform road network queries more effectively.

Acknowledgments: This work was supported in part by the National Science Foundation under Grants IIS-09-48548, IIS-08-12377, CCF-08-30618, and IIS-07-13501, as well as NVIDIA Corporation, Microsoft Research, Google, the E.T.S. Walton Visitor Award of the Science Foundation of Ireland, and the National Center for Geocomputation at the National University of Ireland at Maynooth.

References

[1] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In transit to constant time shortest-path queries in road networks. In *ALENEX*, pp. 34–43, New Orleans, LA, Jan 2007.

[2] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *SODA*, pp. 291–300, Austin, TX, Jan. 1993.

[3] Z. Chen, H.-T. Shen, X. Zhou, and J. X. Yu. Monitoring path nearest neighbor in road networks. In *SIGMOD*, pp. 591–602, Providence, RI, June 2009.

[4] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to CNN queries in a road network. In *VLDB*, pp. 865–876, Trondheim, Norway, Sep. 2005.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] G. G. Filho and H. Samet. A hybrid shortest path algorithm for intra-regional queries in hierarchical shortest path finding. CS Tech. Report TR–4417, University of Maryland, College Park, MD, Nov. 2002.

[7] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *JACM*, 34(3):596–615, July 1987.

[8] A. V. Goldberg and R. F. Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX*, Vancouver, Canada, January 2005.

[9] P. Harish and P. J. Narayanan. Accelerating large graph algorithms on the GPU using CUDA. In *HiPC*, vol. 4873 of LNCS, pp. 197–208, Goa, India, Dec. 2007.

[10] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *JCSS*, 55(1):3–23, Aug. 1997.

[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. *TKDE*, 10(3):409–432, May 1998.

[12] M. R. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pp. 840–851, Toronto, Canada, Sep. 2004.

[13] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, pp. 802–813, Berlin, Germany, Sep. 2003.

[14] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, 2006.

[15] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, pp. 43–54, Vancouver, Canada, June 2008.

[16] J. Sankaranarayanan, H. Alborzi, and H. Samet. Efficient query processing on spatial networks. In *ACM GIS*, pp. 200–209, Bremen, Germany, Nov. 2005.

[17] J. Sankaranarayanan, H. Alborzi, and H. Samet. Distance join queries on spatial networks. In *ACM GIS*, pp. 211–218, Arlington, VA, Nov. 2006.

[18] J. Sankaranarayanan, H. Alborzi, and H. Samet. Enabling query processing on spatial networks. In *ICDE*, pp. 163, Atlanta, GA, Apr. 2006.

[19] J. Sankaranarayanan and H. Samet. Distance oracles for spatial networks. In *ICDE*, pp. 652–663, Shanghai, China, Apr. 2009.

[20] J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. *TKDE*, 2010. Best Papers of ICDE 2009 Special Issue. To appear.

[21] J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. In *VLDB*, volume 2, pp. 1210–1221, Lyon, France, Aug. 2009.

[22] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. *GeoInformatica*, 7(3):255–273, Sep. 2003.

[23] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *JACM*, 46(3):362–394, May 1999.

[24] D. Wagner and T. Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In *ESA 2003*, vol. 2832 of LNCS, pp. 776–787, Budapest, Hungary, Sep. 2003.

[25] F. Zhan and C. E. Noon. Shortest path algorithms: an evaluation using real road networks. *Transportation Science*, 32(1):65–73, Feb. 1998.