Query Processing Using Distance Oracles for Spatial Networks

Jagan Sankaranarayanan, *Member*, *IEEE*, and Hanan Samet, *Fellow*, *IEEE*

Abstract—The popularity of location-based services and the need to do real-time processing on them has led to an interest in performing queries on transportation networks, such as finding shortest paths and finding nearest neighbors. The challenge here is that the efficient execution of spatial operations usually involves the computation of distance along a spatial network instead of "as the crow flies," which is not simple. Techniques are described that enable the determination of the network distance between any pair of points (i.e., vertices) with as little as $O(n)$ space rather than having to store the n^2 distances between all pairs. This is done by being willing to expend a bit more time to achieve this goal such as $O(\log n)$ instead of $O(1)$, as well as by accepting an error ε in the accuracy of the distance that is provided. The strategy that is adopted reduces the space requirements and is based on the ability to identify groups of source and destination vertices for which the distance is approximately the same within some ε. The reductions are achieved by introducing a construct termed a *distance oracle* that yields an estimate of the network distance (termed the *ε-approximate distance*) between any two vertices in the spatial network. The distance oracle is obtained by showing how to adapt the well-separated pair technique from computational geometry to spatial networks. Initially, an ε-approximate distance oracle of size $O(\frac{n}{\varepsilon^d})$ is used that is capable of retrieving the approximate network distance in $O(\log n)$ time using a B-tree. The retrieval time can be theoretically reduced further to $O(1)$ time by proposing another ε-approximate distance oracle of size $O(\frac{n \log n}{\varepsilon^d})$ that uses a hash table. Experimental results indicate that the proposed technique is scalable and can be applied to sufficiently large road networks. For example, a 10-percent-approximate oracle ($\varepsilon = 0.1$) on a large network yielded an average error of 0.9 percent with 90 percent of the answers having an error of 2 percent or less and an average retrieval time of $68\ \mu$ seconds. The fact that the network distance can be approximated by one value is used to show how a number of spatial queries can be formulated using appropriate SQL constructs and a few built-in primitives. The result is that these operations can be executed on almost any modern database with no modifications, while taking advantage of the existing query optimizers and query processing strategies.

Index Terms—Road networks, distance oracle, query processing.

1 Introduction

The popularity of web-based mapping applications such as Mapquest, Yahoo Maps, and the subsequent enhancements available in Google Maps and Microsoft Live Search, as well as the increasing pervasiveness of GPS-enabled devices such as PDAs, have led to an expectation of real-time execution for queries on transportation networks, such as computing shortest paths and finding nearest objects from a set R (e.g., restaurants, department stores, and gas stations). For example, suppose that we found the shortest distance from gas station A to the nearest restaurant B, which serves Italian food, and we wish to determine how much farther it is to go to another restaurant C, which serves Chinese food.

The challenge is that these operations involve the computation of distance along a spatial network instead of "as the crow flies," which is not simple. Our overall goal is to be able to determine the network distance between any pair of points (i.e., vertices) without having to store the n^2 distances between all pairs. We are willing to expend a bit more time to achieve this such as $O(\log n)$ instead of $O(1)$ as well as accept an error ε in the accuracy of the distance that is provided. The strategy that we follow reduces the space requirements to as low as $O(\frac{n}{\varepsilon^d})$, and is based on the ability to identify groups of source and destination vertices for which the distance is approximately the same within some ε. In particular, the $O(\frac{n}{\varepsilon^d})$ space comes at a cost of $O(\log n)$ execution time, while greater reductions in the execution time to $O(1)$ can be achieved at a cost of $O(\frac{n \log n}{\varepsilon^d})$ space, which is shown in [1]. Of course, we could always obtain an answer in $O(1)$ time and no extra space by simply approximating the network distance by using the euclidean distance (i.e., as the crow flies) which can be calculated on the fly; but the error is generally unacceptable. In addition, these operations must be capable of being executed in an interoperable database environment which, at the minimum, means that they be accessed in a standard manner. Currently, SQL is the most commonly adopted interface to a database, and thus any solution that we propose must support the formulation of the operations using SQL.

Moreover, the requirement that these distances be computed in real-time precludes the use of conventional graph-based algorithms (e.g., the INE and IER methods [2] and improvements on them [3], and hierarchical graph methods [4], [5]) which usually incorporate Dijkstra's shortest path algorithm in at least some parts of the solution [6]. It is well known that the problem with Dijkstra's

• *The authors are with the Center for Automation Research, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland, Bldg 116, College Park, MD 20742. E-mail: {jagan, hjs}@cs.umd.edu.*

Manuscript received 15 May 2009; revised 30 Sept. 2009; accepted 12 Nov. 2009; published online 20 Apr. 2010.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-2009-05-0435.
Digital Object Identifier no. 10.1109/TKDE.2010.75.

algorithm is that although it reports the shortest path from a starting vertex w to every other vertex v in increasing order of distance from v, it must visit every vertex that is closer to w via the shortest path from w than the vertices associated with the desired objects in R. Thus, it ends up visiting a very large number of the vertices, even though the shortest paths to the objects in R do not pass through them. Methods such as the *transit node routing* of Bast et al. [7] and the landmark approach of Golberg and Harrelson [8] can significantly speedup shortest path computations on large road networks, up to several orders of magnitude compared to Dijkstra's algorithm. However, there are significant differences between distance oracles and these shortest path finding methods. Distance oracles are unique in the sense that they completely abstract away the graph properties of spatial networks. Thus, finding the network distance between two vertices is a simple lookup (select operator) on a relation in a database system. Methods such as [7], [8] still involve searches on graphs during runtime, which are hard to perform in the context of a database system as shortest path finding on a graph cannot be expressed easily using relational operators, which makes it hard to implement these algorithms inside a database system. Moreover, the real utility of the distance oracle lies in its ability to perform a variety of queries on a spatial network and its use in complex query scenarios; approximate network distance query being just one of them.

A drastic alternative to the use of Dijkstra's algorithm is to precompute and store the shortest paths between all possible vertices in the spatial network. The drawback of this approach is that, for n vertices, the amount of storage could be as high as $O(n^3)$. The necessary storage can be reduced to $O(n^2)$ by taking advantage of the fact that the shortest paths from vertex u to all remaining vertices can be decomposed into subsets based on the first edges on the shortest paths to them from u [9], [10], [11]. The cost is a slower process of retrieving the shortest path which makes use of a sequence of point location operations (e.g., [6]). We have shown that the storage necessary for these subsets can be reduced substantially further to $O(n^{1.5})$ [9], [10], [12] by noting the spatial coherence of the subsets and representing them using a shortest path quadtree, which is a variant of the region quadtree, where the blocks are decomposed until all vertices in the block are in the same subset. Note that the use of the quadtree in that context is primarily to take advantage of its dimension-reducing property [13] to decrease the storage requirements.

The above algorithms exploit the spatial coherence of the destination vertices of the spatial network to reduce the storage requirements of the collection of precomputed shortest paths from a specified source vertex. In this paper, we continue our work [9] by showing how to also take advantage of the spatial coherence of the source vertices to further reduce the space requirements. In particular, we observe that given a set of source vertices A and a set of destination vertices B such that A and B are *sufficiently* far away from each other, while the vertices comprising them are close to one another, then the shortest paths between them may share common vertices, which in turn implies that the network distance between any source vertex in A to any destination vertex in B will more or less be the same. Fig. 1 is an example of such a configuration where all the 30,000 shortest paths between vertices in A and in B have many vertices in common, while the network distances between them can be approximated by a single value.

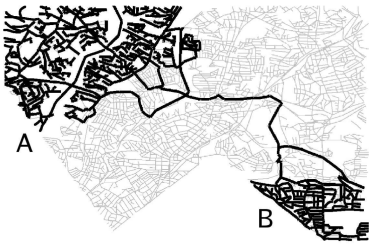

Fig. 1. The 30,000 shortest paths between all pairs of vertices in sets A and B in the spatial network of Silver Spring, MD, are marked in a darker shade. These network distances can be approximated by a single value as their shortest paths have many vertices in common.

The techniques that we develop in this paper are based on our inference that given our assumptions on the proximity of the vertices that comprise A and those that comprise B, and the lack of proximity between A and B, that the network distance to the vertices in A from the vertices in B will more or less be similar and can be approximated by a single value (termed the *path coherence property* [9], [10], [14]). The novelty of our approach is that in the case of the computation of the distance between two vertices, we show how to correlate the extent of this reduction of the space requirements with the approximation error in the value of the distance that is obtained. This is achieved via the introduction of a more general construct, termed an *approximate distance oracle* for spatial networks, that is capable of responding to network distance queries between any two vertices of the spatial network with a specified approximation ε—that is, given a start vertex u and a destination vertex w in spatial network G, the network distance $S_\varepsilon(u, w)$ produced by the oracle S_ε is no more or less than an ε fraction of the actual network distance $d_G(u, w)$ between u and w in G.

The use of a single value to approximate these distances is based on our observation that the distance distortion (i.e., the ratio of the network distance to the spatial distance between two vertices in a spatial network) decreases as the separation between the vertices increases and our demonstration that it has a reasonable bound. Assuming d-dimensional data (usually $d = 2$), the latter, coupled with the path coherence property, enable us to devise an $O(\frac{n}{\varepsilon^d})$ size oracle represented using a B-tree that is capable of retrieving the ε-approximate network distance in $O(\log n)$ time with a deterministic guarantee on the error. An alternative oracle that can reduce the retrieval time further to $O(1)$ with an increase in space to $O(\frac{n \log n}{\varepsilon^d})$ using a hash table is possible [1]. Regardless of the size of the oracle that is used, we take advantage of the fact that the network distance can be approximated by one value to show how a number of spatial queries can be formulated using appropriate SQL constructs and a few built-in primitives. The result is that these operations can be executed on almost any modern database with no modifications, while taking advantage of the existing query optimizers and query processing strategies.

We achieve our results by showing how to efficiently represent the network distance between the spatially coherent collections of source vertices and the spatially coherent collections of destination vertices. This is done by showing how to adapt the notion of a well-separated decomposition of a point set, originally proposed by Callahan and Kosaraju [15] for a point set and used by others (e.g., [16], [17]), to a spatial network.

One way of evaluating the significance of our work lies in determining the extent to which we can improve on the storage costs that we obtained in our earlier work [9] where we represent the spatially coherent destination vertices by a shortest path quadtree, which had a factor of $O(n^{0.5})$. Clearly, if we choose ε to be very small, then our space requirements, which, although appearing to be linear (i.e., $O(n/\varepsilon^d)$), will become sufficiently large to counteract any advantage drawn from the use of this linear-size oracle. However, the execution time of the distance computation process is also quite low when using the linear-size oracle as the shortest path quadtree method [9] does not explicitly store the distances between the vertices (instead, it stores distance intervals), and thus whenever it needs to compute the distance between a pair of vertices, it must compute the shortest path between them (with a possible halt once the required approximation error threshold is attained). This usually involves a large number of refinement operations, which can be slow.

At this point, we mention a few related methods, but we first present a few definitions. A spatial network can be abstracted to form an equivalent graph representation $G = (V, E)$, where V is the set of vertices, E is the set of edges, $n = |V|$, and $m = |E|$. Given $e \in E$, $w(e) \geq 0$ denotes the distance along that edge. In addition, for every $v \in V$, $p(v)$ denotes the spatial position of v with respect to S, a *spatial domain*, also referred to as an *embedding space* (i.e., a reference coordinate system). We define the *network distance* $d_G(u, v)$ to be the distance along the shortest path between u and v in the spatial network. Similarly, we define the spatial distance $d_S(u, v)$ to be a function of the position of the vertices u and v on the embedding plane. For example, in the case of a road network the network distance between two vertices is the shortest distance in miles, or the time taken to travel the road network, while the spatial distance (e.g., "crow flying" distance) is a function of latitude/longitude positions of the vertices.

Furthermore, we assume that for some spatial networks (e.g., the road networks), the network distance between any two vertices is bounded from above and below by two constants γ_L, γ_H (presumably large), such that

$$\gamma_L \leq \frac{d_G(u, v)}{d_S(u, v)} \leq \gamma_H;\ \gamma_L, \gamma_H > 0.$$

The constants γ_L, γ_H are termed the minimum and maximum distortions of G. Narasimhan and Smid [18] provide a simple technique for estimating the value of γ_H for euclidean networks which is easy to adapt to spatial networks. Our experiments show γ_H to be large for road networks.

The technique that we propose is similar to the RNE technique of Shahabi et al. [19] and an improvement to it by Kriegel et al. [20] who apply a Lipschitz embedding [21] to spatial networks. The RNE technique *embeds* the vertices of the spatial network in a high-dimensional vector space, so that vertices of the spatial network are now points in a high-dimensional vector space. A simpler distance measure (e.g., L_∞ metric) between these high-dimensional vector space points approximates the network distance between the corresponding vertices in the spatial network. The RNE technique uses $O(n\sqrt{n})$ storage, has a distortion of $O(\log n)$ and an approximate network distance query takes $O(\sqrt{n})$ time. On the other hand, our linear-size oracle can also be viewed as an embedding technique with the difference that the vertices are retained in their original embedding space (i.e., two-dimensional for road networks), while having superior space and execution times and a distortion that lies between $(1 - \varepsilon)$ and $(1 + \varepsilon)$. This bounded distortion, instead of being a function of n, is what leads to the linear size of our oracle in contrast to RNE's $O(n\sqrt{n})$ storage requirements. Another difference between our proposed method and that of Shahabi et al. [19] and Kriegel et al. [20] is that our distance oracle *decouples* the spatial network from the objects that lie on it. Thus, once an approximate distance oracle of a spatial network has been computed, it can be reused for any data set lying on the spatial network, which is not the case for the other methods.

The concept of an approximate distance oracle has been proposed for a variety of graph networks. Thorup and Zwick [22] show that it is possible to construct an approximate oracle of size $O(kn^{1+\frac{1}{k}})$ for general graphs that can answer approximate distance queries in $O(1)$ time. The distortion of the approximate oracle of Thorup and Zwick lies between 1 and $(2k - 1)$, where $k \geq 1$ is an integer. Gudmundsson et al. [17] construct an approximate oracle of size $O(n \log n)$ for *geometric t-spanner* graphs, such that the shortest path queries can be performed in $O(1)$ time with a distortion of $(1 + \varepsilon)$. Gao and Zhang [16] propose an approximate oracle of size $O(n \log n)$ for unit-disk graphs that can retrieve approximate network distance in $O(1)$ time, with a distortion of $(1 + \varepsilon)$. Our work goes beyond the work of Gudmundsson et al. [17] on geometric t-spanners and Gao and Zhang [16] on unit-disk graphs by dealing with spatial networks, while taking advantage of the spatial positions of the vertices to provide efficient search structures, such as B-trees, and hash tables, to the oracle.

The rest of this paper is organized as follows: Section 2 reviews the well-separated pairs technique. Sections 3 and 4 describe oracles of unit, $O(\frac{n}{\varepsilon^d})$, and $O(\frac{n \log n}{\varepsilon^d})$ sizes, respectively. Section 5 discusses strategies to integrate the distance oracle into a database system, while Section 6 shows how to optimize queries involving distance oracles. Finally, Section 7 contains the results of experiments, while concluding remarks are drawn in Section 8.

2 Well-Separated Pairs

Given a set of points A, the *diameter* of A is the maximum possible distance between any two points belonging to A. Similarly, given two sets of points A and B, the *separation distance* between A and B is the distance between a point in A and a point in B, both of which are chosen at random. Two sets of points A and B are said to be *well separated* if the separation distance between A and B is at least $s \cdot r$, where $s > 0$ is a *separation factor* and r is the larger diameter of the

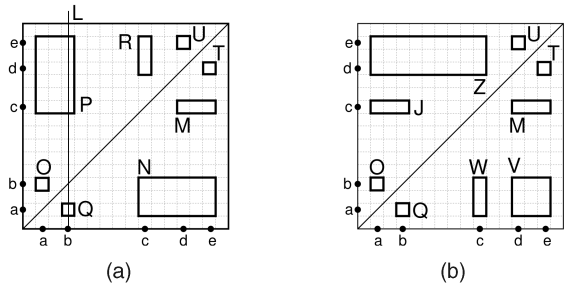

Fig. 2. Example of a well-separated pair decomposition (WSPD) of a one-dimensional point set containing five points. The separation factors for the decompositions are (a) $s = 1$ and (b) $s = 0.25$.

two sets. The pair (A, B) is termed a *well-separated pair* (WSP). A *WSPD* of a point set R, decomposes R into pairs of subsets (A, B), such that $\forall p, q \in S, p \neq q$, there exists exactly one WSP (A, B), such that $p \in A, q \in B$. The simplest WSPD of a point set R of n points contains $n \cdot (n-1)$ pairs of singleton element subsets (p, q) $\forall p, q \in S, p \neq q$. The key motivation for using WSPD is that for data of dimension d, and a separation factor s, we can always construct a WSPD containing $O(ns^d)$ pairs in $O(n \log n + ns^d)$ time [23], which can be further reduced to $O(n)$ time [24] under some conditions. Thus, the number of pairs is reduced to $O(n)$ as s is usually a fairly small constant independent of n.

As an example, consider the set of five one-dimensional points a, b, c, d, and e at positions 1.5. 3.5, 9.5, 12.5, and 14.5, respectively. There are a number of possible WSPDs for this data set. Letting $s = 1$, one decomposition consists of $M = (\{d, e\}, \{c\})$, $N = (\{c, d, e\}, \{a, b\})$, $O = (\{a\}, \{b\})P = (\{a, b\}, \{c, d, e\})$, $Q = (\{b\}, \{a\})$, $R = (\{c\}, \{d, e\})$, $T = (\{e\}, \{d\})$, and $U = (\{d\}, \{e\})$. This decomposition can be visualized by treating the individual pairs that make up the WSPD as rectangles in a two-dimensional space, where the axes correspond to the elements that make up the two sets involved in the WSP. For example, Fig. 2a illustrates the WSPD described above for $s = 1$, while Fig. 2b illustrates another decomposition for the same points with $s = 0.25$. Notice that from the figure, we can see that any vertical (or horizontal) line L through one of the points, say p (e.g., b in Fig. 2a), will cut the disjoint rectangles through which it passes so that the projection of their constituent points onto the y (or x) axis covers all of the points in R with the exception of p which means that, given any point $p', p' \neq p$, in the data set, there is exactly one WSP A, B in the WSPD such that $p \in A$ and $p' \in B$. Also observe that just because the WSP (A, B) is a member of a WSPD does not necessarily mean that the symmetric pair (B, A) need be a member of the same WSPD. For example, consider the WSPD in Fig. 2b where the symmetric pairs of $Z = (\{a, b, c\}, \{d, e\})$, $M = (\{d, e\}, \{c\})$, and $V = (\{d, e\}, \{a, b\})$ are not present.

3 Oracles of Unit Size

We first assume that the ratio, termed distortion, between the network and spatial distances between any source and destination vertices in spatial network G is bounded both from above and below. Next, we show how these bounds can be computed. Note that given any finite spatial network G, the upper and lower bounds on the distortions are finite as well.

Assumption 1. *Given $s, t \in V$, $\gamma_L \leq \frac{d_G(s,t)}{d_S(s,t)} \leq \gamma_H$, where γ_L and $\gamma_H > 0$, albeit large.*

Given a spatial network $G(V, E)$, Lemma 3.1 provides a simple method to determine the minimum γ_L distortion of G.

Lemma 3.1. *The minimum γ_L distortion of a spatial network $G(V, E)$ containing m edges, satisfying Assumption 1, is given by: $\gamma_L = \min_{i=0}^{m} \{\frac{w(\pi_i, \pi_{i+1})}{d_S(\pi_i, \pi_{i+1})}\}$, where $<\pi_i, \pi_{i+1}>$ is the ith edge in G.*

Proof. Let π be a shortest path of length p in G. Let π_i be the ith vertex in π, such that π_1 is the source vertex and π_{p+1} is the destination vertex. Suppose that the distortion $\gamma_L = \gamma = \frac{d_G(\pi_1, \pi_{p+1})}{d_S(\pi_1, \pi_{p+1})}$ of π is the minimum possible distortion among the $O(n^2)$ shortest paths in G. Now, let γ_i^* be the distortion of the ith edge $\phi_i = (\pi_i, \pi_{i+i})$ in π comprising the shortest path π. We know that

$$d_G(\pi_1, \pi_{p+1}) = \gamma d_S(\pi_1, \pi_{p+1}) = \sum_{i=1}^{p} \gamma_i^* d_S(\pi_i, \pi_{i+1}), \quad (1)$$

$$d_S(\pi_1, \pi_{p+1}) = \sum_{i=1}^{p} \frac{\gamma_i^*}{\gamma} d_S(\pi_i, \pi_{i+1}). \quad (2)$$

Note that in (2), from our initial assumption, $\frac{\gamma_i^*}{\gamma} \geq 1$. From the triangle inequality, we have that

$$d_S(\pi_1, \pi_{p+1}) \leq \sum_{i=1}^{p} d_S(\pi_i, \pi_{i+1}). \quad (3)$$

Combining (2) and (3), we get

$$d_S(\pi_1, \pi_{p+1}) = \sum_{i=1}^{p} \frac{\gamma_i^*}{\gamma} d_S(\pi_i, \pi_{i+1}) \leq \sum_{i=1}^{p} d_S(\pi_i, \pi_{i+1}).$$

Hence, $\gamma = \gamma_1^* = \gamma_2^* = \gamma_3^* \cdots = \gamma_p^*$. This means that γ_L can be estimated by simply examining the minimum distortion value of the m edges in E. In other words, $\gamma_L = \min_{i=0}^{m} \{\frac{w(\pi_i, \pi_{i+1})}{d_S(\pi_i, \pi_{i+1})}\}$ such that $<\pi_i, \pi_{i+1}>$ is the ith edge in E. □

Narasimhan and Smid [18] provide an algorithm that is based on a WSPD of vertices for estimating an upper bound on the maximum distortion of a spatial network. We omit the description of the algorithm in this paper and provide a lemma below which captures their result.

Lemma 3.2. *Given a spatial network G, we can compute an upper bound γ_H^* of the maximum distortion γ_H of G in $O(n \log n)$ time such that $\gamma_H^* \leq (1 + \delta)\gamma_H$ and δ, $0 < \delta < 3$, is the desired approximation [18].*

We define a simple approximate distance oracle that uses the values of γ_L and γ_H to provide an approximate network distance between two vertices, though with a large approximation error.

Theorem 3.3. *Given a spatial network $G(V, E)$ with minimum γ_L and maximum γ_H distortions, $S_\varepsilon = \{\gamma_L, \gamma_H\}$ is an ε-approximate distance oracle of unit size such that the approximate network distance between a source vertex u and a destination vertex v is given by $S_\varepsilon(u, v) = \frac{\gamma_L + \gamma_H}{2} d_S(u, v)$, where $\varepsilon \leq \frac{\gamma_H - \gamma_L}{\gamma_H + \gamma_L}$, which can be computed in $O(1)$ time.*

The drawback of the above oracle is that the resulting error is dependent on the nature of the input spatial network G as the values of γ_L and γ_H may vary for different G. Moreover, γ_H can be quite large which means that the resulting error of the distance oracle is also very large, close to 100 percent. Hence, S_ε is unsuitable for any meaningful query processing on spatial networks.

Distortion spectrum. The behavior of the unit size oracle in Theorem 3.3 can be improved by noting that the minimum and the maximum distortion values of a spatial network depend on the spatial distance between a given source s and destination u. That is, usually for small spatial distances on G, the distortion values are large. However, distortion values quickly decrease as the spatial distance between the source and the destination increases. We can capture this relationship between the spatial distance and the distortion using a *distortion spectrum*, which provides the minimum and the maximum distortion values for different ranges of values of the spatial distances on G. The idea in capturing the distortion spectrum of a spatial network, instead of just computing the minimum γ_L and maximum γ_H distortion values of G, is that the resulting oracle while still taking $O(1)$ space and answering queries in $O(1)$ time, will provide better approximations, at least for large spatial distances on G.

Given a spatial network G, we first compute the spatial distance between the closest d_c and the farthest d_f pairs (diameter) of vertices in V. We then break up the spatial distance interval $[d_c, d_f]$ into l arbitrary subintervals. Now for each of the l distance intervals, the minimum and maximum distortion values are computed using an algorithm similar to that of Narasimhan and Smid [18], except that we prune away WSPs that do not lie inside the specified spatial distance interval. The distortion spectrum of G stores l spatial distance intervals and their corresponding minimum and maximum distortion values. Our experimental analysis confirms our earlier hypothesis that large distortions occur at small spatial distances. Moreover, the distortion values quickly reduce to smaller values as the sources and destinations get farther. An approximate distance oracle S_ε defined using the distortion spectrum of G would provide better approximation, at least when s and u are far apart in G.

4 $O(\frac{n}{\varepsilon^d})$ Distance Oracle

Given a source vertex u and a destination vertex w, an ε-approximate distance oracle S_ε provides an ε-approximate network distance $S_\varepsilon(u, w)$ such that:

$$(1 - \varepsilon) \cdot S_\varepsilon(u, w) \leq d_G(u, w) \leq (1 + \varepsilon) \cdot S_\varepsilon(u, w).$$

S_ε takes advantage of the path coherence in spatial networks which states that shortest paths from proximal sources to proximal destinations share common vertices. This can be seen in Fig. 1 and was discussed in Section 1.

We define a distance oracle S_ε of a spatial network as follows: $S_\varepsilon = \{(Z_A, Z_B, d_\varepsilon) | Z_A, Z_B \subset V, d_\varepsilon \in \mathbb{R}^+\}$. That is, we partition V into triples of the form $(Z_A, Z_B, d_\varepsilon)$ such that Z_A is a set of source vertices, Z_B is a set of destination vertices, and d_ε is a value that approximates the network distances of all the shortest paths from sources in Z_A to destinations in Z_B. The partitioning of the vertices into appropriate subsets of source and destination vertices is achieved by appealing to the well-separated pair decomposition [23]. This section specifies conditions under which it is satisfied for a spatial network.

The ε-approximate network distance between a source u and a destination w is obtained by searching S_ε, which is indexed by a B-tree, for a triple $(Z_A, Z_B, d_\varepsilon)$ such that Z_A contains u and Z_B contains w, in which case, d_ε is the ε-approximation of the network distance between u and w. In this section, we develop such a distance oracle S_ε that is *linear* in the number of vertices in G, and can produce an ε-approximate network distance in $O(\log n)$ time using a B-tree.

4.1 Preliminaries

Given a point set R in a d-dimensional space, there are many ways of constructing a WSPD on R. In essence, we do so by finding the well-separated pairs of blocks resulting from building a PR quadtree [6], [25] T on R. For simplicity, we assume that R is contained in a unit $[0, 1]^d$ d-dimensional hypercube. This hypercube forms the root block of the PR-quadtree T on R. The PR-quadtree is obtained by recursively decomposing the block into 2^d congruent children blocks. The process continues until each block contains at most one point. Unfortunately, if two points in R are close to one another, it may lead to a long path of trivial blocks of which only one block would form an internal node. This problem is avoided in Callahan and Kosaraju's construction [15] as they used a fair-split tree which is a data-dependent decomposition, very much in the spirit of the point quadtree [6]. Fischer and Har-Peled [26] avoid this problem by using a variant of a *path-compressed* quadtree which is obtained from the PR-quadtree by compressing such trivial paths into one compressed link. The advantage of the path-compressed quadtree over the PR-quadtree is that its use yields a tree with a total of $O(n)$ blocks.

In this paper, we restrict our discussion to a spatial network that is embedded in a two-dimensional space (e.g., road networks) where the objects are points specified by their location in terms of, say, latitude and longitude. Nevertheless, many of the definitions apply to d-dimensional data and hence our discussion is often in terms of d. The usual implementation of a quadtree makes use of an access structure in the form of a tree of out degree $C = 2^d$ (four in two dimensions), which implies the existence of a pointer-based explicit search hierarchy. Such a structure, which uses memory-based pointers, is not suitable for disk-based quadtree data structures. There are a number of alternative access structures to the pointer-based tree structure. The one that we use represents the quadtree as a collection of the leaf nodes (i.e., blocks) that comprise it. The basic idea is that each leaf is encoded by a pair of numbers. The first corresponds to the depth of the tree at which its corresponding node appears (also referred to as its level), while the second is a base 4 number corresponding to a sequence of digits whose values are directional codes that locate the leaf along a path from the root of the quadtree. Assuming a two-dimensional square universe of side length 2^m (i.e., containing $2^m \times 2^m$ unit-sized elements called *pixels*), it is analogous to taking the binary representation of the interleaved values of the x and y

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

Fig. 3. Example of the Morton code mapping for the two-dimensional grid [0:7,0:7].

coordinates of a designated pixel in the block (e.g., the one at the upper left corner when the origin is at the upper left corner of the $2^m \times 2^m$ universe) and interleaving them (i.e., alternating the bits for each coordinate value), with the result called a *Morton code*. We can simplify the representation by combining the depth (level) with the base 4 number corresponding to the path so that the depth is at the left to yield what is termed a *Morton block*, say t whose components are referenced by $\text{LEVEL}(t)$ and $\text{CODE}(t)$, respectively.

What we have done above is to construct a mapping from the multidimensional space containing the quadtree blocks to a one-dimensional space. We now show how to apply the same type of mapping to points. Without loss of generality, we assume that the points are drawn from R, the unit $[0,1]^2$ square. Scaling the universe to be a large square of side length 2^m, for each point p in R, we can obtain its Morton code $Z(p)$ by using the following mapping $Z(p) : \mathbb{R}^d \rightarrow \mathbb{N}$. Letting, p_i be the ith dimension of p, we have that $B(p_i) = p_i * 2^m$ is bit representation of p_i of length m. We define an *integer dilation* function E, such that $E(B(p_i))$ spreads the bits of $B(p_i)$ apart with $d-1$ zeros. For example: for $d = 2$, $E(111) = 10101$, where 111 and 10101 are binary numbers. The Morton code of p is obtained by shifting $E(B(p_i))$ by i bits to the left and adding the shifted results (i.e., $\sum_{i=0}^{d-1}(E(B(p_i)) \ll i)$). Fig. 3 shows the Morton code mapping for a set of points drawn from the two-dimensional grid [0:7,0:7], which yields an ordering of the underlying space. Notice how each point has been mapped to a unique point in the interval [0:63].

For all practical cases, we restrict the length of the Morton code to 58 bits (29 each for x and y-dimensions) and assign 6 bits to encode the *level* (i.e., depth) of the Morton code, thereby, yielding a resolution of roughly 7.46 cm at the Equator of the Earth. This means that the level and the code that define the Morton block of each leaf block or point are packed into a single 64 bit-integer, thereby, taking advantage of the bit-level parallelism in 64 bit machines and achieved only using bit operations. In addition, we define another transformation function $Z_4 : (\mathbb{R}^2, \mathbb{R}^2) \rightarrow \mathbb{N}$ that transforms a pair of two-dimensional points into a 128 bit Morton block. In this case, 30 bits are assigned to each of the four dimensions with 8 bits assigned to storing the level.

Once the Morton blocks of the quadtree is computed, they are sorted using a *Z or Morton* ordering [6], [27], [28] (described in the following paragraphs) and stored on disk. The Z-order or Morton Order is an example of a space-filling curve on R, which takes a set of multidimensional objects and maps them to a one-dimensional space. Furthermore, given two Morton blocks p and q, we can arrange p and q based on their relative positions in a Z-order space-filling curve using the $\leq_Z$ operation in $O(1)$ time. Note that p appears before q in the space-filling curve only if: $p \leq_Z q, \iff \text{CODE}(p) \leq \text{CODE}(q)$. Note that if p and q are at different levels, then we first chop off the least significant bits from the CODE of the longer of p and q before numerically comparing them. Now, two Morton blocks p and q are said to be *equal* (i.e., $p =_Z q$ is true) only if p contains q, or q contains p. This can be achieved using bit operations by first making the Morton blocks be at the same LEVEL and then numerically comparing the CODE fields of p and q for equality.

When applying the WSPD on the set of vertices V on a spatial network, our discussion does not need to resort to the path-compressed quadtree while still using regular decomposition because of certain assumptions that we make about the distribution of the vertices in the embedding space. In particular, letting Δ be the ratio of the diameter of the set of vertices V to the distance between the closest pair of vertices in V and letting T be a PR-quadtree on V, the height h of T is $O(\log \Delta)$. Consequently, given a vertex v in V, the Morton code of $p(v)$, the spatial position of v, would be $O(\log \Delta)$ bits long. Assuming, without loss of generality, that the closest pair of vertices are one unit apart and that the embedding space is two-dimensional, we can pack as many as $\frac{\Delta^2}{2}$ vertices into our domain, where each vertex is in a cell of the appropriate width. Therefore, Δ is at least $O(\sqrt{n})$ so that the domain is large enough to accommodate at least n points. In our analysis, we assume that not all of the cells have vertices associated with them. In particular, we allow for Δ to be as large as $O(n)$ (i.e., number of possible point positions is $O(n^2)$), which is not unreasonable as demonstrated by our experiments with real road networks. To cast this quantity in terms of n, we note that even if the data are heavily skewed so that Δ is $O(n)$, the length of the Morton code representation of v would still be $O(\log n)$. We claim that this assumption fits closely with the nature of real road networks.

The decomposition of R into WSPs is a *realization* on T, i.e., subsets A_i, B_i of R forming a WSP (A_i, B_i) in $R \otimes R$ are pairs of blocks in R. The algorithm decomposes R into WSPs using T and s (i.e., the separation factor) as inputs. The algorithm uses a list Q that is initialized to the pair (T, T) corresponding to the root of the quadtree on R. In each iteration of the algorithm, a pair (A, B) of blocks in T is retrieved from Q. If (A, B) is well separated, it is reported as a WSP. Otherwise, new pairs are obtained by replacing A and B with their 2^d children blocks, which are inserted into Q. The algorithm terminates when Q is empty.

Suppose that a pair (u, v) is reported as a WSP by the algorithm. This would indicate that $(P(u), P(v))$ is not well separated, where $P(b)$ denotes the parent block of a block b. Suppose further that the maximum possible diameter of $P(u)$ (or $P(v)$) is x. The total number of blocks that are not well separated from $P(u)$ is bounded by the number of blocks of diameter x that are contained within a hypersphere of diameter $(2s+1)x$ centered at $P(u)$, which contains a maximum of $O(s^d)$ blocks. Since T has $O(n)$ nodes, the algorithm creates a maximum of $O(s^d n)$ WSPs. This result and proof sketch is due to Callahan and Kosaraju [15] and we restate it below as Lemma 4.1, which is referenced in the subsequent discussion.

Lemma 4.1. *Given a point set R containing n d-dimensional points, a fixed separation factor $s \geq 2$, the WSPD of R, $S \otimes S$ has $O(s^d n)$ WSPs [15].*

4.2 Construction of the Oracle

We now describe an algorithm to construct the distance oracle S_ε of a spatial network $G(V, E)$. The algorithm takes a spatial network $G(V, E)$, a PR-quadtree T on the spatial position of the set of vertices V as inputs, the desired approximation ε, and produces a set L of triples $(Z_A, Z_B, d_\varepsilon)$, such that Z_A, Z_B are Morton blocks in T and moreover, Z_A and Z_B are nonoverlapping. A triple $(Z_A, Z_B, d_\varepsilon)$ in the output has the property that the network distances between source locations in Z_A to destination locations in Z_B can be approximated by d_ε. The output list L is initially empty. The value of d_ε is obtained by choosing a pair of representative points p_A in Z_A and p_B in Z_B, such that d_ε is the network distance between p_A and p_B. The conditions under which d_ε will provide an ε-approximation of the network distances between Z_A and Z_B are discussed in this section. The algorithm uses a list Q of block pairs drawn from T. At the start of the algorithm, Q is initialized with a block pair formed by the root block of T, as shown in line 1. The algorithm stops when Q is empty.

At each iteration the algorithm retrieves the first block pair (A, B) in Q. If A and B correspond to the same block in T but are not *leaf* blocks, then A and B are both split into their $C = 2^d$ children blocks, and the resulting C^2 block pairs are inserted into Q as shown in lines 6-9. If A (B) correspond to the same leaf block, it is simply discarded.

If A and B refer to different blocks in T, then the algorithm examines if the block pair (A, B) is well separated, which is done as follows: We first choose two representative points $p_A \in A$, $p_B \in B$ (line 12) either using a randomized or some deterministic strategy. We then estimate the maximum of the *network diameters* (or an over-approximation of the network diameters) of A and B, which is defined as the farthest vertex in A (or B) from p_A (or p_B) using a network distance measure (line 14). Estimating the network diameter r can be done in a number of ways; a few strategies are discussed in Section 4.3. If the ratio of the network distance $d_G(p_A, p_B)$ to the network diameter r is greater than or equal to $s = \frac{2}{\varepsilon}$, then the block pairs A, B are well separated, in which case, the triple $\{Z(A), Z(B), d_G(p_A, p_B)\}$ is added to L (lines 15 and 16). Note that we show in Section 4.4 that if A, B are well separated using a separation factor $s \geq \frac{2}{\varepsilon}$, then $d_G(p_A, p_B)$ is indeed an ε-approximation of the network distances between A and B.

If the block pair (A, B) is not well separated, then both A and B are split into their C children blocks, provided they are nonleaf, and the resulting pairs are inserted into Q as shown in lines 18-28.

We now briefly show that Algorithm 1 decomposes G into a set of triples $(Z(A), Z(B), d_\varepsilon)$ that satisfies the properties of a WSPD. First of all, we distinguish between block pairs in Q that point to the same block in the PR-quadtree T on V, which are referred to as SINGLETONS, and those that point to different blocks in T, which are referred to as PAIRS. Moreover, we collectively refer to A, B as *heads* in the discussion below. The list Q is initialized with the SINGLETON (ROOTOF(T), ROOTOF(T)) in line 1 at the start of the algorithm. During the course of the algorithm, if a SINGLETON (A, A) is retrieved from the top of Q, then it is replaced with C SINGLETONS and C^2 PAIRS formed by the children nodes of A (or B). If a PAIR (A, B) is retrieved from the top of Q, then it is replaced with C^2 PAIRS formed by the children nodes of A and B. By induction, we can show that every block in the quadtree is retrieved from Q as a SINGLETON. To show that the heads of the reported triples are disjoint (Property 2), we point out that only SINGLETONS have overlapping heads (owing to the fact that quadtrees are nondisjoint space decompositions), but only PAIRS are reported as triples. We now show that any vertex pair u, v is contained in one and only one of the triples in the output L (Property 3). To do this, we use contradiction. Assume that u and v are contained in two of the triples in L, say $(A^i, B^i, d^i_\varepsilon)$ and $(A^j, B^j, d^j_\varepsilon)$ *s.t.*, $i \neq j$, $u \in A^i, A^j$, and $v \in B^i, B^j$. Let M be the nearest common ancestor block of u, v in the quadtree. Before M is retrieved from the top of Q as a SINGLETON, u and v are contained in the same head. When M is decomposed, u and v are no longer contained in any SINGLETON, but are present in different blocks, which may not yet have satisfied the ε-approximation guarantees. After each subsequent decomposition, only one PAIR contains both u and v. Thus, the nature of our decomposition process makes it impossible for u and v to be contained in both (A^i, B^i) and (A^j, B^j) *s.t.*, $i \neq j$. Similarly, we can also show that given a pair of vertices u, v, it is exactly contained in one of the triples in L. Finally, as every vertex pair is contained in exactly one of the triples in L, we have also shown that algorithm captures all the n^2 network distances in G. Hence, given a vertex pair u, v, we are guaranteed that there exists exactly one triple $(Z_A, Z_B, d_\varepsilon)$ in L, such that Z_A contains u and Z_B contains v.

Algorithm 1.
Procedure BUILDORACLE[G, T, ε]
Input: $G \leftarrow$ spatial network $G(V, E)$
Input: $T \leftarrow$ PR-quadtree on the spatial positions of V
Input: $\varepsilon \leftarrow$ desired approximation; $\varepsilon > 0$
Output: $L \leftarrow$ set of triples $(Z_A, Z_B, d_\varepsilon)$; initially empty
($* s \leftarrow \frac{2}{\varepsilon}$; separation factor $*$)
($* Q \leftarrow$ list of block pairs; initially empty $*$)
1. INSERT(Q, ROOTOF(T), ROOTOF(T))
2. **while** (ISNOTEMPTY(Q)) **do**
3. $(A, B) \leftarrow$ TOP(Q)
4. **if** $A = B$ **then**
5. ($*$ reject if A (B) is a LEAF block $*$)
6. **if** ISNOTLEAF(A) **then**
7. Split A, B each into C children blocks
8. Insert C^2 children block pairs of A, B into Q
9. **end-if**
10. **else**
11. ($*$ Choose representative points $*$)
12. $p_A \leftarrow$ CHOOSEREP(A); $p_B \leftarrow$ CHOOSEREP(B)
13. ($*$ Estimate diameter r of A and B $*$)
14. $r \leftarrow$ MAX(DIAMETER(A), DIAMETER(B))
15. **if** ($\frac{d_G(p_A, p_B)}{r} \geq s$) **then**
16. INSERT(L, $\{Z_A = Z(A), Z_B = Z(B), d_\varepsilon = d_G(p_A, p_B)\}$)
17. **else**

18. **if** ISNOTLEAF(A) **then**
19. $L_a \leftarrow \{C$ children blocks of $A\}$
20. **else**
21. $L_a \leftarrow \{A\}$
22. **end-if**
23. **if** ISNOTLEAF(B) **then**
24. $L_b \leftarrow \{C$ children blocks of $B\}$
25. **else**
26. $L_b \leftarrow \{B\}$
27. **end-if**
28. INSERT pairs in $L_a \times L_b$ into Q
29. **end-if**
30. **end-if**
31. **end-while**
32. **return** L

Finally, we compute the cost of constructing the distance oracle using Algorithm 1. We have shown above that the output of the distance oracle is a WSPD of the vertices in a spatial network, which can be constructed in $O(n \log n + s^d)$ time [15]. We still have to account for the cost in determining if two subsets of vertices A and B are well separated using a network distance measure, which involves a shortest path computation between representative points p_A of A and p_B of B. Note that we assume here that finding the distance between the representative points is more expensive than finding the network diameter of a quadtree block, which is true in practice. In order to estimate how many shortest path computations are performed by the algorithm, for every block in the quadtree, we choose an arbitrary vertex as its representative point. As the number of nodes in the quadtree is $O(n)$, we will end up with $O(n)$ representative points. Given a block A in the quadtree, the algorithm would form $O(s^d)$ block pairs of the form (A, B), although note that not of all of these block pairs will ultimately form WSPs. This means that given a block A, $O(s^d)$ shortest paths are invoked with the representative point of A as the source vertex. So, the total number of shortest path computations that will be performed is $O(s^d n)$, which would take $O(s^d n^2 \log n)$ time using Dijkstra's algorithm. The total cost of constructing a distance oracle is $O(s^d n^2 \log n)$ as the cost of shortest path computation dominates the cost of constructing a WSPD on the vertices of a spatial network.

4.3 Estimating Network Diameter

Given a vertex p_A and a block in the PR-quadtree A in T corresponding to a set of sources, we define the network diameter r of A as the farthest vertex from p_A in A using a network distance measure. One simple strategy to computing the network diameter of A is to obtain the network distance from p_A to other every vertex in A. The maximum of those network distance values is the diameter of A. However, this approach may be expensive to compute and may not be a scalable solution. Our strategy is to compute an over-approximation r of the network diameter of A if it is easier to compute compared to the exact network diameter of A. However, this strategy has the unfortunate consequence that as r is an over-approximation of the network diameter of A, Algorithm 1 would have to split the block pairs much more than necessary in order to make them well separated. Consequently, there is a trade-off between the time spent on computing the network diameter of a block and the total storage space taken up by the oracle. Below, we discuss several strategies to compute the network diameter of a set of vertices contained in block A of the PR quadtree.

1. Given a block pair (A, B), we first obtain the network distance $d_G(p_A, p_B)$ between the representative points $p_A \in A$ and $p_B \in B$. We then apply an early terminating variant of Dijkstra's algorithm from p_A (p_B) that takes advantage of the *incremental* nature of Dijkstra's algorithm. That is, Dijkstra's algorithm with p_A (p_B) as a starting vertex visits vertices in G in an increasing order of their network distance from p_A (p_B). The algorithm terminates when it encounters a vertex that is farther than $\frac{d_G(p_A,p_B)}{s}$ from p_A (p_B). We now check to see if all the vertices in A (B) have already been visited by Dijkstra's algorithm. If yes, then A and B are well separated. Note that this method of determining if a block pair (A, B) is well separated is only applicable to spatial networks that are undirected.
2. If r' is the diameter of the geometric bounding box of A, the network diameter of A can be over-approximated by $\gamma_H r'$, which can be computed using Theorem 3.3, or using the distortion spectrum of the spatial network.
3. Use the approach of Goldberg and Harrelson [8] which first selects a set of vertices, termed *landmarks*, at random. The network distance from each of the landmark vertices to all the vertices in G is precomputed. Once precomputed, the diameter of A (B) can be upper bounded using the triangle inequality and the network distance to the nearest landmark.

4.4 Analysis

This section provides bounds on the size of the distance oracle of G by appealing to the equivalence between the decomposition of a spatial network in Algorithm 1 and the WSPD of a point set. We now show how to extend the notion of a WSPD in terms of a spatial distance to one in terms of a network distance. This is captured by Lemma 4.2 below.

Lemma 4.2. *Given an s-WSPD of the vertices V of a spatial network $G(V, E)$ based on a spatial distance also yields a s'-WSPD of V using a network distance with $s' = s \cdot \frac{\gamma_L}{\gamma_H}$.*

Proof. Given a WSP, (A, B) on the decomposition of $V \otimes V$ using the spatial distance measure, the minimum spatial distance between A and B is at least $s \cdot r$, where r is the larger of the diameters of A and B.

Consider two vertices u, v in A (or B). We have $d_G(u, v) \leq \gamma_H \cdot d_S(u, v) \leq \gamma_H \cdot r$ as $d_S(u, v) \leq r$ by virtue of r being the diameter of A or B. r', the maximum value of $d_G(u, v)$, is the diameter of A (and B) using a network distance measure and we have that $r' \leq \gamma_H \cdot r$. Therefore, the spatial distance diameter of A (or B) is scaled by at most a factor of γ_H to obtain the network distance diameter r'.

Considering a vertex pair (a, b), such that $a \in A, b \in B$, we have from the WSP condition and Assumption 1 (see Section 3) that:

$$s \cdot r \leq d_S(a, b) \leq \frac{d_G(a, b)}{\gamma_L}. \quad (4)$$

Replacing r with $\frac{r'}{\gamma_H}$ in (4), we obtain $s \cdot \frac{r'}{\gamma_H} \leq d_S(a,b) \leq \frac{d_G(a,b)}{\gamma_L}$. The above relationship between the minimum and maximum bounds on $d_S(a,b)$ can be rewritten as $r' \cdot s \cdot \frac{\gamma_L}{\gamma_H} \leq d_G(a,b)$. Now, letting $s' = s \cdot \frac{\gamma_L}{\gamma_H}$, leads to the desired result $s' \cdot r' \leq d_G(a,b)$, which is equivalent to saying that A and B are well separated using the network distance measure with a separation factor of s'. □

We now show that a WSPD of the vertices of a spatial network is a realization of an approximate distance oracle.

Lemma 4.3. *Let (A,B) be a WSP in the s-WSPD of G using a network distance measure, such that u^*, v^* are the representative points of A and B, respectively. The network distance $d_G(u^*,v^*)$ between the representative points is an $\varepsilon = \frac{2}{s}$ approximation of the network distance $d_G(u,v)$ between any pair of vertices (u,v), such that $u \in A$ and $v \in B$.*

Proof. Given a pair of vertices (u,v), such that $u \in A, v \in B$, we know from the triangle inequality that

$$d_G(u^*,v^*) - d_G(u,u^*) - d_G(v^*,v) \leq d_G(u,v),$$
$$d_G(u,u^*) + d_G(u^*,v^*) + d_G(v^*,v) \geq d_G(u,v).$$

Without loss of generality, we assume that $d_G(v^*,v) \geq d_G(u,u^*)$. Substituting above, we get

$$d_G(u^*,v^*) - 2d_G(v^*,v) \leq d_G(u,v),$$
$$d_G(u^*,v^*) + 2d_G(v^*,v) \geq d_G(u,v),$$

$$d_G(u^*,v^*)\left(1 - \frac{2d_G(v^*,v)}{d_G(u^*,v^*)}\right) \leq d_G(u,v),$$
$$d_G(u^*,v^*)\left(1 + \frac{2d_G(v^*,v)}{d_G(u^*,v^*)}\right) \geq d_G(u,v).$$

In line 15 of Algorithm 1, we ensure that the condition $\frac{d_G(u^*,v^*)}{d_G(v^*,v)} \geq s$ is satisfied for all vertices in B. Substituting it above,

$$\left(1 - \frac{2}{s}\right) d_G(u^*,v^*) \leq d_G(u,v) \leq \left(1 + \frac{2}{s}\right) d_G(u^*,v^*).$$

Substituting, $\varepsilon = \frac{2}{s}$, we get

$$(1-\varepsilon) d_G(u^*,v^*) \leq d_G(u,v) \leq (1+\varepsilon) d_G(u^*,v^*).$$

□

At this point, having established that $\varepsilon = \frac{2}{s}$, we now obtain a bound on the size of the distance oracle.

Lemma 4.4. *For a given value of $\varepsilon = \frac{2}{s}$, the size of the oracle produced by Algorithm 1 is no worse than $O((\frac{\gamma_H}{\varepsilon\gamma_L})^d n)$.*

Proof. Let (A,B) be a WSP, such that u^*, v^* are the representative points of A and B, respectively. We assume that A (B) is contained in a bounding hypersphere of diameter r. The network diameter of A and B is bounded by

$$\gamma_L r \leq \text{DIAMETER}(A) \leq \gamma_H r$$
$$\gamma_L r \leq \text{DIAMETER}(B) \leq \gamma_H r.$$

As (A,B) is a WSP, $d_G(u^*,v^*)$ can be similarly bounded by

$$d_G(u^*,v^*) \leq \gamma_H r s.$$

The effective separation factor s' of the WSPD is $\frac{s\gamma_H}{\gamma_L}$. Hence, the worse case storage requirement of the oracle is $O((\frac{\gamma_H}{\varepsilon\gamma_L})^d n)$. □

This leads us to the final result of this section:

Theorem 4.5. *Given a spatial network $G(V,E)$, we can construct an oracle of size $O(\frac{n}{\varepsilon^d})$ to retrieve the ε-approximate network distance between any vertex pair in $O(\log n)$ time.*

The real utility of the above theorem is to establish the linear size of our distance oracle. Note that the constants of proportionality estimated using an empirical analysis were found to lie, in most cases, between 1.5 and 3 which is much smaller than the worse case bound of $(\frac{\gamma_H}{\gamma_L})^d$ established in Lemma 4.4.

5 Query Processing

A major data engineering problem in databases is that spatial networks cannot be easily represented using the relational model and, furthermore, operations on it cannot be cast in terms of relational operators, namely *selection*, *projection*, and *joins*, etc. In this paper, we introduced an approximate distance oracle which is stored as a *relation* in a database system indexed by a B-tree. We now show how operations on spatial networks can be cast in terms of relational operations on the distance oracle relation. In particular, we demonstrate how to perform region search [6], k-nearest neighbor [29], and spatial joins [30], [31] using relational operations expressed using SQL language. Our proposed setup is very desirable from a systems point of view because, now, all operations can be performed in the context of a database system. Our work opens up the use of a commercial database for building interactive applications (e.g., GIS applications [32]) on spatial networks. The real success of our method lies not so much in being able to succinctly capture the n^2 network distances in a spatial network using only $O(n)$ space, but rather in the ability of our oracle construct to be seamlessly integrated into a relational database system, indexed using a traditional indexing scheme (e.g., B-tree), and used in complicated query processing scenarios without making any significant changes to existing database systems. We now describe strategies for storing, indexing, and query processing using an ε-approximate distance oracle on a relational database system.

5.1 Storing the Oracle

The output of Algorithm 1 is a list L of triples $(Z_A, Z_B, d_\varepsilon)$ which is stored as database relation O, where Z_A and Z_B are Morton blocks corresponding to sets of sources and destinations, respectively, such that the network distances between the sources in Z_A and destinations in Z_B are approximated using d_ε. In the context of a database, O is a relation whose attributes are Z_A, Z_B, and d_ε. Moreover, each tuple in O is associated with a unique tuple number, denoted by *tid* which serves as the *primary key* of O. Note that we no longer need to retain the original graph representation of the spatial networks, or for that matter, even the positions of the vertices and edges in the spatial network. Now, query processing using the distance oracle O, requires efficient indexes on O such that given a

source location p and a destination location q, we can obtain the ε-approximate network distance between p and q efficiently. We can do this by first computing the Morton block $Z(p), Z(q)$ corresponding to the spatial positions of p and q with bit operations in $O(1)$ time. Using the indexes on O, we can obtain a tuple (A, B, d_ε) in O such that A contains $Z(p)$ and B contains $Z(q)$ in which case, d_ε approximates the network distance between p and q. This operation can be described using an SQL construct which corresponds to applying a "selection" operator on the oracle relation O.

SELECT $O.d_\varepsilon$ FROM O WHERE $O.Z_A =_Z Z(p)$ and $O.Z_B =_Z Z(q)$

5.2 Indexing the Oracle

Our goal here is to devise an index that enables accessing the distance oracle in $O(\log n)$ time. Queries on O are processed differently by the database depending on the kind of indexes on O. The simplest strategy is to build a pair of B-tree indexes, one each on Z_A and Z_B. Given p, q, the query processing engine will first use $Z(p)$ as the search key on the B-tree on Z_A to obtain a list T_A^p of tids in O whose Z_A values contain $Z(p)$. Next, using $Z(q)$ as the search key on the B-tree on Z_B, we can obtain a list T_B^q of tids in O whose Z_B values contain $Z(q)$. Now, we need to find the identity of the common tuple between the lists T_A^p and T_B^q, whose d_ε value approximates the network distance between p and q. From the property of the WSPD, we are guaranteed that there will be exactly one tid in common between T_A^p and T_B^q. The asymptotic complexity of this search process can be calculated by taking the sum total of the individual steps. Given any search key $Z(p)$, there are $O(s^d \log n)$ tuples in O whose Z_A values contain $Z(p)$. To locate all of these tuples, can take $O((\log n + s^d) \cdot \log n)$ time in the worse case as it may need $O(\log n)$ searches on the B-tree, each search taking $O(\log n + s^d)$ time. Finding the common tid will take $O(s^d \log n)$ time because every element in T_A^p and T_B^q will have to be examined. Summing all these steps, the cost of finding an ε-approximate network distance between p and q will take $O(\log^2 n)$. Another alternative is to use a B-tree on (Z_A, Z_B) which is essentially a B-tree on Z_A whose leaves are B-trees, called *treelets*, on Z_B for those tuples in O having the same Z_A. Such a situation obviates the need for intersecting two lists of tids, but does not reduce the cost of the initial search process. The complexity of the search process is $O(\log^2 n)$ as we need to perform $\log n$ searches, each search taking $\log n + d \cdot \log s$ time.

There are several other strategies of indexing (Z_A, Z_B), but they still do not provide an $O(\log n)$ access time. We can index (Z_A, Z_B) using a B-tree where the primary sort is based on Z_A while Z_B is used as a secondary sort key. Using the level of Z_A, we can create two B-tree variants that have slightly different behavior. The first approach orders objects based on the ordering imposed by $(Z_A, \text{LEVEL}(Z_A), Z_B)$. That is, the primary sort key is Z_A with the LEVEL of Z_A serving to break ties between Morton blocks that overlap one another, and Z_B serving as the secondary sort key. A B-tree using such a sort comparator leads to a depth-first ordering of the blocks forming Z_A in the PR-quadtree on V. The desirable property of such an arrangement is that given a source location p, the tuples whose Z_A attributes contain $Z(p)$ are more or less *clustered* together which means that they are stored in nearby disk pages in the B-tree. On the other hand, the ordering imposed by $(\text{LEVEL}(Z_A), Z_A, Z_B)$ will result in a breadth-first ordering of the blocks forming Z_A. This means that the bulk of the n^2 network distances will be captured in the first few disk pages of the B-tree, which makes it suitable for caching. Unfortunately, none of these strategies provide an $O(\log n)$ access time because we are imposing an ordering on (Z_A, Z_B), which is essentially two-dimensional (i.e., Z_A and Z_B), by establishing a primary ordering on Z_A and a secondary ordering of Z_B. The resulting structure is similar to the two-dimensional *range* tree of Bentley [6], [33] (as is the treelet structure described earlier), which can answer range queries in $O(\log^2 n + f)$ time, where f is the number of tuples in the result set.

The only strategy to obtain an $O(\log n)$ access time is to store the distance oracle as a relation of schema $O(Z_{AB}, d_\varepsilon)$, which is similar as before except that instead of storing Z_A and Z_B as separate attributes, we merge them to form a single four-dimensional Morton block representation. The attribute Z_{AB} in O is indexed using a B-tree. Storing Z_A and Z_B as a single four-dimensional Morton block requires that initially Z_A and Z_B are of the same level (i.e., size). Now, Algorithm 1 does not always produce blocks Z_A and Z_B of the same level which means that the output L of the algorithm has to be decomposed further so that Z_A and Z_B are made to be the same level. We first show that such a transformation of L does not change the number of triples in L, which is given by the following lemma:

Lemma 5.1. *The output of Algorithm 1 can be decomposed further so that Z_A and Z_B are of the same level without incurring any additional storage.*

Proof. Algorithm 1 breaks up block pairs symmetrically until one of the blocks is a leaf, in which case, only the nonleaf block is further subdivided. At the end of the algorithm, if a block pair (Z_A, Z_B) has both nonleaf blocks, then they are of the same level. On the other hand, if one or both blocks are leaf blocks, then either Z_A or Z_B has to be decomposed further to make them the same level. If Z_A is a leaf block but the level of Z_A is less than Z_B, then we keep subdividing Z_A into its C children blocks until it is the same level as Z_B. Each time we subdivide Z_A it results in $C - 1$ empty blocks which can be discarded and one nonempty block that contains the point which is subdivided further. This means that Z_A and Z_B can be made the same level without creating any additional block pair. Next, if Z_A and Z_B are both leaf blocks such that the level of Z_A is less than the level of Z_B, we can apply the same strategy as before to make them the same level without incurring any additional storage. The more interesting case is when Z_A is a nonleaf block whose level is less than Z_B which is a leaf block. We can show that given that our decomposition rule in Algorithm 1, this configuration is not possible. If Z_A is a nonleaf block whose level is less than that of Z_B which is a leaf would mean that Z_B was decomposed more times than Z_A, even though Z_A is not a leaf block, which is not possible. The other three symmetric cases with Z_A and Z_B interchanged follow a similar argument. This means that the output of Algorithm 1 can be further decomposed so that Z_A and Z_B are at the same level without incurring any additional storage. □

Given an approximate distance oracle $O(Z_{AB}, d_\varepsilon)$ such that Z_{AB} is indexed using a B-tree, we can find the approximate network distance between a source location p and a destination location q, by first computing a four-dimensional Morton block $Z_4(p,q)$ which uses the positions of p and q. Using $Z_4(p,q)$ as the search key on the B-tree on Z_{AB} yields exactly one tuple in O such that Z_{AB} contains $Z_4(p,q)$. The cost of the search is $O(\log n)$ as making Z_A and Z_B the same level did not increase the size of the distance oracle, which is still $O(n/\varepsilon^d)$. This leads to a very efficient organization of the distance oracle and reduces the cost of searching for an approximate network distance to $O(\log n)$, which is an improvement from before. The SQL query to obtain the ε-approximate network distance using this schema is given by the following:

SELECT $O.d_\varepsilon$ FROM O WHERE $O.Z_{AB} =_Z Z_4(p,q)$

5.3 Queries on the Oracle

We now discuss how to perform spatial queries on a spatial network using the ε-approximate network distance oracle stored as a relation $O(Z_{AB}, d_\varepsilon)$ of a predefined approximation. Let us assume the following setup: Let R be a relation of restaurants with schema $(\text{tid}, \text{pos}, \text{type}, \text{price})$, where *pos* is the position of the restaurants given by a two-dimensional point object, *type* is the type of the cuisine served by the restaurant, and *price* is the average cost. Furthermore, we assume that there is a B-tree on $Z(R.\text{pos})$. We also define another relation Q of movie theaters given by the same schema $(\text{tid}, \text{pos}, \text{movie\_id})$ where *pos* is the position of movie theater, and *movie_id* is a movie playing in the theater. Also, $Z(Q.\text{pos})$ is indexed using a B-tree. We present the following queries on a spatial network:

ε-Approximate Network Distance: Given a source p, and destination q, find the ε-approximate network distance between them.

SELECT $O.d_\varepsilon$ FROM O WHERE $O.Z_{AB} =_Z Z_4(p,q)$

Region Search: Given a query location q, find all restaurants in R that are within 10 miles of q that serve *Italian* cuisine.

SELECT $R.\text{pos}$, $O.d_\varepsilon$ FROM R, O WHERE
$O.Z_{AB} =_Z Z_4(q, R.\text{pos})$ and $R.\text{type} =$ "Italian"
and $O.d_\varepsilon \leq 10$ miles

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

SELECT $R.\text{pos}$, $O.d_\varepsilon$ FROM R, O WHERE
$O.Z_{AB} =_Z Z_4(q, R.\text{pos})$ and $R.\text{type} =$ "Italian"
ORDER BY $O.d_\varepsilon$ LIMIT k

Distance Join: Find the k closest pairs of restaurants in R and movie theaters in Q.

SELECT $R.\text{pos}$, $Q.\text{pos}$, $O.d_\varepsilon$ FROM R, Q, O
WHERE $O.Z_{AB} =_Z Z_4(R.\text{pos}, Q.\text{pos})$
ORDER BY $O.d_\varepsilon$ LIMIT k

Each of the above operations are simple relational operations on the oracle relation that uses a B-tree. A commercial database can optimize complicated query processing scenarios involving B-trees. For example, the ε-approximate network distance query and the region search are simple selection operators using the B-tree index. The rest of the queries involve simple join operations aided by a B-tree which can be efficiently handled by a query optimizer. In short, query processing on spatial networks can be easily integrated into a traditional database system. Finally, our strategy relies on the precomputation of a distance oracle for a prespecified value of ε. For example, a distance oracle for the road network of the US can be precomputed and commercially distributed. This will then enable query processing on any spatial data set residing on the road network of the US using a commercial database. Moreover, the wide use of such an oracle will justify the large cost of the precomputation, provided that an appropriate value of ε is chosen.

6 Query Optimization

In this section, we discuss how to optimize queries on spatial networks expressed using SQL statements with an ε-approximate distance oracle. From the above discussion, we see that expressing operations on a spatial network with the help of an ε-approximate distance oracle is no different than performing these operations in a euclidean space, except that operations now involve an additional join with the distance oracle relation. We now illustrate how a relational database system efficiently processes some of these queries expressed as SQL commands.

Any spatial query that is expressed using SQL can be converted into an *operator tree* [34], which is a computational tree made up of spatial and nonspatial relational operators. Fig. 5a is an example of an operator tree of an SQL query that performs region search on a spatial network. Note that the operator tree shown in the figure is just one of many possible strategies for answering a query. In other words, there are several ways of arriving at the correct answer and the job of generating all possible strategies and choosing among them is typically done by a query optimizer. Note that even though there are many ways of answering a query, they typically have different associated *costs*. The database system chooses the one with the least cost. Our discussion on query optimization of queries involving distance oracles will be limited to the setup described in the previous section. Moreover, for each query, we only discuss one unoptimized way of answering it and then, we proceed to describe strategies for making it more efficient.

Given an SQL tree expressed as an operator tree [34], the database system has enough knowledge to process these queries correctly in an efficient manner. However, there are some difficulties in optimizing queries involving the distance oracle unless the query optimizer is made aware of certain additional strategies it can employ to make the execution of these queries efficient. For example, one scenario where optimization of queries involving distance oracles becomes difficult is if the user is only interested in a small subset of tuples, where the criterion for selecting relevant tuples is expressed either directly or indirectly in terms of network distances. We elaborate on this problem using the following example: Consider a query to find all the restaurants, given by the relation R, that are closer than 10 miles on a spatial network from an input address (i.e., point) q. One way of processing this query is to obtain the

approximate network distances of every restaurant in R from q by joining the tuples of R with O, and then subsequently pruning away all restaurants that are farther than 10 miles on the spatial network from q. Now, it is clear that this solution strategy is not very efficient, especially if R is a large relation such as the one containing all the restaurants in the US. Another variation of this query is if the user is only interested in the "top k" closest restaurants to q in R, in which case, obtaining the network distances of all the restaurants in R from q and subsequently retaining the k closest to q results in wasted work. Both these inefficiencies occur because we cannot really exclude any restaurant in R without first determining its network distance from q, which is not known until R is joined with O. The fundamental problem is that the network distance, in our setup, is not a *computable* quantity but, instead, is retrieved from O. This means that a join of R with O seems almost inevitable at runtime as the identity of the query point q (or a relation S in case of distance joins) is only known at the time of execution of the query.

The proposed strategy for making query processing using a distance oracle efficient is to make the query optimizer aware that the spatial distance between two objects on a spatial network always lower bounds the network distance between them. If the query processor uses this strategy effectively, then it can prune away most of the tuples in R using this relationship between the spatial and network distance functions. To briefly illustrate how we can use this strategy, in the case of the network range query, the query optimizer can first restrict the output to only contain those tuples in R whose spatial distance from q is less than 10 miles, which are then joined with distance oracle relation O, after which it again prunes those restaurants in R that are farther than 10 miles. In the rest of this section, we show how a query optimizer can modify the operator tree suitably so that queries involving the distance oracle can be processed correctly and efficiently.

Our discussion uses the same database setup as in Section 5. In particular, we assume that the spatial attributes on R and S have some spatial indexes on them, but we place no restriction on their type which can be either a quadtree (e.g., linear quadtree) [6], [28], an R-tree [35], or others. We express queries in SQL using an operator tree whose nodes are made up of algebraic operators, such as selection, projection, or join. For the sake of clarity, we now describe a few of the spatial relational operators that are needed for query processing which are commonly available on commercial database systems.

Index Scan: A spatial index such as a quadtree is an ordering of the tuples in a relation based on the space that they occupy. A spatial index allows for the browsing or retrieval of the tuples in a relation in many different ways, which we refer to as an *index scan*. For example, an index scan of R can retrieve tuples based on the distance from a query point q either in an increasing or decreasing order of distances from q. It can even obtain tuples based on containment or noncontainment within a geometric shape, or some combination of distance and geometric constraints. An index scan can also be made *incremental*, which means that we need not specify how many tuples are needed before the start of the query but can keep obtaining more tuples until all the tuples in R have been exhausted. One method for incrementally retrieving tuples in R based on distances from q is the best first nearest neighbor method of Hjaltason and Samet [6], [30].

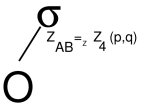

Fig. 4. Approximate network distance query using a distance oracle.

Spatial Select: A spatial select operator, represented by γ, is analogous to its nonspatial variant in the sense that it allows for the pruning of tuples in a relation based on a set of spatial constraints. An example of a spatial selection operation is, given a relation of restaurants in the US, we wish to obtain only those restaurants that are in New York city, which is represented by a polygon.

Spatial Join: Given two relations R and S, the distance join $R \bowtie_{dist} S$ of R and S generates pairs of tuples (p, q) such that p is drawn from R and q from S, usually subject to an additional constraint involving the values of the spatial attributes of R and S. If the output pairs are ordered either in an increasing or decreasing order of distances between p and q, such a join is referred to as a distance join [30], [36]. Our processing of spatial network distance join queries uses the incremental distance join algorithm of Hjaltason and Samet [30].

In order to process SQL queries with LIMIT constraints, we use two SQL operators—STOP and RESTART. These operators have been adapted from [37] and are described briefly below.

STOP: As the name suggests, this operator suspends the output of an operator *subtree* once a predetermined condition is satisfied. The condition can either be specified in terms of a limit on the numbers of tuples produced by an operator subtree, or some other constraint determined at runtime.

RESTART: Restarts a previously suspended operator subtree so that more tuples can be produced. A subtree of an operator tree is said to be *restartable* only if it can produce more tuples without having to recompute all the previously computed tuples. This is where incremental approaches in spatial database literature [6], [29], [30] excel as they are restartable.

At this point, we have defined all the operators required for our discussion on query processing in spatial networks. We now show how a query optimizer can optimize queries on spatial networks using the distance oracle with the aid of the examples from the previous section.

Approximate Network Distance. Given a source p and a destination q on a spatial network, obtaining an ε-approximate network distance between p and q involves a selection on O with $Z_4(p, q)$ as the search key. Fig. 4 shows the operator tree corresponding to this query. Such a query involves a simple selection operator on the oracle relation O with the aid of the B-tree on Z_{AB}.

Region Search. Fig. 5a shows the operator tree corresponding to a query that finds all the Italian restaurants within 10 miles of a query point q. One way of processing such a query is to first determine the ε-approximate network distance of all the Italian restaurants in R, and then prune away all the restaurants in the output that are farther than 10 miles using a select operator. This is an inefficient approach as the query processing will visit every

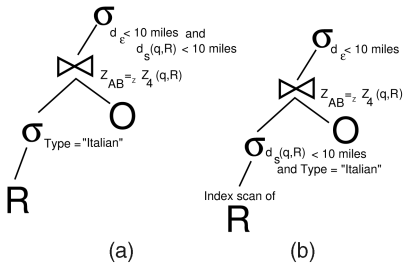

Fig. 5. (a) Unoptimized and (b) optimized region search query using a distance oracle.

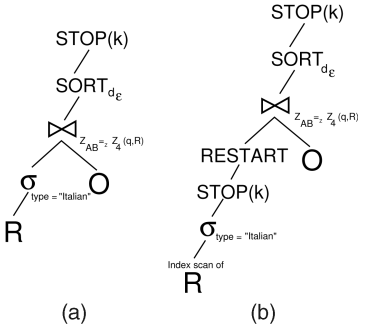

Fig. 6. (a) Unoptimized and (b) optimized nearest neighbor search query using distance oracle.

tuple in R, even the ones that are much further than 10 miles from q which will result in wasted work.

A more efficient way of performing this query, shown in Fig. 5b, uses the property that the spatial distance between any two points on a spatial network always lower bounds the network distance between them. Our optimized algorithm first performs an index scan on R which is an incremental retrieval of the restaurants in R based on the spatial distance from q. The index scan stops when we obtain a restaurant in R whose spatial distance from q is farther than 10 miles. Among this subset of restaurants, we choose only those that serve Italian food, which are subsequently joined with the distance oracle relation O to yield their network distances from q. We select only those restaurants that are within 10 miles of q in terms of network distance. The correctness of this approach can be easily seen by observing that any restaurant in R that was pruned away because its spatial distance from q was greater than 10 miles cannot be at a distance of less than 10 miles from q using a network distance function as the spatial network distance lower bounds the network distance between them.

Note that it is conceivable that a restaurant r in R that is at both spatial and network distances that are a bit more than 10 miles from q, while having an ε-approximate network distance from q that is less than 10 miles owing to the approximation. In this case, our optimized algorithm will miss reporting r even though its approximate network distance is less than 10 miles because r would not be in the initial set of restaurants whose spatial distance from q is less than 10 miles. We can overcome this problem by increasing the above spatial distance restriction to $(1+\varepsilon)\cdot 10$ miles, which would mean that r will now be reported in the result, but we choose not to do so as r is clearly an incorrect answer so there is no point to report it. In order to make the results of both the optimized and unoptimized versions of the algorithm identical, we need to add an additional constraint to the select operator in our unoptimized algorithm in Fig. 5a to prune away all those restaurants that are farther than 10 miles in the spatial distance measure.

k-Nearest Neighbor Search. Fig. 6a shows an unoptimized version of a k-nearest neighbor search on a spatial network which suffers from the same drawback as the region search in Fig. 5a in the sense that the algorithm ends up retrieving the network distances of all Italian restaurants in R, even though we only require a small subset of the Italian restaurants (i.e., k of them) in the result set. In particular, we first select all tuples in R that are Italian restaurants which are then joined with the distance oracle relation O to yield the network distances of R from q. We only retain the top k tuples from this set via the use of the STOP operator which corresponds to the k closest restaurants in R to q on the spatial network.

We optimize the k-nearest neighbor query on spatial networks by obtaining the tuples in R in an increasing order of network distance from q. Our approach follows the INE algorithm of Papadias et al. [2]. We first obtain the closest k Italian restaurants to R to q in terms of the spatial distance function, at which time the query subtree is suspended by the STOP operator. Now, these k tuples are joined with the distance oracle relation O to obtain their approximate network distances from q, which are then sorted to yield an upper bound on the network distance of the k farthest restaurant in R, which we refer to as d_k. Now, we restart (via RESTART) the earlier suspended query subtree to yield more tuples incrementally in an increasing order of spatial distance from q, until we obtain a restaurant in R whose spatial distance to q is greater than d_k. These additional tuples are also joined with O to yield their network distances, at which point the closest k restaurants to q are chosen from a pool of restaurants from the initial set of k restaurants from before application of the STOP operator and the additional restaurants produced after application of the RESTART operator.

Distance Join. The distance join works in a similar manner to the k-nearest neighbor search in the sense that we simply replace a query point q with a relation (e.g., the coffee shop S relation in our example). As can be seen from Fig. 7, the main difference between the operator trees for the nearest neighbor and the distance join queries is the use of a distance join operation between R and S using the spatial distance function. We use an incremental distance join algorithm [30], which provides us with the ability to suspend and restart the operator as needed. The strategy for optimizing distance join queries on spatial networks is the same as before. It starts with application of the incremental distance join generating k pairs, whose network distances are ascertained by a join with O. We compute an upper bound on the network distance between the k farthest pair in the output which we refer to as d_k. Now the distance join operation is restarted and more pairs are generated until we obtain a pair whose spatial distance is farther than d_k, at which time, the operator is suspended once again. After obtaining the network distances of the additional pairs, we choose the k closest pairs from the

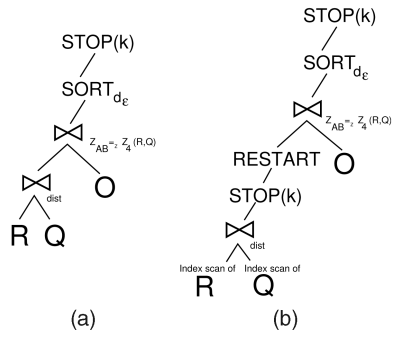

Fig. 7. (a) Unoptimized and (b) optimized distance join query using distance oracle.

original k pairs as well as the newly generated pairs.

7 Experimental Results

In this section, we report the results of our experimental evaluation of the oracles of size $O(1)$ and $O(n)$. We did not evaluate the $O(n \log n)$-size and $O(1)$ execution time oracles as they were presented primarily as a theoretical exercise to show the interplay between optimal execution time and space requirements, although, of course, there is really no justification for their use. The experiments were run on a Linux (2.4.2 kernel) quad 2.4 GHz Xeon server with one gigabyte of RAM. We implemented our algorithms using GNU C++. A number of publicly available road network data sets were used in the evaluation. These were obtained from the US Tiger Census [39] and the National Atlas [40] websites. In particular, we used a data set containing all the major roads in USA (i.e., more than 380,000 vertices and 400,000 edges). Sample random rectangular regions were drawn from the data set and the road network segments contained completely within them were extracted to serve as inputs to the evaluation. By taking the samples at random we were able to account for variations of road networks such as rural versus urban, and spatial network configurations that would lead to different sizes of the oracle.

Fig. 8 shows the maximum distortion γ_H of spatial networks obtained by applying the algorithm of Narasimhan and Smid [18], which is captured by Lemma 3.2, to spatial networks of varying sizes. The value of γ_L for all the input spatial networks was one. Note that this need not always be the case. For example, if the edge weights are in terms of the time taken to travel the edge, and spatial distance is in miles, then the value of γ_L would correspond to distortion of the edge in the spatial network with the lowest speed limit. From Fig. 8, we see that the value of γ_H for road networks can be very large and ranges between 10 and 1,000. An $O(1)$-size oracle, described in Theorem 3.3, that uses γ_L and γ_H to provide approximate network distances cannot provide a reasonable answer for query processing as the resulting error $\varepsilon = \frac{\gamma_H - \gamma_L}{\gamma_H + \gamma_L} \approx 1(100\%)$ is very large.

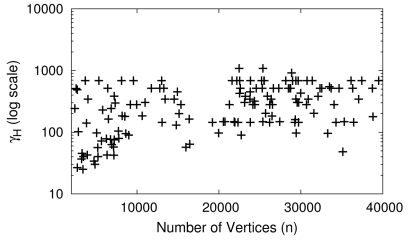

Fig. 8. Maximum distortion γ_H of different road networks.

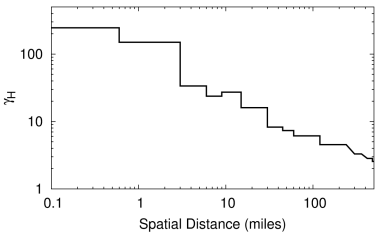

Fig. 9. Distortion spectrum of the eastern seaboard road network data set containing 91,113 vertices.

Next, we computed the distortion spectrum of a large road network data set corresponding to the important roads in the eastern seaboard states of USA, consisting of 91,113 vertices and 114,176 edges, shown in Fig. 9. As we can see, the maximum distortion for small spatial distances (less than 2 miles) can be very large. However, as the spatial distance between the source and destination increases (and is greater than 50 miles), the maximum distortion quickly reduces to a low value. Note that an approximate distance oracle of size $O(1)$ that uses the distortion spectrum of a spatial network may be suitable when the spatial distance between a given source vertex and destination vertex is large.

We now examine the characteristics of the $O(n)$-size oracle that has deterministic guarantees on the quality of the approximate answers it provides. We built the oracles by applying Algorithm 1 to the same road networks of different sizes used to obtain Fig. 8. Fig. 10 shows the effect of the size of the road networks, in terms of the number of vertices n, on the size of the resultant distance oracle, which is measured in terms of the number of Morton blocks normalized by n/ε^d. We chose $s = 8$ and $d = 2$ for this set of evaluations. It is easy to see that the size of the oracle does indeed follow $c \cdot n/\varepsilon^d$, where c is estimated empirically to lie between 1 and 6 and in most cases lies between 1.5 and 3 for the road networks used in our experiments. This study shows the applicability of our technique to large road networks as the size of the oracle is linear in n and that the constants involved are small, typically between 1.5 and 3. The large value of γ_H shown in Fig. 8 seems to have little effect on the size of the oracle.

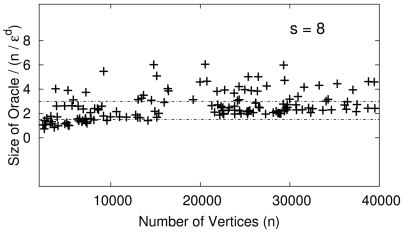

Fig. 10. Size of the oracle in terms of the number of Morton blocks, normalized by n/ε^d.

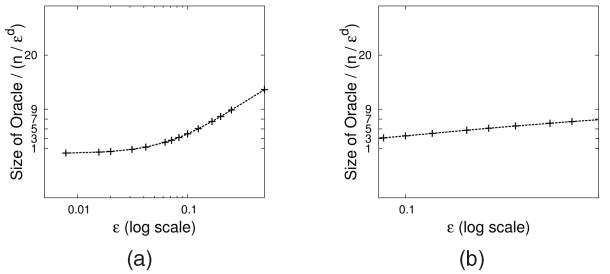

Fig. 11. The size of the oracle in terms of number of Morton blocks, normalized by n/ε^d applied to (a) Washington, DC, and (b) eastern seaboard data sets.

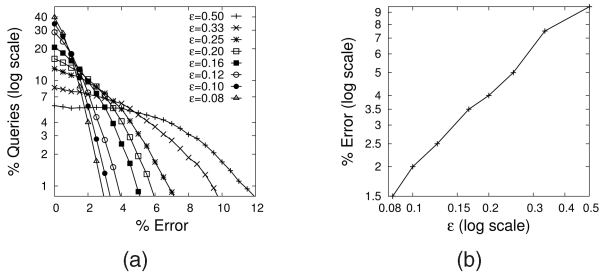

Fig. 13. (a) Percentages of queries along with their associated errors and (b) maximum error of 90 percent of the queries, for the oracles in Fig. 11b.

Next, we built $O(n)$-size oracles on the Washington, DC, data set containing 12,304 vertices for varying values of ε ranging between 0.50 and 0.0078, i.e., $s = 4$ to 256, which is shown in Fig. 11a. In Fig. 11b, we recorded the size of the oracles for the eastern seaboard data set containing 91,113 vertices for values of ε ranging between 0.50 and 0.0625, i.e., $s = 4$ to 32. Again, we recorded the size of the resulting distance oracle in terms of the number of Morton blocks, normalized by n/ε^d. We can see that the constants of proportionality are small values that vary between 1 and 10.

For each of the distance oracles computed in Figs. 11a and 11b, we made 100,000 ε-approximate distance queries between a vertex pair chosen at random. We computed the actual network distance between the pairs, and recorded the *maximum*, *average*, and the *standard deviation* of the error due to the approximation. The resultant error for the oracles is shown in Fig. 12. We can see that while the maximum error is within the prescribed bounds, the average and the standard deviation of the error are much lower than the actual value of ε. For example, for the distance oracles on the Washington, DC, data set shown in Fig. 12a, the average error, standard deviation, and the maximum error (in percentage) of the answers provided by the $\varepsilon = 0.1$ (10 percent error) oracle are 0.5, 2.7, and 9.0 percent, respectively. In the case of the eastern seaboard data set, shown in Fig. 12b for $\varepsilon = 0.1$, the corresponding average error, standard deviation, and the maximum error (in percentage) values are 0.9, 1.8, and 7.3 percent, respectively. These low average errors (i.e., less than 1 percent) mean that, in practice, the quality of the answers provided by this oracle is very close to the exact network distance.

Fig. 13a tabulates the resulting errors in the answers provided by the distance oracles as a percentage of the total number of queries for the 100,000 queries on the eastern seaboard data set of Fig. 11b. For example, Fig. 13a shows that for $\varepsilon = 0.25$ (i.e., 25 percent approximation), 12.9 percent of the queries are provided with more or less exact answers (i.e., less than 0.5 percent error). Moreover, 90 percent of the queries have errors of less than 5 percent as can be seen from Fig. 13b, which tabulates the maximum error of 90 percent of the queries. We note from our data (not shown here) that while the maximum possible error of the oracle is 25 percent, less than 1 percent of the answers to queries have errors of more than 10 percent.

We also computed the average time taken to retrieve an ε-approximate network distance for some of the oracles in Fig. 11a. Figs. 14a and 14b show that the average time taken to compute an ε-approximate network distance is on the order of tens of microseconds with maximums of 100 microseconds for the Washington, DC, data set in Fig. 14a, and 86 microseconds for the eastern seaboard data set in Fig. 14b. Note that this time can be further reduced by using a more efficient implementation of the B-tree structure.

In the remainder of this section, we discuss the use of distance oracles for performing region search and nearest neighbor queries on spatial networks. Our goal here is twofold. First, we show that an approximate distance oracle used in performing region and nearest neighbor queries yields answers that are generally of a good *quality*, which makes distance oracles applicable to general query processing on large spatial networks. Next, we examine the kind of speedup that can be obtained by adopting some of the strategies developed in Section 6. To measure quality of the results produced for the region and nearest neighbor queries with the distance oracles, we use the *precision* and *recall* scores. In particular, precision records how many of

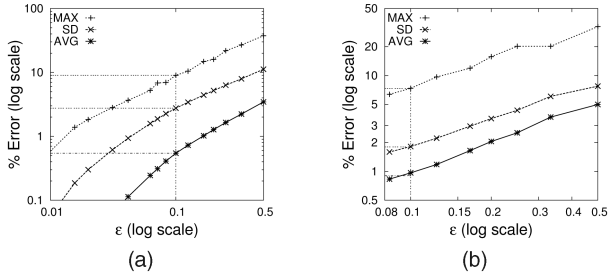

Fig. 12. The maximum, average, and the standard deviation errors for 100,000 network distance queries on the various oracles in Fig. 11 on (a) Washington, DC, and (b) eastern seaboard data sets.

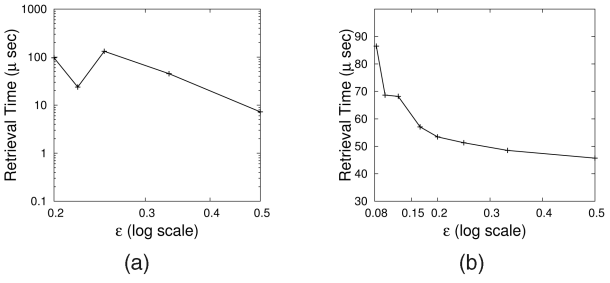

Fig. 14. Average time to retrieve an ε-approximate network distance for values of ε between (a) 0.1 and 0.5 for Washington, DC, and (b) 0.08 and 0.5 for the eastern seaboard data set.

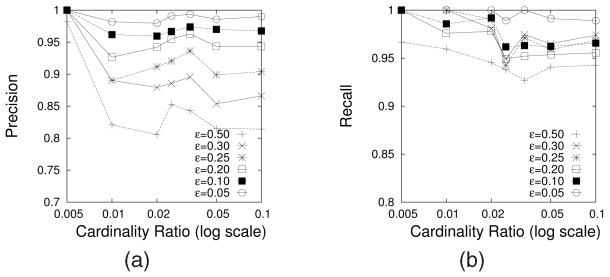

Fig. 15. Average (a) precision and (b) recall scores for region queries on spatial networks for varying cardinality ratio values.

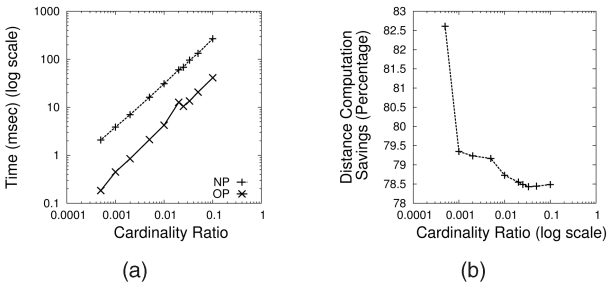

Fig. 16. Improvements in (a) time and (b) distance computations in using an optimized (OP) region search algorithm over an unoptimized (NP) variant for varying input size.

the objects in the result set produced with the aid of a distance oracle are indeed the correct answers, while recall measures what fraction of the correct answers appear in the result set produced by the oracle. All of our experiments use the eastern seaboard data set but vary the ε to better understand its effect on the quality of the answers produced by the distance oracle. For both the region and nearest neighbor queries, we make use of a data set of objects R. The size of R is denoted using the *cardinality ratio* [2], which is defined as the ratio of the number of objects in R to the number of vertices n in a spatial network. In our experiments, we vary the cardinality ratio of R between 0.0005 and 0.1 and ε between 0.5 and 0.01.

The first set of experiments evaluate the performance of the region search query in terms of the quality of the output as well as the speedup owing to the optimization strategies discussed in Section 6. In our experiments, we are interested in obtaining all objects in R that are within 40 km in terms of network distance of query point q. We vary both the cardinality ratio of R as well as use a variety of distance oracles with different ε values.

Figs. 15a and 15b examine the average precision and recall values, respectively, for region search queries on a spatial network as the cardinality ratio values of R vary between 0.0005 and 0.1. We repeated the experiments for 100 query points which were chosen at random and tabulated the average precision and recall values. Furthermore, we varied the value of ε between 0.5 (50 percent) and 0.01 (1 percent). We can see that for a 10 percent distance oracle, the precision and recall scores are at least 0.9 or more, which means that the resulting errors in using a distance oracle are quite low.

Next, we estimated the resulting speedup in using an optimized version (OP) of our region search algorithm when compared to its analogous unoptimized (NP) variant. We measure performance of our algorithms in terms of time taken to process the query as well as the savings in distance computations. Fig. 16a shows that the optimized algorithm is at least an order of magnitude faster than the unoptimized one. Moreover, the savings in distance computations given in Fig. 16b, which are shown as a percentage of the total distance computations performed by the unoptimized algorithm, indicates that the optimized algorithm only performs a fraction of the distance computations compared to the unoptimized algorithm. Therefore, it is easy to see that the optimized algorithm is much faster than the unoptimized algorithm.

The second set of experiments analyzes the performance of the k-nearest neighbor finding algorithm, which given a query point q finds the k-nearest neighbors to q from objects drawn from R. In the experiment described below, we examine the quality of the answers produced by the distance oracles as well as measure the resulting speedup in using some of the optimizations developed in Section 6.

Fig. 17a shows the average precision and recall scores for k-nearest neighbor queries for 100 query points chosen at random on a data set R residing on a spatial network with a cardinality factor of 0.01 for varying values of k between 1 and 200. Note that in the case of nearest neighbor search, the precision and recall values are always of the same value as we compare the first k neighbors obtained from using the distance oracle with the correct set of k neighbors. Next, Fig. 17b examines the average precision and recall values, respectively, as we varied the cardinality ratio values of R between 0.0005 and 0.1 while keeping k fixed at 10.

Fig. 17a records the precision and recall scores for varying values of ε ranging between 0.5 (50 percent) and 0.01 (1 percent). As we can see, the quality of the answers produced by the distance oracles is, on the average, quite high. A 10 percent distance oracle consistently produces an answer that is more than 0.95 on both the precision and recall measures, which is quite satisfactory for many application. In Fig. 17b, we varied the cardinality ratio of R between 0.005 and 0.1, while keeping k at 10. We can see that for a 10 percent distance oracle the precision and recall scores are at least 0.9 or more again indicating the applicability of distance oracles to this problem.

Finally, we show how large savings in time and distance computations can be achieved by using the optimization strategies discussed in Section 6. Fig. 18a shows the time taken to perform nearest neighbor searches for values of k

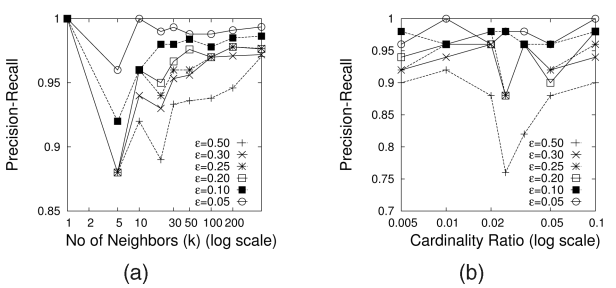

Fig. 17. Average precision and recall scores of nearest neighbor queries for varying values of (a) k with the cardinality ratio of R at 0.01 and (b) cardinality ratio values with $k = 10$.

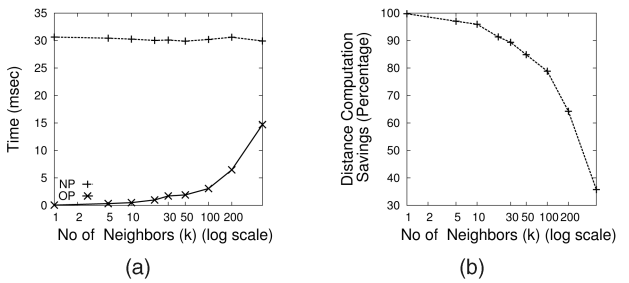

Fig. 18. Improvements in (a) time and (b) distance computations in using an optimized (OP) nearest neighbor algorithm over an unoptimized (NP) variant.

between 1 and 200 on a data set of points R of cardinality ratio of 0.01. We can see that the optimized variant (denoted by "OP") is almost an order of magnitude faster than the unoptimized version (denoted by "NP"). These savings, however, decrease as the value of k gets larger as by then k becomes a significant fraction of R. For even larger values of k, which is not shown here, the unoptimized variant of the algorithm becomes a feasible option. Next, Fig. 18b shows the savings in distance computations as a percentage of the number distance computations performed by the unoptimized nearest neighbor algorithm. We can see from the figure that the optimized algorithm only performs a fraction of the distance computations of the unoptimized algorithm.

8 Concluding Remarks

In this paper, we presented three approximate oracles for spatial networks that can answer approximate network distance queries. Our first oracle took unit space and could answer approximate network distance queries in $O(1)$ time. The drawback of this oracle was that the resulting error was large and dependent on the characteristics of the given spatial network. This led us to propose an oracle of size $O(\frac{n}{\varepsilon^d})$ that took advantage of the path coherence in spatial networks by decomposing the spatial network into sets of coherent source vertices and coherent destination vertices such that the network distances between them are represented by a single value that approximates them. Such an oracle could answer queries in $O(\log n)$ time using a B-tree. We also presented a theoretical analysis of a third variant that took $O(\frac{n \log n}{\varepsilon^d})$ space, but which could retrieve approximate network distances in $O(1)$ time with the aid of a hash table. Experiments performed on the $O(n)$-size oracle confirmed its linear storage requirements, while enabling us to answer approximate network distance queries in the order of tens of microseconds. Moreover, our experiments also demonstrated that the average and the standard deviation of the approximation error were low and, in fact, on the average, the error was much lower than the theoretical maximum ε value. For example, in the case of an oracle with $\varepsilon = 0.1$ (10 percent approximation) on the eastern seaboard data set, our average error was just around 0.9 percent with 90 percent of the queries making less than 2 percent errors which is negligible. This means that our oracle can be used as an efficient construct to provide *near* exact network distances in real time. Finally, we showed how to integrate our distance oracle with a relational database system and demonstrated strategies for making queries involving distance oracles efficient. Future work will explore a general strategy for query processing using a distance oracle, which can optimize complicated query scenarios involving data sets residing on a spatial network. Additional future work will address how to deal with updates due to events such as road closure, traffic congestion, etc.

Acknowledgments

The authors have been benefited greatly from discussions with Houman Alborzi. This work was supported in part by the US National Science Foundation under Grants EIA-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] J. Sankaranarayanan and H. Samet, "Distance Oracles for Spatial Networks," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, pp. 652-663, Apr. 2009. Q3

[2] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query Processing in Spatial Network Databases," *Proc. Conf. Very Large Data Bases (VLDB)*, pp. 802-813, Sept. 2003.

[3] H.-J. Cho and C.-W. Chung, "An Efficient and Scalable Approach to CNN Queries in a Road Network," *Proc. Conf. Very Large Data Bases (VLDB)*, pp. 865-876, Sept. 2005.

[4] N. Jing, Y.-W. Huang, and E.A. Rundensteiner, "Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation," *IEEE Trans. Knowledge and Data Eng.*, vol. 10, no. 3, pp. 409-432, May 1998.

[5] S. Jung and S. Pramanik, "An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps," *IEEE Trans. Knowledge and Data Eng.*, vol. 14, no. 5, pp. 1029-1046, Sept./Oct. 2002.

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

[7] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes, "In Transit to Constant Time Shortest-Path Queries in Road Networks," *Proc. Workshop Algorithm Eng. and Experiments (ALENEX)*, pp. 34-43, Jan. 2007.

[8] A.V. Goldberg and R.F. Werneck, "Computing Point-to-Point Shortest Paths from External Memory," *Proc. Workshop Algorithm Eng. and Experiments (ALENEX)*, Jan. 2005.

[9] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable Network Distance Browsing in Spatial Databases," *Proc. ACM SIGMOD*, pp. 43-54, June 2008.

[10] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Efficient Query Processing on Spatial Networks," *Proc. ACM Int'l Workshop Geographic Information Systems (GIS)*, pp. 200-209, Nov. 2005.

[11] D. Wagner and T. Willhalm, "Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs," *Proc. European Symp. Algorithms (ESA)*, pp. 776-787, Sept. 2003.

[12] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Enabling Query Processing on Spatial Networks," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, p. 163, Apr. 2006. Q4

[13] G.M. Hunter and K. Steiglitz, "Operations on Images Using Quad Trees," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 145-153, Apr. 1979.

[14] J. Sankaranarayanan, H. Samet, and H. Alborzi, "Path Oracles for Spatial Networks," *Proc. Conf. Very Large Data Bases (VLDB)*, vol. 2, no. 1, pp. 1210-1221, Aug. 2009.

[15] P.B. Callahan and S.R. Kosaraju, "Faster Algorithms for Some Geometric Graph Problems in Higher Dimensions," *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA)*, pp. 291-300, Jan. 1993.

[16] J. Gao and L. Zhang, "Well-Separated Pair Decomposition for the Unit-Disk Graph Metric and Its Applications," *Proc. Ann. ACM Symp. Theory of Computing (STOC)*, pp. 483-492, July 2003.

[17] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid, "Approximate Distance Oracles for Geometric Graphs," *Proc.*

ACM-SIAM Symp. Discrete Algorithms (SODA), pp. 828-837, Jan. 2002.
[18] G. Narasimhan and M. Smid, "Approximating the Stretch Factor of Euclidean Graphs," *SIAM J. Computing,* vol. 30, no. 3, pp. 978-989, 2000.
[19] C. Shahabi, M.R. Kolahdouzan, and M. Sharifzadeh, "A Road Network Embedding Technique for k-Nearest Neighbor Search in Moving Object Databases," *GeoInformatica,* vol. 7, no. 3, pp. 255-273, Sept. 2003.
[20] H.-P. Kriegel, P. Kröger, M. Renz, and T. Schmidt, "Hierarchical Graph Embedding for Efficient Query Processing in Very Large Traffic Networks," *Proc. Int'l Conf. Scientific and Statistical Database Management (SSDBM),* pp. 150-167, July 2008.
[21] N. Linial, E. London, and Y. Rabinovich, "The Geometry of Graphs and Some of Its Algorithmic Applications," *Combinatorica,* vol. 15, pp. 215-245, 1995.
[22] M. Thorup and U. Zwick, "Approximate Distance Oracles," *Proc. Ann. ACM Symp. Theory of Computing (STOC),* pp. 183-192, 2001.
[23] P.B. Callahan and S.R. Kosaraju, "A Decomposition of Multidimensional Point Sets with Applications to k-Nearest-Neighbors and n-Body Potential Fields," *J. ACM,* vol. 42, no. 1, pp. 67-90, Jan. 1995.
[24] T.M. Chan, "Well-Separated Pair Decomposition in Linear Time?" *Information Processing Letters,* vol. 107, no. 5, pp. 138-141, Aug. 2008.
[25] J.A. Orenstein, "Multidimensional Tries Used for Associative Searching," *Information Processing Letters,* vol. 14, no. 4, pp. 150-157, June 1982.
[26] J. Fischer and S. Har-Peled, "Dynamic Well-Separated Pair Decomposition Made Easy," *Proc. Canadian Conf. Computational Geometry (CCCG),* pp. 235-238, Aug. 2005.
[27] G.M. Morton, "A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing," technical report, IBM Ltd., 1966.
[28] I. Gargantini, "An Effective Way to Represent Quadtrees," *Comm. ACM,* vol. 25, no. 12, pp. 905-910, Dec. 1982.
[29] G.R. Hjaltason and H. Samet, "Ranking in Spatial Databases," *Proc. Int'l Symp. Advances in Spatial Databases (SSD),* pp. 83-95, Aug. 1995.
[30] G.R. Hjaltason and H. Samet, "Incremental Distance Join Algorithms for Spatial Databases," *Proc. ACM SIGMOD,* pp. 237-248, June 1998.
[31] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Distance Join Queries on Spatial Networks," *Proc. ACM Int'l Workshop Geographic Information Systems (GIS),* pp. 211-218, Nov. 2006.
[32] H. Samet, H. Alborzi, F. Brabec, C. Esperança, G.R. Hjaltason, F. Morgan, and E. Tanin, "Use of the SAND Spatial Browser for Digital Government Applications," *Comm. ACM,* vol. 46, no. 1, pp. 63-66, Jan. 2003.
[33] J.L. Bentley, "Decomposable Searching Problems," *Information Processing Letters,* vol. 8, no. 5, pp. 244-251, June 1979.
[34] S. Chaudhuri, "An Overview of Query Optimization in Relational Systems," *Proc. Symp. Principles of Database Systems (PODS),* pp. 34-43, June 1998.
[35] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD,* pp. 47-57, June 1984.
[36] H. Shin, B. Moon, and S. Lee, "Adaptive Multi-Stage Distance Join Processing," *Proc. ACM SIGMOD,* pp. 343-354, May 2000.
[37] M.J. Carey and D. Kossmann, "On Saying 'Enough Already!' in SQL," *Proc. ACM SIGMOD,* pp. 219-230, May 1997.
[38] S. Har-Peled, "Geometric Approximation Algorithms," Collection of Lecture Notes, Nov. 2008.
[39] U.S. Census Bureau, "TIGER/Line Files, Census 2000," http://www.census.gov/geo/www/tiger/tiger2k/tiger2000.html, Oct. 2001.
[40] USGS, "Major Roads of the United States," http://nationalatlas.gov/atlasftp.html, Nov. 1999.

Jagan Sankaranarayanan received the PhD degree in computer science from the University of Maryland in 2008 under the guidance of Prof. Hanan Samet. He is an assistant research scientist at the Center for Automation Research (CfAR), University of Maryland. He is the recipient of the Best Paper Awards at the SIGMOD 2008 and ACM SIGSPATIAL GIS 2008 conferences and the Best Journal Paper of 2007 Award by the *Computers & Graphics* Journal. He is a member of the IEEE.

Hanan Samet received the BS degree in engineering from the University of California, Los Angeles, and the MS degree in operations research and the MS and PhD degrees in computer science from Stanford University, California. His PhD dissertation was the first work in the field of compiler translation validation. He is a fellow of the IEEE, the ACM, and the International Association for Pattern Recognition (IAPR), and was also elected to the ACM Council in 1989-1991 where he served as the capital region representative. He is the recipient of the 2009 UCGIS Research Award, the 2010 University of Maryland College of Computer, Mathematical and Physical Sciences Board of Visitors Distinguished Faculty Award, and the Science Foundation of Ireland (SFI) Walton Visitor Award at the Centre for Geocomputation at the National University of Ireland at Maynooth (NUIM). In 1975, he joined the Computer Science Department at the University of Maryland, College Park, where he is now a professor. He is a member of the Computer Vision Laboratory of the Center for Automation Research and also has an appointment in the University of Maryland Institute for Advanced Computer Studies. At the Computer Vision Laboratory, he leads a number of research projects on the use of hierarchical data structures for geographic information systems. His research group has developed the QUILT system which is a GIS based on hierarchical spatial data structures such as quadtrees and octrees, the SAND system which integrates spatial and nonspatial data, the SAND Browser (http://www.cs.umd.edu/brabec/sandjava) which enables browsing through a spatial database using a graphical user interface, the VASCO spatial indexing applet (found at http://www.cs.umd.edu/hjs/quadtree/index.html), a symbolic image database system, and the STEWARD system for spatiotextual retrieval of documents on the web as well as the NewsStand and TwitterStand systems for news and Twitter tweets, respectively. He is the founding chair of the ACM Special Interest Group on Spatial Information (SIGSPATIAL). He has served as the cogeneral chair of the 15th ACM International Conference on Advances in Geographic Information Systems (ACMGIS'07) and the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACMGIS'08). His research interests include data structures, computer graphics, geographic information systems, computer vision, robotics, and database management systems, and he is the author of more than 300 publications on these topics. He is the author of the recent book titled *Foundations of Multidimensional and Metric Data Structures* (http://www.cs.umd.edu/hjs/multidimensional-book-flyer.pdf) published by Morgan-Kaufmann, an imprint of Elsevier, in 2006, an award winner in the 2006 best book in Computer and Information Science competition of the Professional and Scholarly Publishers (PSP) Group of the American Publishers Association (AAP), and of the first two books on spatial data structures titled *Design and Analysis of Spatial Data Structures*, and *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, both published by Addison-Wesley in 1990. He is an area editor of the *Graphical Models*, and on the editorial board of the *Image Understanding*, the *Journal of Visual Languages*, and the *GeoInformatica*. He received best paper awards in the 2008 SIGMOD Conference, the 2008 SIGSPATIAL ACMGIS'08 Conference, and the 2007 *Computers & Graphics* Journal. His paper at the 2009 IEEE International Conference on Data Engineering (ICDE) was selected as one of the best papers for publication in the *IEEE Transactions on Knowledge and Data Engineering*.