

# Path Oracles for Spatial Networks\*

Jagan Sankaranarayanan

Hanan Samet

Houman Alborzi†

Center for Automation Research

Institute for Advanced Computer Studies

Department of Computer Science

University of Maryland, College Park, MD 20742

{jagan,hjs,houman}@cs.umd.edu

## ABSTRACT

The advent of location-based services has led to an increased demand for performing operations on spatial networks in real time. The challenge lies in being able to cast operations on spatial networks in terms of relational operators so that they can be performed in the context of a database. A linear-sized construct termed a path oracle is introduced that compactly encodes the  $n^2$  shortest paths between every pair of vertices in a spatial network having  $n$  vertices thereby reducing each of the paths to a single tuple in a relational database and enables finding shortest paths by repeated application of a single SQL SELECT operator. The construction of the path oracle is based on the observed coherence between the spatial positions of both source and destination vertices and the shortest paths between them which facilitates the aggregation of source and destination vertices into groups that share common vertices or edges on the shortest paths between them. With the aid of the Well-Separated Pair (WSP) technique, which has been applied to spatial networks using the network distance measure, a path oracle is proposed that takes  $O(s^d n)$  space, where  $s$  is empirically estimated to be around 12 for road networks, but that can retrieve an intermediate link in a shortest path in  $O(\log n)$  time using a B-tree. An additional construct termed the path-distance oracle of size  $O(n \cdot \max(s^d, \frac{1}{\epsilon}^d))$  (empirically  $(n \cdot \max(12^2, \frac{2.5^2}{\epsilon}))$ ) is proposed that can retrieve an intermediate vertex as well as an  $\epsilon$ -approximation of the network distances in  $O(\log n)$  time using a B-tree. Experimental results indicate that the proposed oracles are linear in  $n$  which means that they are scalable and can enable complicated query processing scenarios on massive spatial network datasets.

†Now at Google Inc., 4720 Forbes Ave, Pittsburgh, PA 15213

\*This work was supported in part by the National Science Foundation under Grants IIS-08-12377, CCF-08-30618, and IIS-07-13501, as well as NVIDIA Corporation, Microsoft Research, 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.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '09, August 24-28, 2009, Lyon, France

Copyright 2009 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

## 1. INTRODUCTION

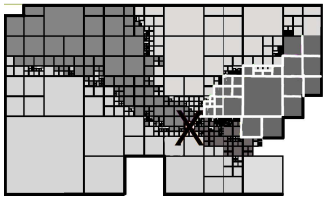
The trend towards smaller and smaller computing platforms, as well as the concomitant increase in their power, has led to an increasing interest in their use to facilitate the mobility of their owners. This has resulted in the development of a new component of the software industry known collectively as location-based services. Such services can be used to continuously monitor ever-changing user positions as well as their destinations, and provide up-to-the-minute (second) information about their environment, as well as paths to their destinations. This is especially useful in an emergency response environment where destination, or, even more drastically, the destinations may change given the ever-changing conditions on the ground. The challenge in providing this information lies in the fact that these paths are not simple to compute as obviously we want to take the shortest paths, and the total distance is computed along a network instead of “as the crow flies.”

In fact, these changes may happen so fast that we may not even have enough time to compute the exact answer and thus must resort to estimates (i.e., approximate answers) with some guarantee of optimality (i.e., within some tolerance  $\epsilon$ ). Moreover, in these situations, there is a need to interact with a multitude of information sources such as road networks, building diagrams, service areas, etc. Increasingly, these information sources are stored in databases that must be accessible using a common interface (i.e., language), most often some variant of SQL.

In this paper we address how to find shortest paths in such a dynamically changing environment which is only the first step in being able to perform a wide variety of operations on spatial networks such as region searches [9, 10], nearest neighbor finding [2, 3, 7, 9–11] and distance joins [9]. The problem is that finding shortest paths and distances invariably involve a search process (e.g., via use of a shortest path algorithm [5, 6, 17]), which takes quite a bit of time, and is not a satisfactory solution in terms of data that is organized using a relational database and is accessed via SELECT operations. Our solution is based on precomputing the shortest paths between all possible sources and destinations, and encoding them in a compact form which lends itself to being stored and retrieved in/from a conventional relational database.

Our results are a natural extension of our earlier work that led to the formulation of the concept of the shortest-path map, whose representation by the shortest-path quadtree, enabled us to reduce the necessary storage for the set of all of the shortest paths from  $O(n^3)$  to  $O(n^{1.5})$  [11]. This was achieved by taking advantage of the fact that the shortest paths from a particular vertex  $v$  to all of the remaining vertices can be partitioned into  $i$  subsets depending on the identity of the first edge (there are  $i$  such edges) on the shortest paths to them from  $v$  (e.g., Figure 1). We characterize this as taking advan-

tage of the *path coherence* of the destination vertices. The shortest paths are derived by an iterative process that repeatedly finds the next edge to the destination and thus lends itself to a database solution (i.e., the repeated retrieval of information) rather than a graph search. In particular, the process is several times faster than conventional graph-based solutions [9], and also casts a nice balance between offline and online computation of the shortest paths.



**Figure 1: Space partition induced by aggregating vertices sharing the same first link in the shortest path from a given source vertex in the road network of Silver Spring, MD**

In this paper we expand on this work to reduce the space requirements from  $O(n^{1.5})$  to  $O(n)$  by also capturing the path coherence of the source vertices, instead of just the path coherence of the destination vertices. Our motivation can be seen by observing that someone who is driving along the shortest route from the Northeast of the US to the Northwest Coast of the US will invariably end up using Highway I-80 West. This path coherence is most evident when the sources and destinations are sufficiently far apart in the road network. As an example, of the amount of space that can be potentially saved, suppose that we have one million source vertices and one million destination vertices. If for each such pair of vertices, we store a vertex on I-80 West, then we would incur a storage cost of  $10^{12}$  pieces of information, while when using our path coherence techniques, we can achieve this in  $O(1)$  space by virtue of storing just one item of information for the set of vertices in the Northeast Coast, one intermediate vertex, and one item of information for the set of vertices on the Northwest Coast. Figure 2 is an example of such a configuration of a set of sources and destinations that share vertices in the shortest paths resulting in storing partial path information of 30,000 shortest paths using  $O(1)$  storage. In the rest of the paper we show that for a spatial network of size  $n$ , there are  $O(n)$  such groups of  $O(1)$  size that capture all of the  $O(n^2)$  shortest paths of the network. This partitioning of the vertices into appropriate subsets of source and destination vertices is achieved by appealing to the Well-Separated Pair (WSP) decomposition [1], and conditions under which it is satisfied for a spatial network are specified here. Our presentation uses the term *oracle* to describe a data structure, or representation, or a relation in a database that captures all the shortest paths. We describe three such oracles.

First, given a source and a destination vertex in a spatial network  $G$ , and letting  $s$  be a factor depending on  $G$ , we present a path oracle of size  $O(s^2n)$  that enables the retrieval of an intermediate vertex on the shortest path between them in  $O(\log n)$  time. This is done via use of a B-tree.

Next, we make use of an earlier result of ours [13] that for a source and a destination vertex in a spatial network  $G$ , an  $\epsilon$ -approximate distance oracle that requires  $O(\frac{n}{\epsilon^2})$  space can be constructed that enables the retrieval of a network distance between the two vertices that lies within  $\epsilon$  of the true network distance between them in  $O(\log n)$  time.

Finally, the main result of this paper is the combination of the above two oracles to define a path-distance oracle that can provide both an intermediate vertex, as well as an  $\epsilon$ -approximate distance in  $O(\log n)$  time using  $O(n \cdot \max(s^2, \frac{1}{\epsilon^2}))$  space via the use of a B-tree.

Experimental results show that for road networks of different

sizes  $n$  (for which  $s$  turned out to lie in the range 8–15), the size of the path-distance oracle is about  $12^2n$  and provides answers in 10–35  $\mu$ seconds for  $\epsilon = 20\%$ . When  $\epsilon$  was allowed to vary between 1 and 50%, and  $n$  was kept constant at 91,113, the size of the path-distance oracle was  $n \cdot \max(12^2, \frac{2.5^2}{\epsilon^2})$  and provided answers in 30–35  $\mu$ seconds.

The significance of our work lies in demonstrating that shortest paths can be calculated by repeated SQL SELECT operators. Moreover, by having the ability to refine the paths, we can also refine the network distance value, which is an improvement on our earlier result [13] that only provided one predicted approximate distance value. This ability to refine the distance enables us to obtain exact values and thereby enables obtaining exact, rather than approximate, responses to a number of spatial queries via SQL relational operators. Our path-distance oracle on a spatial network is related to oracles on other kinds of graphs (e.g., general graphs [14]) as well as to those that view spatial networks as a general metric space and apply an embedding method [8]. The interested reader is referred to [13] for a more a detailed discussion of these related methods.

The oracles proposed in this paper, and our earlier efforts with shortest-path quadtrees [11] and distance oracles [13], depart from the existing literature in three fundamental ways. First of all, methods [5, 6, 17] that speed up shortest-path finding do so by limiting the search space on a general graph. Precomputation of the shortest-paths obviates the need to search, and instead replaces it with a *retrieval* process which is much faster. Next, traditional query processing on spatial network is made up of graph-based approaches [2, 3, 9]. Query processing using the oracles and the shortest-path quadtree instead perform operations that are purely spatial in nature and hence, more suitable for incorporation into a database. Finally, once our representation is computed for a spatial network it can be used in conjunction with any dataset of entities (e.g., restaurants, coffee shops) that lie on the spatial network as long as the network itself remains unchanged which is different from other precomputational efforts (e.g., [7]) that require a renewed precomputational effort every time a dataset is changed, or a new dataset is introduced.

The rest of this paper is organized as follows. Section 2 provides some definitions. Section 3 expands on the notion of path-coherent pairs, the construction of the various oracles, and also shows how a number of spatial queries can be implemented via SQL relational operators. Section 4 contains an analysis of the storage requirements of the decomposition of a spatial network into a set of path-coherent pairs by appealing to the equivalence between the path coherent pair decomposition of a spatial network and the WSP decomposition of a point set. The results of experiments are discussed in Section 5, while conclusions are drawn in Section 6.

## 2. PRELIMINARIES

Spatial networks are general graphs whose vertices and edges are augmented with spatial information. Let  $S$  denote a  $d$ -dimensional *embedding space* (i.e., a reference coordinate system), which is two-dimensional for road networks. 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 edge  $e \in E$ ,  $w(e) \geq 0$  denotes the distance along  $e$ . In addition, for every  $v \in V$ ,  $p(v)$  denotes the spatial position of  $v$  with respect to  $S$ .

A *path*  $\pi$  of length  $k$  is a sequence of vertices  $(\pi_1, \dots, \pi_{k+1})$  such that  $(\pi_i, \pi_{i+1}) \in E$  for  $1 \leq i \leq k$ . We refer to  $\pi_1$  as the *source* vertex of  $\pi$  and refer to  $\pi_{k+1}$  as the *destination* vertex of  $\pi$ . Let

$\pi(u, v)$  denote a path (not necessarily the shortest path) with  $u$  as its source vertex and  $v$  as its destination vertex. The sequence of edges that make up the path  $\pi$  is denoted by the sequence  $\varphi(\pi)$ , where  $\varphi_i(\pi) = (\pi_i, \pi_{i+1})$ . Furthermore, the *weight*  $w(\pi)$  of a path  $\pi$  of length  $k$  is  $w(\pi) = \sum_{i=1}^k w(\varphi_i(\pi))$ . Two paths  $\pi_1(v, t)$  and  $\pi_2(t, u)$  can be composed to form another path  $\pi$  denoted by  $\pi_1 \rightsquigarrow \pi_2$ . A *subpath* of a path  $\pi$  is a subsequence of  $\pi$ . The set of vertices that make up the shortest path between a pair of vertices  $u, v \in V$  is denoted by  $\pi_G(u, v)$ . Also, any subpath  $\pi(r, t)$  of  $\pi_G(u, v)$  is also the shortest path between  $r$  and  $t$ . If there are multiple shortest paths of the same length between vertex pairs, extra care must be taken to ensure that the above property holds. In such cases, the first path in the lexicographic ordering on the set of possible shortest paths is chosen, such that the ordering is defined on triples  $(w(\pi), k, \text{reverse}(\pi))$ , where the reverse operator takes an ordered set as input and reverses its ordering. Furthermore, two sequences  $\pi_1$  and  $\pi_2$  are *disjoint*, if and only if  $\pi_1 \cap \pi_2 = \emptyset$ . Notice that if two paths  $\pi_1(v, t)$  and  $\pi_2(t, u)$  are disjoint from a path  $\pi^*$  then the path  $\pi_1 \rightsquigarrow \pi_2$  is also disjoint from  $\pi^*$ . Moreover, if  $\pi_1$  is disjoint from  $\pi_2$  then any subpath of  $\pi_1$  is also disjoint from  $\pi_2$ .

For vertices  $u, v \in V$ , we define  $d_G(u, v) = w(\pi_G(u, v))$  to be the *shortest network distance* from  $u$  to  $v$  with respect to  $G(V, E)$ . We define the spatial distance  $d_S(u, v)$  (i.e., “as the crow flies”) between vertices  $u, v \in V$  in a spatial network as a function on  $p(u)$  and  $p(v)$ . We also define  $l_u(v)$  to be the *next vertex* visited (after  $u$ ) on the shortest path from  $u$  to  $v$ . Note that the first edge on the shortest path from  $u$  to  $v$  is  $(u, l_u(v))$ .

Let  $T$  be the root block of a PR-quadtrees  $H$  (e.g., [10]) on the spatial positions of the vertices  $V$ . A PR-quadtrees is a hierarchical decomposition of the embedding space  $S$  such that every block is decomposed into  $2^d$  children blocks until a *leaf* block is obtained, which corresponds to a region in space containing a single vertex in  $V$ . A *non-leaf* block in  $H$ , on the other hand, represents a region in the embedding space containing a subset of vertices (more than one) in  $V$ . We can devise an addressing scheme for a block  $b$  by a bit-encoding obtained by concatenating the bits corresponding to the path taken to reach  $b$  from the root block  $T$ . This representation, also known as the MORTON BLOCKS [10], is used by us in order to uniquely identify blocks in a PR-quadtrees.

### 3. PATH-COHERENT PAIRS



**Figure 2: The 30,000 shortest paths between two subsets  $A, B$  of vertices in a road network of Silver Spring, MD pass through a single common vertex**

We observe that vertices in a spatial network that are spatially close to one another share a number of common properties. In particular, often two vertices  $u, v$  that are spatially close to each other share large common segments of their shortest paths to two other vertices  $t, w$  that are also spatially close to each other, but far from  $u, v$ . We use the term path coherence to describe the coherence between the shortest paths from nearby sources to nearby destinations. Figure 2 shows a configuration of source  $A$  and destination  $B$  vertices such that every shortest path from a vertex  $u \in A$  to a vertex

$t \in B$  passes through a particular set of vertices. Two sets of vertices  $A, B$  are said to form a *Path-Coherent Pair (PCP)* if and only if all the shortest paths from source vertices in  $A$  to destination vertices in  $B$  have at least one vertex or one edge in common as shown in Figure 2. Our path oracle relies on the decomposing a given spatial network  $G$  into path-coherent pairs such that they capture all the  $n^2$  shortest paths in  $G$ .

#### 3.1 Definition of Path-Coherent Pairs

The arrangement of vertices and shortest paths in Figure 2 describes a *dumbbell*-like structure. A *Path-Coherent Pair (PCP)*  $(A, B, \Psi, \lambda)$  in a spatial network  $G(V, E)$  consists of a set of source vertices  $A \subset V$ , a set of destination vertices  $B \subset V$ , a set  $\Psi$  which is a vertex or an edge such that all the shortest paths from source vertices in  $A$  to destination vertices in  $B$  contain  $\Psi$ ,  $\lambda \in \mathbb{R}^+$  approximates the network distances of all the shortest paths from  $A$  to  $B$ . We refer to  $A$  and  $B$  as the *heads* of the PCP. A PCP  $(A, B, \Psi, \lambda)$  belongs to one of the two possible configurations given below:

- $(A, B, \Psi = \{u\}, \lambda)$ , where  $A \cap B = \emptyset$ ,  $u \in V$ ,  $u \notin A$  and  $u \notin B$
- $(A, B, \Psi = \{<u, v>\}, \lambda)$ , where  $A \cap B = \emptyset$ ,  $<u, v> \in E$  and  $u \notin B$  and  $v \notin A$

Note that any edge  $<u, v> \in E$  in  $G$  is a PCP of the form  $(u, v, <u, v>, \lambda)$ .

#### 3.2 PCP Decomposition

Given a spatial network  $G$ , we now perform a *decomposition*  $G \otimes G$  of  $G$ , into a set of PCPs, such that the resulting decomposition has the following properties given below:

1.  $G \otimes G = \bigcup_{i=1}^l (A_i, B_i, \Psi_i, \lambda_i)$ , where  $(A_i, B_i, \Psi_i, \lambda_i)$  is a PCP  $\forall i = 1, \dots, l$ , such that  $A = \bigcup_{i=1}^l A_i$  and  $B = \bigcup_{i=1}^l B_i$ .
2.  $(A_i \cap B_i) = \emptyset, \forall i = 1, \dots, l$ .
3. For any two PCPs  $(A_i, B_i, \Psi_i, \lambda_i), (A_j, B_j, \Psi_j, \lambda_j), 1 \leq i < j \leq l$  in  $G \otimes G$ , the resulting decomposition has the property that  $(A_i \cap A_j) \times (B_i \cap B_j) = \emptyset$ . In other words, for any pair of vertices  $(u, v)$ , there exists a unique PCP  $(A_i, B_i, \Psi_i, \lambda_i)$  in  $G \otimes G$ , s.t.  $u \in A_i, v \in B_i$ .

The first property ensures that the decomposition of  $G$  results in a set containing  $l$  PCPs, where the  $i^{\text{th}}$  PCP in the decomposition is denoted by  $(A_i, B_i, \Psi_i, \lambda_i)$ . The second property ensures that the heads  $A_i$  and  $B_i$  of a PCP are disjoint. The third property ensures that any pair of vertices  $(u, v)$  in  $G$  is contained in exactly one of the PCPs in the decomposition. This also means that the PCP decomposition contains all the  $n^2$  shortest paths in  $G$ . This leads us to the definition of path, distance and path-distance oracles.

**DEFINITION 1.** A *path oracle* of a spatial network  $G$  is a PCP decomposition of  $G$ .

**DEFINITION 2.** A *path-distance oracle* of a spatial network  $G$  is a PCP decomposition of  $G$  of the form  $(A_i, B_i, \Psi_i, \lambda_i)$  with the additional property that

$$(1 - \varepsilon) \cdot \lambda_i \leq d_G(u, v) \leq (1 + \varepsilon) \cdot \lambda_i,$$

where  $u \in A_i$  and  $v \in B_i$ . In other words, a path-distance oracle stores a common intermediate vertex or edge  $\Psi_i$  in the shortest paths from  $A_i$  to  $B_i$  as well as an  $\varepsilon$ -approximation  $\lambda_i$  of the network distances from  $A_i$  to  $B_i$ .

**DEFINITION 3.** A distance oracle [13] is a decomposition of a spatial network  $G$  into triples of the form  $(A_i, B_i, \lambda_i)$  such that  $A_i \subset V$ ,  $B_i \subset V$ ,  $A_i \cap B_i = \emptyset$ ,  $\lambda_i$  is an  $\varepsilon$ -approximation of the network distances from  $A_i$  to  $B_i$  and given any pair of source and destination vertices, it is contained exactly in one of the triples in the decomposition. In other words, the above decomposition is similar to the path-distance oracles with the difference that, now, all the shortest paths from  $A_i$  to  $B_i$  no longer have to pass through a single common vertex or edge.

### 3.3 Oracle Construction

We now describe an algorithm for constructing path, distance, and path-distance oracles. Algorithm 1 decomposes a spatial network  $G(V, E)$  into a set of PCPs. The algorithm takes  $G$ , the root block  $T$  of a PR-quadtrees  $H$  (e.g., [10]) on the spatial positions of  $V$ , the type of oracle ORACLETYPE, and the quality of approximation  $\varepsilon$  as inputs. The possible values for ORACLETYPE are PATH, DISTANCE, or PATH-DISTANCE. The value of  $\varepsilon$  needs to be specified for distance and path-distance oracles, but is assumed to be  $\infty$  for path oracles.

Let  $Q$  be a list of block-pairs, which is initialized with the pair  $(T, T)$  in line 1. At each stage of the algorithm, a block-pair  $(A, B)$  is retrieved from  $Q$  and examined in line 3. Note that  $A$  and  $B$  correspond to blocks in the PR-quadtrees on  $V$ . If  $A$  and  $B$  refer to the same block (line 4) and  $A$  (and  $B$ ) is a non-leaf block, then we invoke DECOMPOSEANDINSERT operator with  $Q$ ,  $A$ , and  $B$ . This operator operator takes two blocks  $A$  and  $B$  and a list  $Q$  as inputs. It first breaks up  $A$  and  $B$  into their  $2^d$  children blocks, forms all possible pairs obtained by taking non-empty children blocks of  $A$  and  $B$  which are then inserted into  $Q$ . Note that if the DECOMPOSEANDINSERT operator is invoked on  $A$  and  $B$  such that one of them is a LEAF block, it can be still broken up into  $2^d$  children blocks, but all but one of the children blocks will be empty.

If  $A$  and  $B$  do not refer to the same block in the PR-quadtrees, the algorithm invokes the FINDPATHCOHERENTPAIRS operator on  $A$  and  $B$  in line 9 that returns, if it exists, a vertex or an edge  $\Psi$  that is common to the shortest paths from source vertices in  $A$  to destination vertices in  $B$ , as well as an approximate network distance  $\lambda$ , and the maximum error  $\varepsilon_H$  in approximating the network distances by  $\lambda$ . Optionally, we can also obtain the minimum error in approximating the network distances between vertices in  $A$  and  $B$  by  $\lambda$ . Both the values of  $\varepsilon_L$  and  $\varepsilon_L$  can also be stored along with the PCPs in the output. Given  $(A, B)$ , efficient methods for computing  $\lambda$  and  $\varepsilon_H$  are described in [13]. One possible way of finding if a common vertex or edge  $\Psi$  exists is to compute the shortest path from every pair of source vertices in  $A$  to destination vertices in  $B$  using Dijkstra's algorithm and determine if there is a vertex or edge in common between all the shortest paths. If there is more than one vertex in common to all the shortest paths then one of the vertices is chosen at random; preferably one not belonging to either  $A$  or  $B$ . The case that an edge is common to all the shortest paths occurs only if all the shortest paths pass through an edge  $\langle u, v \rangle$ , such that  $u \in A$  or  $\{u\} = A$ , and  $v \in B$  or  $\{v\} = B$ . An example of such a configuration arises if a bridge or a tunnel represented by an edge connects vertices belonging to two cities denoted by  $A$  and  $B$ . The algorithm SPATH (Algorithm 2) that we propose for computing the shortest paths will not work when  $\Psi$  is a vertex and if  $\Psi \in A$ , or  $\Psi \in B$ , or  $\{\Psi\} = A$ , or  $\{\Psi\} = B$ . Similarly, if  $\Psi = \langle u, v \rangle$  is an edge, the algorithm will not work  $u \in B$  or  $v \in A$ . Algorithm 1 handles all these cases appropriately.

If ORACLETYPE is DISTANCE (line 10), we examine if the quality of the approximation  $\varepsilon_H$  is less than or equal to the desired approximation  $\varepsilon$ . If so, we add the pair  $(A, B, \lambda)$  to the result set

using the REPORTPCP procedure. If not, we split  $A$  and  $B$  into block-pairs formed by children blocks of  $A$  and  $B$ , which are then inserted into  $Q$ .

If ORACLETYPE is PATH, we check if  $\Psi$  is empty in which case  $(A, B)$  is not a valid PCP and the block-pair  $(A, B)$  has to be further decomposed. If  $\Psi$  is a vertex that does not belong to either  $A$  or  $B$ , we report it as a PCP, else we further decompose it. If  $\Psi = (u, v)$  is an edge, we ensure that  $u \notin B$  and  $v \notin A$ , in which case it is reported as a PCP. We point out that all other possible cases i.e., when  $u \in B$ , or/and  $v \in A$  are all undesirable. Finally, if ORACLETYPE is PATH-DISTANCE and  $(A, B, \Psi, \lambda)$  satisfies both the path and distance constraints, it is added to the result set.

The algorithm terminates when  $Q$  is empty at which point the decomposition of  $G$  into PCPs is complete. Note that our algorithm breaks up both  $A$  and  $B$  in a symmetric fashion which means that the heads of the PCP in the output are at the same level in  $H$ , and consequently, of the same size.

#### Algorithm 1

**Procedure** PCPDECOMPOSE[ $G, T, \text{ORACLETYPE}, \varepsilon = \infty$ ]

**Input:**  $G(V, E) \leftarrow$  Input spatial network

**Input:**  $T \leftarrow$  Root node of a PR-quadtrees  $H$  on  $V$

**Input:** ORACLETYPE  $\leftarrow$  PATH, DISTANCE, or PATH-DISTANCE

**Input:**  $\varepsilon \leftarrow$  Required approximation;  $\infty$  if not specified

**Output:** Set of PCPs  $(A, B, \Psi, \lambda, [\varepsilon_L, \varepsilon_H])$ ;  $[\varepsilon_L, \varepsilon_H]$  is optional  
 (\*  $Q \leftarrow$  list of block-pairs \*)

```

1.  INSERT( $Q, (T, T)$ )
2.  while (ISNOTEMPTY( $Q$ )) do
3.      ( $A, B$ )  $\leftarrow$  POP( $Q$ ) (* Remove head element *)
4.      if  $A = B$  then
5.          if ISNONLEAF( $A$ ) then
6.              DECOMPOSEANDINSERT( $Q, A, B$ )
7.          end-if
8.      else
9.          ( $\Psi, \lambda, \varepsilon_H, [\varepsilon_L]$ )  $\leftarrow$  FINDPATHCOHERENTPAIRS( $A, B$ )
10.         if ORACLETYPE = DISTANCE or PATHDISTANCE then
11.             if  $\varepsilon_H > \varepsilon$  then
12.                 DECOMPOSEANDINSERT( $Q, A, B$ )
13.             else if ORACLETYPE = DISTANCE then
14.                 REPORTPCP( $A, B, \Psi, \lambda, [\varepsilon_L, \varepsilon_H]$ )
15.             end-if
16.         end-if
17.         if ORACLETYPE = PATH or PATH-DISTANCE then
18.             if  $\Psi = \emptyset$  then
19.                 DECOMPOSEANDINSERT( $Q, A, B$ )
20.             else if TYPE( $\Psi$ ) = VERTEX then
21.                 if  $\Psi \in A$  or  $\{\Psi\} = A$  or  $\Psi \in B$  or  $\{\Psi\} = B$  then
22.                     DECOMPOSEANDINSERT( $Q, A, B$ )
23.                 else
24.                     REPORTPCP( $A, B, \Psi, \lambda, [\varepsilon_L, \varepsilon_H]$ )
25.                 end-if
26.             else if TYPE( $\Psi = \langle u, v \rangle$ ) = EDGE then
27.                 if  $u \notin B$  and  $v \notin A$  then
28.                     REPORTPCP( $A, B, \Psi = \langle u, v \rangle, \lambda, [\varepsilon_L, \varepsilon_H]$ )
29.                 else
30.                     DECOMPOSEANDINSERT( $Q, A, B$ )
31.                 end-if
32.             end-if
33.         end-if
34.     end-if
35. end-while
    
```

It is not difficult to see that Algorithm 1 decomposes  $G$  into a set of PCPs that satisfies properties 1–3. Hence, given a ver-

text pair  $a, b$ , we are guaranteed that there exists exactly one PCP  $(A, B, \Psi, \lambda)$  in the output of Algorithm 1 such that  $A$  contains  $a$ ,  $B$  contains  $b$ ,  $\Psi$  is an intermediate vertex or edge in the shortest path from  $a$  to  $b$ , and  $\lambda$  approximates the network distance between  $a$  and  $b$ .

### 3.4 The Oracle as a Database Relation

The REPORTPCP routine in Algorithm 1 stores the PCP decomposition as a relation  $\mathbb{O}$  in a database system with the following schema:  $\mathbb{O}(AB, IE, VI, ES, EE, \lambda)$ , where  $AB$  are the heads of the PCP represented as a single four-dimensional Morton block,  $IE$  is a Boolean flag that indicates if the tuple representing a PCP stores an intermediate vertex or edge. If  $IE$  is set to *false*, then  $VI$  stores an intermediate vertex, else  $\langle ES, EE \rangle$  represents an intermediate edge. Finally, the value of  $\lambda$  represents the approximate network distance between the heads of the PCP. Recall that both the heads of the PCP,  $A$  and  $B$ , correspond to nodes of a PR-quadtrees on  $V$ . Hence,  $(A, B)$  can be compactly represented as a single four-dimensional Morton block [10]. A four-dimensional Morton block  $AB$  of  $(A, B)$  is obtained by first constructing the Morton blocks corresponding to  $A$  and  $B$ , and then bit-interleaving them to obtain  $AB$ . Now, the attribute  $AB$  in the relation  $\mathbb{O}$  is indexed using a B-tree or  $B^+$ -tree, and the resulting representation is known as a *linear quadtree* [10], which is a disk-efficient access structure.

Given a source  $u$  and destination  $v$ , we can obtain an intermediate vertex or edge in the shortest path between  $u$  and  $v$  as well as an approximate network distance  $\lambda$  using the relation  $\mathbb{O}$  by first constructing the four-dimensional Morton block  $Z_4(u, v)$  of  $(u, v)$ , and using it to search the B-tree or  $B^+$ -tree on  $AB$  for the longest *prefix match* to  $Z_4(u, v)$ , where  $Z_4(\cdot, \cdot)$  is a function that takes a pair of two dimensional points and converts them into a single four-dimensional Morton block representation. Note that  $Z_4(\cdot, \cdot)$  can be computed in  $O(1)$  time using bit-operations. Searching the index on  $AB$  takes logarithmic time in the number of tuples in  $\mathbb{O}$ . We will later show that this can be achieved in  $O(\log n)$  time as the size of the oracle is linear in  $n$ . The above operation corresponds to a simple “SELECT” operator on  $\mathbb{O}$ . The shortest path between  $u$  and  $v$  can be retrieved by the repeated application of the SELECT operation on  $\mathbb{O}$ , which is described next.

### 3.5 Finding Shortest Paths

Algorithm 2 takes a source vertex  $u$  and a destination vertex  $v$  as inputs and retrieves the shortest path  $\pi_G(u, v)$  between  $u$  and  $v$  in  $G$ . In lines 1–3, we check if  $u$  and  $v$  are the same, in which case, the algorithm returns  $\{u\}$ . This is the *base case* of the algorithm. In line 4, using the Morton block  $Z_4(u, v)$  of  $u, v$  as the search key on the B-tree on  $AB$ , we can obtain tuple in  $\mathbb{O}(AB, IE, VI, ES, EE, \lambda)$ , such that  $AB$  contains  $Z_4(u, v)$ . Note that the nature of the PCP decomposition guarantees that there will be exactly one matching tuple in  $\mathbb{O}$  for any search key  $Z_4(u, v)$ . If  $A$  and  $B$  share an edge (in which case  $IE$  is set to *true* as shown in lines 5–6),  $\pi_G(u, v)$  is represented as a composition of  $\text{SPATH}(u, ES) \rightsquigarrow \langle ES, EE \rangle \rightsquigarrow \text{SPATH}(EE, v)$ , resulting in subsequent recursive calls to Algorithm 2. If  $A$  and  $B$  share an intermediate vertex  $VI$  (in which case  $IE$  is set to *false*), we recursively invoke two instances of Algorithm SPATH with inputs  $(u, VI)$  and  $(VI, v)$  as shown in line 8.

#### Algorithm 2

**Procedure** SPATH[ $u, v$ ]

**Input:**  $u, v \leftarrow$  Source and Destination vertices

**Output:** Shortest path  $\pi_G(u, v)$  between  $u$  and  $v$   
 (\*  $\mathbb{O}(AB, IE, VI, ES, EE, \lambda) \leftarrow$  Oracle relation \*)

1. **if**  $u = v$  **then** (\* Base case of the recursion \*)
2. **return**  $\{u\}$

3. **end-if**
4. SELECT IE, VI, ES, EE FROM  $\mathbb{O}$  WHERE  $AB = Z_4(u, v)$
5. **if**  $(IE = \text{true})$  **then**
6. **return**  $\text{SPATH}(u, ES) \rightsquigarrow \langle ES, EE \rangle \rightsquigarrow \text{SPATH}(EE, v)$
7. **else** (\*  $IE = \text{false}$  \*)
8. **return**  $\text{SPATH}(u, VI) \rightsquigarrow \text{SPATH}(VI, v)$
9. **end-if**

### 3.6 Query Processing

We now outline how to incorporate a path-distance oracle in a SQL-based system that can perform complex queries on a spatial network using a relational database system. Let  $\mathbb{O}$  be the path-distance oracle with the schema:  $\mathbb{O}(AB, IE, VI, ES, EE, \lambda, \epsilon_L, \epsilon_H)$ . Notice that we have expanded on the schema from Section 3.4 by augmenting two additional attributes  $\epsilon_L$  and  $\epsilon_H$ ,  $\epsilon_L \leq \epsilon_H \leq \epsilon$  which are the minimum and maximum errors, respectively, due to approximating all the network distances from  $A$  to  $B$  with  $\lambda$ . The value of  $\epsilon_L$  and  $\epsilon_H$  can be trivially obtained by modifying the procedure FINDPATHCOHERENTPAIRS in Algorithm 1 (line 9) to compute and associate it with a PCP. We also introduce a *refinement* operator which when given a source  $u$  and destination  $v$ , can improve upon the quality of the approximation (known as the *effective error*) provided by  $\lambda$  by retrieving additional intermediate vertices in the shortest path from  $u$  to  $v$ . Note that an approximate error can be refined at most  $k$  times before the exact network distance is obtained, where  $k$  is the number of intermediate vertices in the shortest path between  $u$  and  $v$ . Also, we assume that the refinement operator will keep its own state information such as the set of intermediate vertices, effective error, etc., all of which are abstracted from the user of this system. In addition, we assume that a number of macros have been provided which given a source and destination vertex, will keep invoking the refinement operator as many times as needed to either achieve a predicted approximation quality, or until a certain condition is satisfied. We make use of a few macros below, but due to space constraints, we assume that their workings are clear from their assigned descriptive names.

**Approximate Network Distance Query:** Given source vertex  $p$  and destination vertex  $q$ , find an intermediate vertex or edge, approximate network distance  $\lambda$ , and minimum  $\epsilon_L$  and maximum  $\epsilon_H$  errors due to the use of the approximation.

```
SELECT IE, VI, ES, EE  $\lambda$ ,  $\epsilon_L$ ,  $\epsilon_H$  FROM  $\mathbb{O}$  WHERE  $AB = Z_4(p, q)$ 
```

**$\eta$ -approximate Network Distance Query:** Given source  $p$  and destination  $q$ , find an approximate network distance with an approximation quality of  $\eta$  or better. The macro REFINE\_UNTIL invokes the refinement operator until the desired approximation  $\eta$  is attained.

```
SELECT  $\lambda$ ,  $\epsilon_L$ ,  $\epsilon_H$  FROM  $\mathbb{O}$  WHERE  $AB = Z_4(p, q)$  and  
REFINE_UNTIL(EFFECTIVE_ERROR_OF( $\mathbb{O}$ ). $\lambda$ )  $\leq \eta$ 
```

**Region Search:** Given a query location  $q$ , find all restaurants  $R(\text{position}, \text{type})$  that are within 10 miles of  $q$ . The macro REFINE\_UNTIL\_DECIDE\_IF invokes the refinement operator until it is clear if a restaurant is within 10 miles of  $q$  or is not.

```
SELECT  $R$ .position,  $\lambda$  FROM  $\mathbb{O}, R$  WHERE  $AB = Z_4(q, R$ .position) and  
REFINE_UNTIL_DECIDE_IF( $\mathbb{O}$ . $\lambda$   $\leq 10$  miles)
```

**$k$ -Nearest Neighbor Search:** Given a query location  $q$ , find the  $k$  closest restaurants in  $R(\text{position}, \text{type})$  to  $q$  that serve *Italian* cuisine. The results are produced by sorting on the  $\lambda$  values using a comparator (not given here) that is modified so that given two restaurants at distances  $\lambda_1$  and  $\lambda_2$ , the comparator refines both  $\lambda_1$  and  $\lambda_2$  until it is clear if  $\lambda_1 \leq \lambda_2$  or  $\lambda_1 > \lambda_2$ .

```
SELECT R.position, λ FROM Q, R
WHERE AB = Z4(q, R.position) and R.type = "Italian"
ORDER BY Q.λ LIMIT k
```

**Distance Join Operator:** Suppose that in addition to  $R$ , we are also given a relation of coffee shops  $Q(\text{position}, \text{type})$ . Now, find the  $k$  closest pairs of restaurants and coffee shops. This query uses the same sorting operator on  $\lambda$  as in the  $k$ -nearest neighbor algorithm.

```
SELECT R.position, Q.position, λ FROM Q, R, Q
WHERE AB = Z4(R.position, Q.position)
ORDER BY Q.λ LIMIT k
```

## 4. ANALYSIS

In this section, we provide bounds on the size of the decomposition of  $G$  into PCPs by appealing to the equivalence between the PCP decomposition of a spatial network and the WSP decomposition of a point set. The rest of this section is organized as follows. Section 4.1 introduces the concept of a Well-Separated Pair (WSP) decomposition of a point set. Next, Section 4.2 develops a model of a path-coherent spatial network and shows that the size of the WSPD using the network distance measure is linear in  $n$ . Section 4.3 discusses the *four coherent paths* problem, using it in Section 4.4 to arrive at our final result that the WSPD of a spatial network for a suitable WSP parametric value  $s$  is a PCP decomposition. Finally, Section 4.5 derives the size of the PCP decomposition of a spatial network that is a regular grid, which is one of the problematic cases of the PCP decomposition.

### 4.1 Preliminaries

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 *minimum distance* between  $A$  and  $B$  is the minimum possible distance between any point in  $A$  to any point in  $B$ . Two sets of points  $A$  and  $B$  are said to be *well-separated* if the minimum 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 two sets. The pair  $(A, B)$  is termed a *Well-Separated Pair* (WSP). A *Well-Separated Pair Decomposition* (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  $p \in A, q \in B$ . Given a point set  $R$  in a  $d$ -dimensional space, we construct a WSPD of  $R$  by first constructing a PR-quadtrees [10]  $H$  on  $R$ . For the sake of simplicity, we assume that  $R$  is contained in a unit  $[0, 1]^d$   $d$ -dimensional hypercube. This hypercube forms the root block  $T$  of  $H$ . The PR-quadtrees is constructed by recursively decomposing the block into  $2^d$  congruent children blocks. The process continues until each block contains a single point, in which case, further subdivision is not possible. Unfortunately, if two points are close to one another in  $R$ , it may lead to a long path of trivial blocks of which only one block would form an internal node. Callahan and Kosaraju's construction [1] did not incur this problem because they used a fair-split tree which is a data-dependent decomposition. This problem is overcome by Fischer and Har-Peled [4] through the use of a variant of a *path compressed* quadtree which is obtained from the PR-quadtrees by compressing such trivial paths into one single compressed link. The advantage of the path compressed quadtree over the PR-quadtrees is that its use results in a tree with a total of  $O(n)$  nodes.

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  $\Delta$  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  $H$  be a PR-quadtrees on  $V$ , the maximum depth of  $H$  is  $O(\log \Delta)$ . Consequently, given a vertex  $v$  in  $V$ , the Morton block representation of  $p(v)$ , the spatial position of  $v$ , would be  $O(\log \Delta)$  bits long. To cast this quantity in terms of  $n$ , we note that even if the data is heavily skewed so that  $\Delta$  is linear in  $n$ , the length of the Morton block of  $v$  would still be  $O(\log n)$ . We claim that this assumption fits closely with the actual nature of real road networks. From a practical standpoint with respect to our experience with road networks, we observe that the minimum geodesic distance between any two vertices on a road network is at least 1 meter. A PR-quadtrees on a sphere corresponding to the Earth with radius 6378 km and depth 24, has a 1 meter resolution at the equator. For such data, the size of the Morton block for a vertex on the road network using geographical coordinates is at most 48 bits in length.

The following result is due to Callahan and Kosaraju [1] 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  $R$ ,  $R \otimes R$  has  $O(s^d n)$  WSPs [1]*

### 4.2 Path-Coherent Spatial Networks

We now introduce the concept of Well-Separated Pair Decomposition (WSPD) of a spatial network by first making two key assumptions about the nature of spatial networks. We first assume that the ratio between the network and spatial distances is bounded from both above and below (Assumption 1). Note that for any given finite graph, we can always compute the values of  $\gamma_1$  and  $\gamma_2$ , albeit large. The interested reader is referred to [13] for methods to compute  $\gamma_1$  and  $\gamma_2$  values of a spatial network.

ASSUMPTION 1.  $\gamma_1 \leq \frac{d_G(u,v)}{d_S(u,v)} \leq \gamma_2, u, v \in V, \gamma_1$  and  $\gamma_2 > 0$ .

At this point, we show how to extend the notion of a well-separated pair decomposition 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 a WSPD with a separation factor  $s$  of the vertices  $V$  of  $G(V, E)$  based on a spatial distance also yields a WSPD with a separation factor of  $s'$  of  $V$  using a network distance such that  $s' = s \cdot \frac{\gamma_1}{\gamma_2}$  [13].*

DEFINITION 4. *Two cycle free paths  $\pi_1(u_1, v_1)$  and  $\pi_2(u_2, v_2)$  are core-disjoint, if and only if they share no common vertices besides the source or destination vertices. Formally  $\pi_1(u_1, v_1)$  and  $\pi_2(u_2, v_2)$  are core-disjoint if and only if  $\pi_1 \cap \pi_2 = (\{u_1\} \cap \{v_1\}) \cup (\{u_2\} \cap \{v_2\})$ .*

DEFINITION 5. *The shortest core-disjoint path  $\pi(u, v)$  from  $u$  to  $v$  in  $G$  is a cycle-free path  $\pi$  between  $u$  and  $v$  in  $G$  that is core-disjoint from  $\pi_G(u, v)$  such that  $w(\pi)$  is minimized.*

We now introduce another assumption on the nature of our spatial networks which enables us to ensure that there cannot be many shortest core-disjoint paths of the same length between subsets of sources and destinations in a spatial network. Spatial networks for which the lengths of the alternate shortest core-disjoint paths are lower bounded are said to be *path-coherent*. The motivation is that the greater the number of shortest core-disjoint paths of the same length, the less likelihood that the shortest paths share a common vertex (i.e., be path-coherent).

**ASSUMPTION 2.** Given any  $u, v \in V$  in a spatial network  $G(V, E)$ , we let  $\frac{1}{\delta}$ ,  $\delta > 1$ , serve as an upper bound on the ratio of the network distance  $d_G(u, v)$  along the shortest path  $\pi_G(u, v)$  between  $u$  and  $v$ , to the network distance along the shortest core-disjoint path to  $\pi_G(u, v)$ .

We now define two concepts that builds on Assumption 2.

**DEFINITION 6.** The shortest path between any pair of vertices  $(u, v)$  is said to be  $\delta$ -redundant with respect to  $G(V, E)$  if and only if for any path  $\pi$  from  $u$  to  $v$  that is core-disjoint from  $\pi_G(u, v)$ , we have  $w(\pi) \geq \delta w(\pi_G(u, v))$ .

**DEFINITION 7.** A spatial network  $G(V, E)$  is said to be path-coherent if and only if the shortest paths between every pair of vertices are  $\delta$ -redundant.

The careful reader will have observed that our model of a path-coherent spatial network does not include spatial networks that are regular grids, as given a source  $u$  and destination  $v$  vertex on such a spatial network  $G$  there is more than one shortest core-disjoint path between  $u$  and  $v$  of the same length. This also means that the value of  $\delta$  for  $G$  is 1 and Assumption 2 does not hold. Please note that the correctness of Algorithm 1 when applied to regular grid spatial networks is not under question here. Instead, the storage bounds of the PCP decomposition derived in this Section are not applicable to them. We address the size of the PCP decomposition of regular grids separately in Section 4.5. Finally, it is clear that real road networks are not path-coherent spatial networks as there may be a small percentage of source and destinations vertices with multiple shortest core-disjoint paths of equal length. Although our analysis assumes that the minimum value of  $\delta$  computed over all the  $n^2$  shortest paths in a spatial network is greater than 1, empirical results in Section 5 show that even for road networks like Manhattan, NY where a large percentage of vertices lying in a regular grid, the storage bounds derived in this Section hold pretty well. We state the following lemma without providing a proof.

**LEMMA 4.3.** If the path  $\pi_1(u_1, v_1)$  is core-disjoint from the two paths  $\pi_2(u_1, t)$  and  $\pi_3(t, v_2)$ , then  $\pi_1$  is core-disjoint from  $\pi_2 \rightsquigarrow \pi_3$ .

### 4.3 Four Coherent Paths Problem

We now examine the nature of the shortest paths between two subsets of vertices  $(A, B)$  in a path-coherent spatial network that are well-separated. In particular, we will show that for a suitably defined value of  $s$ , four shortest paths from two sources in  $A$  to two destinations in  $B$  will always have some vertices in common.

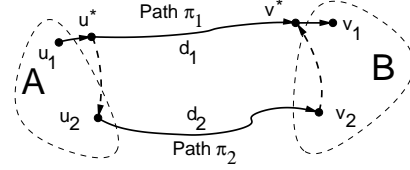
**DEFINITION 8.** The branch-out vertex of a path  $\pi_1$  with respect to a path  $\pi_2$  is the last vertex of  $\pi_1$  that is in  $\pi_1 \cap \pi_2$ .

**DEFINITION 9.** The branch-in vertex of a path  $\pi_1$  with respect to a path  $\pi_2$  is the first vertex of  $\pi_1$  that is in  $\pi_1 \cap \pi_2$ .

**LEMMA 4.4.** Consider a WSP  $(A, B)$  in the WSPD of a path-coherent spatial network  $G(V, E)$  with  $s > 2 + \frac{2}{\delta-1}$ . For  $u_1, u_2 \in A$  and  $v_1, v_2 \in B$ , the shortest paths  $\pi_1 = \pi_G(u_1, v_1)$  and  $\pi_2 = \pi_G(u_2, v_2)$  are not disjoint.

**PROOF.** Assume to the contrary that the paths  $\pi_1$  and  $\pi_2$  are disjoint. Figure 3 shows an example of such a scenario. Let  $d_1 = w(\pi_1)$  and  $d_2 = w(\pi_2)$ . Without loss of generality, we assume that  $d_2 \leq d_1$ .

Consider the path  $\pi_G(u_1, u_2)$ . Let  $u^*$  be the branch-out vertex of  $\pi_G(u_1, u_2)$  with respect to  $\pi_G(u_1, v_1)$ . Similarly, let  $v^*$  be the



**Figure 3:**  $(A, B)$  is a WSP configuration containing two disjoint paths between them

branch-in vertex of  $\pi_G(v_2, v_1)$  with respect to  $\pi_G(u_1, v_1)$ . Notice that  $\pi_G(u^*, v^*)$  is a subpath of  $\pi_1$  and is hence disjoint from  $\pi_2$  because of our assumption. The definition of  $u^*$  and  $v^*$  ensure that  $\pi_G(u^*, u_2)$  and  $\pi_G(v_2, v^*)$  are both core-disjoint from  $\pi_G(u^*, v^*)$ . Let  $d^* = d_G(u^*, v^*)$ .

Consider the path  $\pi_3 = \pi_G(u^*, u_2) \rightsquigarrow \pi_2$ .  $\pi_G(u^*, v^*)$  is core-disjoint from  $\pi_3$  due to Lemma 4.3. Consider the path  $\pi^* = \pi_3 \rightsquigarrow \pi_G(v_2, v^*)$  between  $u^*$  and  $v^*$ . By re-applying Lemma 4.3, we claim that  $\pi_G(u^*, v^*)$  is core-disjoint from  $\pi^*$ .

For  $r_G$ , the larger diameter of  $A$  and  $B$ , we have,  $d_G(u_1, u_2) \leq r_G$ , and  $d_G(v_2, v_1) \leq r_G$ . Therefore,  $d_G(u_1, u_2) + d_G(v_2, v_1) \leq 2r_G$ .

We have,

$$d_1 = d_G(u_1, u^*) + d^* + d_G(v^*, v_1) \quad (1)$$

$$w(\pi^*) = d_G(u^*, u_2) + d_2 + d_G(v_2, v^*) \quad (2)$$

Adding Equations 1 and 2,

$$\begin{aligned} w(\pi^*) + d_1 &= d_G(u_1, u^*) + d_G(u^*, u_2) + d^* + \\ &\quad d_2 + d_G(v_2, v^*) + d_G(v^*, v_1) \\ w(\pi^*) + d_1 &= d_G(u_1, u_2) + d_G(v_2, v_1) + d^* + d_2 \\ w(\pi^*) + d_1 &\leq 2r_G + d^* + d_2 \end{aligned} \quad (3)$$

By the assumption of  $\delta$ -redundancy, we have  $w(\pi^*) \geq \delta d^*$ . Combining this with Equation 3, we get,  $\delta d^* + d_1 \leq 2r_G + d^* + d_2$ , and as  $d_2 \leq d_1$  we get  $\delta d^* + d_1 \leq 2r_G + d^* + d_1$ , or,

$$(\delta - 1)d^* \leq 2r_G \quad (4)$$

As  $(A, B)$  is a WSP, the network distance  $d_1$  between the pair of points  $u_1 \in A$  and  $v_1 \in B$  is at least  $s \cdot r_G$  and thus  $s \cdot r_G \leq d_1$ . Combined this result with Equation 1 we get,  $s \cdot r_G \leq d_G(u_1, u^*) + d^* + d_G(v^*, v_1)$ . But as  $d_G(u_1, u^*) \leq d_G(u_1, u_2) \leq r_G$  and  $d_G(v^*, v_1) \leq d_G(v_2, v_1) \leq r_G$ , we get  $s \cdot r_G \leq r_G + d^* + r_G$ , or

$$(s - 2)r_G \leq d^* \quad (5)$$

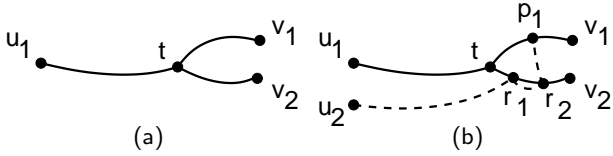
By combining Equations 4 and 5, we get  $(\delta - 1)(s - 2)r_G \leq 2r_G$ , but  $s > \frac{2}{\delta - 1} + 2$  and hence by substituting for  $s$ , we get  $2 < 2$ . This contradicts our initial assumption that  $\pi_1$  and  $\pi_2$  are disjoint. Therefore  $\pi_1$  and  $\pi_2$  must not be disjoint.  $\square$

We now state the following two properties of shortest paths without providing a proof.

**LEMMA 4.5.** If  $t \in a \rightsquigarrow b$  and  $r \in a \rightsquigarrow t$ , then  $t \in r \rightsquigarrow b$ .

**LEMMA 4.6.** If  $r \in a \rightsquigarrow t$  and  $t \in a \rightsquigarrow r$  then  $t = r$ .

**LEMMA 4.7.** [Four Coherent Paths Problem] Consider two source vertices  $u_1, u_2$  and two destination vertices  $v_1, v_2$  in a path-coherent spatial network  $G(V, E)$ , such that there exist four shortest paths between the source and the destination pairs, namely,  $\pi_1 = u_1 \rightsquigarrow v_1$ ,  $\pi_2 = u_1 \rightsquigarrow v_2$ ,  $\pi_3 = u_2 \rightsquigarrow v_1$ , and  $\pi_4 = u_2 \rightsquigarrow v_2$ . Given that no two shortest paths are disjoint from Lemma 4.4, there exists a vertex  $t$  that is common to all the four shortest paths.



**Figure 4: (a) Shortest paths  $\pi_1$  and  $\pi_2$ , where  $t$  is the branch-out vertex of  $\pi_1$  ( $\pi_2$ ) with respect to  $\pi_2$  ( $\pi_1$ ), (b) Shortest path  $\pi_3$  has been added to the setup in (a), such that  $r_1 \in t \rightsquigarrow v_2$ ,  $r_2 \in t \rightsquigarrow v_2$ ,  $p_1 \in t \rightsquigarrow v_1$ ,  $v_1 \neq t$ ,  $v_2 \neq t$ , and  $p_1 \neq t$**

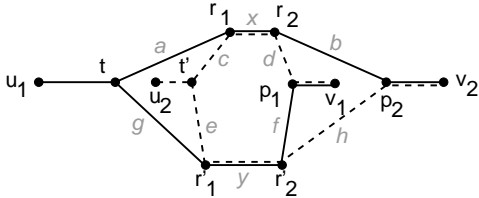
PROOF. Figure 4a shows the shortest paths  $\pi_1 = u_1 \rightsquigarrow v_1$  and  $\pi_2 = u_1 \rightsquigarrow v_2$ , such that  $t$  is the branch-out vertex of  $\pi_1$  ( $\pi_2$ ) with respect to  $\pi_2$  ( $\pi_1$ ). The vertex  $t$  can be trivially shown to exist, as  $t = u_1$  is at least one vertex that is always common to  $\pi_1$  and  $\pi_2$ . We refer to  $\pi_G(u_1, t)$  as the *stem* of the shortest paths.

We now add the  $\pi_3 = u_2 \rightsquigarrow v_1$  to the setup in Figure 4a, while ensuring that that  $\pi_3$  is not disjoint from either  $\pi_1$ , or  $\pi_2$ . Let  $r_1$  be the incoming vertex of  $\pi_3$  with respect to  $\pi_2$ . Let  $r_2$  be the branch-out vertex of  $\pi_3$  with respect to  $\pi_2$ . Let  $p_1$  be the incoming vertex of  $\pi_3$  with respect to  $\pi_1$ . Note that  $\pi_3$  cannot have an branch-out vertex with respect to  $\pi_1$ , as  $p_1 \rightsquigarrow v_1$  is a common subpath to both  $u_1 \rightsquigarrow v_1$  and  $u_2 \rightsquigarrow v_1$ .

We point out that there can only be three possible configurations of  $r_1, r_2$  and  $p_1$  with respect to  $t, u_1, v_1$  and  $v_2$ , which are listed below.

1. If  $r_1 \neq t, r_1 \in t \rightsquigarrow v_2, r_2 \in u_1 \rightsquigarrow v_2$ , and  $p_1 = t$ , it can be verified that path  $\pi_4$  cannot exist now without being disjoint from either  $\pi_1$ , or  $\pi_2$ . Hence, this case is infeasible.
2. If  $r_1 \neq t, r_2 \neq t, r_1, r_2 \in t \rightsquigarrow v_2$ , and  $p_1 \in t \rightsquigarrow v_1, p_1 \neq t$ , we show that  $\pi_4$  can exist only iff  $\delta = 1$ , and hence is infeasible.
3. If  $r_1, r_2, p_1 \in u_1 \rightsquigarrow t$ , we show that  $\pi_4$  can exist only iff  $r_1 = r_2 = p_1$ , thus proving the lemma.

Figure 4b illustrates case-2, which we show to be infeasible. We have added the shortest path  $\pi_4 = u_2 \rightsquigarrow v_2$  to the configuration in Figure 4b, resulting in Figure 5.



**Figure 5: Figure 4b has been redrawn to include  $\pi_4$ , such that  $r'_2 \in t \rightsquigarrow p_1$  and  $p_2 \notin r_1 \rightsquigarrow r_2, p_2 \in r_1 \rightsquigarrow v_2$ . Note that labels  $a-h, x, y$  on an edge indicates its weight.  $r'_1 \rightsquigarrow r'_2$  is the subpath shared between  $t \rightsquigarrow p_2$  and  $t' \rightsquigarrow p_1$ . Similarly,  $r_1 \rightsquigarrow r_2$  is shared between  $t' \rightsquigarrow p_2$  and  $t \rightsquigarrow p_1$**

Let  $r'_1$  and  $r'_2$  be incoming and outgoing vertices of  $\pi_4$  with respect to  $\pi_1$ . While ensuring that  $\pi_4$  is not disjoint with  $\pi_1$ , we observe that the only feasible condition is when  $r'_1 \in t \rightsquigarrow p_1$  and  $r'_2 \in t \rightsquigarrow p_1$  is satisfied. Let  $p_2$  be the incoming vertex of  $\pi_4$  with respect to  $\pi_2$ , and also  $p_2 \notin r_1 \rightsquigarrow r_2, p_2 \in r_1 \rightsquigarrow v_2$ . Figure 5 shows the shortest path configuration containing the four shortest paths. Also, the labels ( $a-h, x, y$ ) assigned to edges in Figure 5 correspond to the weights of the edges. We now show that this configuration is realizable, iff  $\delta = 1$ .

In Figure 5, we observe that  $t \rightsquigarrow p_1$  is at a distance  $g + y + f$ , while an alternate core-disjoint path  $t \rightsquigarrow r_1 \rightsquigarrow r_2 \rightsquigarrow p_1$  is at a dis-

tance  $a + x + d$ , leading to

$$a + x + d \geq \delta(g + y + f) \quad (6)$$

Similarly,  $t \rightsquigarrow p_2$  is at a distance  $a + x + b$ , while an alternate core-disjoint path  $t \rightsquigarrow r'_1 \rightsquigarrow r'_2 \rightsquigarrow p_2$  is at a distance  $g + y + h$ , leading to

$$g + y + h \geq \delta(a + x + b) \quad (7)$$

$t' \rightsquigarrow p_1$  is at a distance  $c + x + d$ , while an alternate core-disjoint path  $t' \rightsquigarrow r'_1 \rightsquigarrow r'_2 \rightsquigarrow p_2$  is at a distance  $e + y + f$ , leading to

$$e + y + f \geq \delta(c + x + d) \quad (8)$$

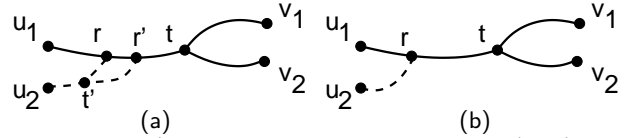
$t' \rightsquigarrow p_2$  is at a distance  $c + x + b$ , while an alternate core-disjoint path  $t' \rightsquigarrow r_1 \rightsquigarrow r_2 \rightsquigarrow p_2$  is at a distance  $e + y + h$ , leading to

$$e + y + h \geq \delta(c + x + b) \quad (9)$$

Now adding the inequalities in Equations 6–9, we get

$$(a + b + c + d + e + f + 2 \cdot x + 2 \cdot y) \geq \delta(a + b + c + d + e + f + 2 \cdot x + 2 \cdot y)$$

As,  $(a + b + c + d + e + f + 2 \cdot x + 2 \cdot y) > 0$ , we get  $\delta = 1$ , which contradicts our assumption that  $\delta > 1$ . Hence, case-2 is infeasible.



**Figure 6: a)  $r, r' \in u_1 \rightsquigarrow t$ , such that  $r = r_1 = p_1, r' = r'_1 = p_2$ , and  $t'$  is the branch-out vertex of  $\pi_3$  ( $\pi_4$ ) with respect to  $\pi_4$  ( $\pi_3$ ), b) the only feasible configuration is when  $r = r'$  and  $t = t'$**

We now examine case-3, when  $r_1, r_2, p_1 \in u_1 \rightsquigarrow t$ . First of all, we can trivially show that  $r_2 = t$ , and  $p_1 = r_1$ . Let  $r'_1$  and  $r'_2$  be incoming and outgoing vertices of  $\pi_4$  with respect to  $\pi_1$ . Let  $p_2$  be the incoming vertex of  $\pi_4$  with respect to  $\pi_2$ . Upon adding  $\pi_4$ , we can further show that that  $r'_1 = p_2$  and  $r'_2 = t$ . Let  $r' = r'_1 = p_2, r = r_1 = p_1$ , and  $t'$  is the branch-out vertex of  $\pi_3$  ( $\pi_4$ ) with respect to  $\pi_4$  ( $\pi_3$ ). The resulting configuration is shown in Figure 6a.

We can now further claim that  $r = r'$ , failing which there would be two shortest paths from  $u_2$  to  $r'$  or alternately, from  $u_2$  to  $r$ , if  $r' \in u_1 \rightsquigarrow r$ . Hence, the only feasible configuration is shown in Figure 6b, where  $r$  is the incoming vertex of  $\pi_3$  and  $\pi_4$  with respect to  $\pi_1$  and  $\pi_2$ , and  $t = t'$  is the common vertex to the four shortest paths.  $\square$

#### 4.4 Size of the Oracles

In this Section, we prove a key property of path-coherent networks that a WSPD of a spatial network using a network distance function for a suitable value of  $s$  is a PCP decomposition.

LEMMA 4.8. *Given a WSPD decomposition of a path-coherent spatial network  $G(V, E)$  with a separation factor  $s > 2 + \frac{2}{\delta-1}$ , such that  $(A, B)$ , is a WSP in the decomposition, then all the shortest paths from source vertices in  $A$  to destination vertices in  $B$  pass through one single common vertex. In other words, for  $s > 2 + \frac{2}{\delta-1}$ , the WSPD of  $G$  is a PCP decomposition of  $G$ .*

PROOF. We adopt the following strategy in proving this lemma. Our initial configuration continues where we left off in Lemma 4.7. We know for  $s > 2 + \frac{2}{\delta-1}$ , the four shortest paths between two source vertices  $u_1$  and  $u_2$  in  $A$ , and destination vertices  $v_1$  and  $v_2$  in  $B$  pass through a single common vertex  $t$ . We will now keep adding one additional destination vertex from  $B$  to this arrangement, in no

particular order, until all the shortest paths from  $u_1$  and  $u_2$  to all vertices in  $B$  have been accounted for. We then start adding one additional source vertex at a time from  $A$  to the arrangement, until all the possible shortest paths from  $A$  to  $B$  have been accounted for. At the end of it, we will show that there will still be one vertex that is in common to all the shortest paths.

Suppose  $u_1, u_2 \in A$  are source vertices and  $v_1, v_2 \in B$  are destination vertices such that the four possible shortest paths between them are  $\pi_1 = \pi_G(u_1, v_1)$ ,  $\pi_2 = \pi_G(u_1, v_2)$ ,  $\pi_3 = \pi_G(u_2, v_1)$ , and  $\pi_4 = \pi_G(u_2, v_2)$ . Let  $t$  be the common vertex to the four shortest paths and corresponds to the branch-out vertex of  $\pi_1$  with respect to  $\pi_2$ . Also,  $r$  is the incoming vertex of  $\pi_1$  and  $\pi_3$  with respect to  $\pi_2$  and  $\pi_4$ .

We claim that the addition of an additional vertex  $v_3 \in B$ , may potentially replace  $t$  with another vertex  $t'$ , such that  $t' \in r \rightsquigarrow t$ . Thus, after all vertices  $v \in B$  have been added,  $r \in u_1 \rightsquigarrow t$  would still have a vertex in common. With the addition of all the destination vertices in  $B$ , we would have accounted for the shortest paths from  $u_1$  and  $u_2$  to all destination vertices in  $B$ .

Let  $v_3$  be a destination vertex in  $B$ . Let  $\pi_5 = \pi_G(u_1, v_3)$ , and  $\pi_6 = \pi_G(u_2, v_3)$  be the shortest paths from  $u_1$  and  $u_2$  to  $v_3$ . Let  $t_1$  be the branch-out vertex of  $\pi_5$  with respect to  $\pi_1$  and  $\pi_3$ . Let  $t_2$  be the branch-out vertex of  $\pi_6$  with respect to  $\pi_2$  and  $\pi_4$ .

In the three cases below, the addition of  $v_3$  does not affect  $t$ .

- If  $t_1 \in t \rightsquigarrow v_1$  and  $t_1 \neq t$ , then  $t_2 = t$ .
- If  $t_2 \in t \rightsquigarrow v_2$  and  $t_2 \neq t$ , then  $t_1 = t$ .
- $t_1 = t_2 = t$ .

If  $t_1, t_2 \in r \rightsquigarrow t$ , then  $t_1$  must be equal to  $t_2$ . Let  $t' = t_1 = t_2$ . We replace  $t$  with  $t'$ ,  $v_1$  (or  $v_2$ ) with  $v_3$ . This resulting configuration would still resemble Figure 4.

The final case is when  $t_1$  is the branch-out vertex of  $\pi_5$  with respect to  $\pi_1$  and  $\pi_2$ , and let  $r_1, r_2$  be the incoming and branch-out vertex of  $\pi_5$  with respect to  $\pi_3$  and  $\pi_4$ . Similarly, let  $r'_1, r'_2$  be the incoming and branch-out vertex of  $\pi_6$  with respect to  $\pi_1$  and  $\pi_2$ , and let  $t_2$  be the branch-out vertex of  $\pi_6$  with respect to  $\pi_3$  and  $\pi_4$ . The resulting configuration resembles Figure 5, which is only true is  $\delta = 1$ , which is a contradiction.

We have shown that an addition of a destination vertex  $v_3$ , either does not affect  $t$ , in which case, it can be ignored, or replaces  $t$  with  $t' \in r \rightsquigarrow t$ . After all the destination vertices have been added,  $r$  would still satisfy  $r \in u_1 \rightsquigarrow t$  and  $r \in u_2 \rightsquigarrow t$ .

Adding the a source vertices  $u_3$  in  $A$  to the setup in Figure 4b is symmetric to adding a destination vertex  $v_3$ , although the insertion of  $u_3$  may affect  $r$  instead of  $t$ . In effect, after all the source vertices in  $A$  have been accounted for, all the shortest paths from  $A$  to  $B$  pass through  $t$ , which means that the WSP  $(A, B)$  is a PCP.  $\square$

An immediate consequence of Lemma 4.8 is that for the separation factor  $s > 2 + \frac{2}{\delta-1}$ , the WSPD is, in fact, a PCP decomposition. That is — the shortest paths between all sources in  $A$  to all destinations in  $B$  in a WSP  $(A, B)$  pass through a single common vertex or an edge. We now show that given such a decomposition, the shortest path between any vertex pair can be retrieved in  $O(k \log n)$  time, where  $k$  is the length of the shortest path.

**THEOREM 4.9.** *Given a PCP decomposition of a path-coherent spatial network  $G(V, E)$  which is of size  $O(s^d n)$  by Lemma 4.2, the shortest path between any vertex pair in  $V$  can be retrieved from the decomposition in  $O(k \log n)$  time, where  $k$  is the length of the shortest path.*

**PROOF.** Assume a PCP  $(A, B, \Psi, \lambda)$  in the decomposition of a spatial network of size  $l$  such that  $A, B$  are nodes in the PR-quadtrees on the spatial positions of  $V$ . The pair  $A, B$  is represented as a Morton block, which is then stored in a B-tree or a  $B^+$ -tree, termed a linear quadtree [10]. Given a source  $u \in V$  and a destination  $t \in V$ , the PCP containing  $u$  and  $t$  is retrieved by invoking a binary search on the linear quadtree, which takes  $O(\log l)$  time. The entire shortest path can be obtained recursively in  $O(k \log l)$  time, where  $k = |\pi_G(u, t)|$ . As  $l = O(s^d n)$  from Lemma 4.2, the shortest path between any pair of vertices in  $G$  can be retrieved in  $O(k \log n)$  time.  $\square$

This leads to the main result of this paper:

**THEOREM 4.10.** *A path oracle of a path-coherent spatial network of size  $O(s^d n)$  can be constructed that can retrieve an intermediate link of a shortest path in  $O(\log n)$  time using a B-tree.*

Below, we repeat the size bound of an  $\varepsilon$ -approximate distance oracle from [13]:

**THEOREM 4.11.** *An  $\varepsilon$ -approximate distance oracle of a spatial network of size  $O((\frac{1}{\varepsilon})^d n)$  can be constructed that can retrieve an  $\varepsilon$ -approximate network distance between a source and a destination in  $O(\log n)$  time using a B-tree. [13]*

Combining Theorems 4.10 and 4.11, we can now obtain the size of a path-distance oracle.

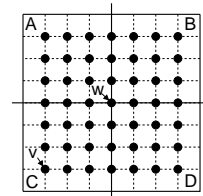
**THEOREM 4.12.** *An  $\varepsilon$ -approximate path-distance oracle of a path-coherent spatial network of size  $O(n \cdot \max(s^d, \frac{1}{\varepsilon}^d))$  can be constructed that can retrieve an intermediate link as well as an  $\varepsilon$ -approximate network distance in  $O(\log n)$  time using a B-tree.*

**PROOF.** A path oracle is a WSPD of  $V$  with a separation factor  $s > 2 + \frac{2}{\delta-1}$ . The distance oracle is also a WSPD of  $V$  with a separation factor of  $\frac{2}{\varepsilon}$ . During the construction of a path-distance oracle in Algorithm 1, a PCP is decomposed until both the path constraint (i.e.,  $\Psi$  exists) and the distance constraint (i.e.,  $\lambda$  is an  $\varepsilon$ -approximation) are fulfilled. Hence, the size of the oracle is upper-bounded by  $O(n \cdot \max(s^d, \frac{1}{\varepsilon}^d))$ .  $\square$

The access times can be further improved to  $O(1)$  by making use of a hash table that takes  $O(s^d n \log n)$  for path oracles and  $O(n \log n \cdot \max(s^d, \frac{1}{\varepsilon}^d))$  for path-distance oracles. For more details, see [12].

## 4.5 PCP Decomposition of Regular Grids

We now derive the size of the PCP decomposition of a spatial network that is a regular grid.



**Figure 7:** A spatial network  $G$  which is a regular grid.  $v$  and  $w$  are vertices in  $G$ .  $A, B, C$  and  $D$  are blocks resulting from the partition of the embedding space spanned by  $G$  into 4 congruent blocks. Notice that  $w$  is an intermediate vertex on the shortest paths from  $A$  to  $B$ ,  $B$  to  $C$ ,  $C$  to  $B$ , and  $D$  to  $A$ .

LEMMA 4.13. *Given a spatial network  $G$  which is a regular grid, the PCP decomposition of  $G$  take  $O(n\sqrt{n})$  space.*

PROOF. Let  $G$  be a spatial network which is a regular grid containing  $n$  vertices as shown in Figure 7. Let  $A, B, C$  and  $D$  be the blocks resulting from the partition of the embedding space spanned by  $G$  into 4 congruent blocks. All the shortest paths from  $A$  to  $B$ ,  $B$  to  $C$ ,  $C$  to  $B$ , and  $D$  to  $A$  pass through the common vertex  $w$  and are recorded using four Morton blocks. However, the shortest paths between the pairs  $(A, A)$ ,  $(A, B)$ ,  $(A, C)$ ,  $(B, B)$ ,  $(B, D)$ ,  $(C, C)$ ,  $(C, D)$ , and  $(D, D)$  would still have to be captured. However, each of the above eight pairs is a smaller instance of the original problem (one-fourth the size), and hence the total storage required by the PCP decomposition of  $G$  in terms of Morton blocks can be represented by the solution of the following recurrence relation:

$$T(n) = 8T\left(\frac{n}{4}\right) + 2 \quad (10)$$

Solving Equation 10, we obtain that the  $n^2$  shortest paths in  $G$  can be captured using  $n\sqrt{n} + n$  Morton blocks.  $\square$

## 5. EXPERIMENTS

In this Section, we evaluate the performance of the oracles obtained by decomposing a spatial network into a set of PCPs. Using this approach, we construct three oracles — path, distance and path-distance — that capture the shortest paths and distances between every pair of vertices in a spatial network. In particular, we will show that the empirical evaluations closely follow our theoretical results. Furthermore, we compare the oracles with the SILC framework [11] which is the only other approach that takes advantage of path coherence in spatial networks. The oracles introduced in this paper capture the coherence in the shortest paths from multiple sources to multiple destinations, which is in contrast to the SILC framework that only captures the coherence between a single source vertex to multiple destination vertices. Note, however, that SILC is still a superior data structure when it comes to query processing such as  $k$  nearest neighbor finding as refinements in SILC result in the retrieval of the next link in the shortest path which means that the refined network distance is a composition of an exact network distance between the source and intermediate vertices, and a network distance interval between the intermediate and destination vertices. In the case of the path-distance oracles, refinement results in the composition of two  $\epsilon$ -approximate network distances, which means that refinements in the path-distance oracles may not be as effective as the SILC framework. We compare the sizes of these two approaches and show that the path oracles yield a smaller representation.

The experiments were carried out 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 datasets were used in the evaluation. These were obtained from the US Tiger Census [15] and the National Atlas [16] websites. Some of the datasets that we used are given in Figure 8a. In particular, we used a dataset containing all the major roads in the USA (*i.e.*, more than 380,000 vertices and 400,000 edges). Sample random rectangular regions were drawn from the dataset and the road network segments contained completely within them were extracted to serve as inputs to the evaluation of our algorithm. By taking the samples at random we were able to account for variations such as rural versus urban, and various spatial arrangements of vertices such as those lying on a regular grid.

In this paper, we derived the size of the path oracles in terms of  $\delta$ , which is the minimum of the ratio of the network distance to the spatial distance recorded over every pair of vertices in a spatial

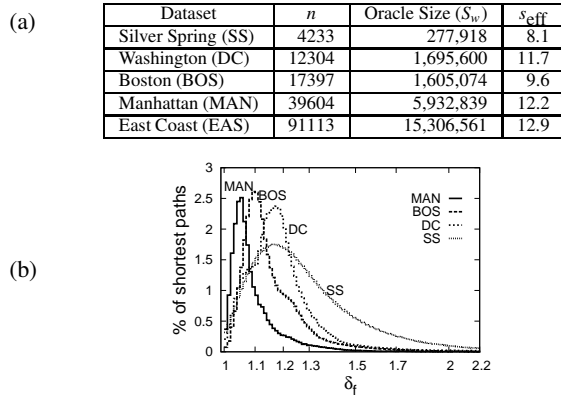
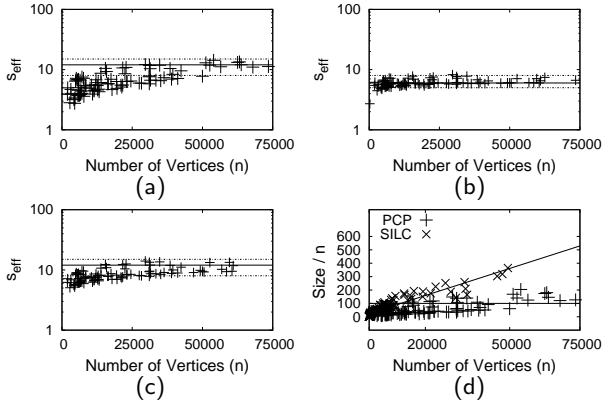


Figure 8: a) Various urban datasets and b) the percentages of shortest paths in them as a function of  $\delta_f$

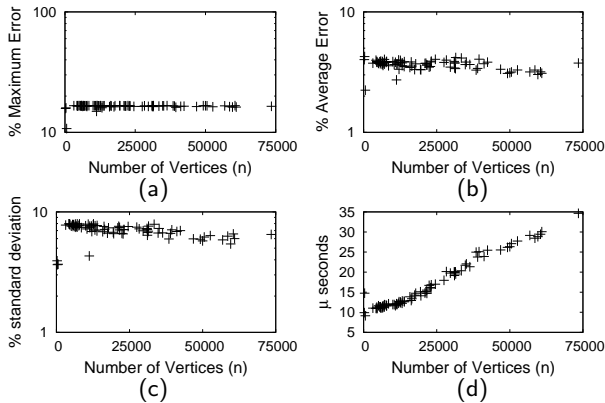
network. Our theoretical model is only applicable to those spatial network that are path-coherent, which, in other words, means that their  $\delta$  value is greater than 1. Road networks are not always path-coherent as there may be a small set of vertex pairs with shortest core-disjoint paths of the same lengths, which means that the value of  $\delta$  of such networks is 1. However, this does not mean that the analysis performed in this paper is not applicable to them. In order to explain why this is the case, we first compute a function for each of the input spatial network datasets in the evaluation which records the ratio  $\delta_f$  of the network and spatial distance between pairs of vertices as a percentage of the total  $n^2$  shortest paths in a spatial network with  $n$  vertices. Note that the minimum  $\delta_f$  value of a road network is its  $\delta$  value, which is usually 1. Figure 8b shows the  $\delta_f$  value distribution of some of the various urban datasets given in Figure 8a. Some of the datasets, especially the road networks of Manhattan (MAN) and Boston (BOS), have a large number of vertices lying on a regular grid. We remind the reader that sub-graphs of road networks that are regular grids will have several source and destination vertex pairs with multiple alternate shortest core-disjoint paths of the same length. The MAN dataset, in particular, has a larger percentage of shortest paths with  $\delta_f$  values very close to 1 compared to the SS, BOS and DC datasets. Figure 8a records the number of vertices  $n$  and the resulting size of the path oracle  $S_w$  in terms of number of PCPs. In addition, we provide the effective separation factor  $s_{\text{eff}}$  of the road network datasets which is defined to be the square-root of the ratio of size of the path oracle  $S_w$ , *i.e.*, the number of PCPs, to the number of vertices  $n$ , *i.e.*,  $s_{\text{eff}} = \sqrt{\frac{S_w}{n}}$ . The  $s_{\text{eff}}$  value of MAN dataset (from Figure 8a) is 12.2 and not unbounded as the theoretical results suggest. It is important to note that even though the MAN dataset represents a bad case scenario for our algorithm, the  $s_{\text{eff}}$  value of the MAN dataset is within acceptable limits. This is because 75.1% of the shortest paths in the MAN dataset have  $\delta_f$  values greater than 1.5, which makes our model of a path-coherent spatial network applicable to datasets such as MAN. In other words, what we have shown is that the size of the path oracle depends more on the average value of  $\delta_f$ , and not so much on the  $\delta$  value, which is 1 for almost all road networks datasets. Moreover, notice that the values of  $s_{\text{eff}}$  did not change considerably between DC and EAS even though the size of the road network dataset essentially increased by a factor of 8. This validates our result in Theorem 4.10 that the size of these oracles are linear in  $n$  which means that the size of the path oracle normalized by  $n$  is a value that is independent of  $n$ .

Next, we further investigate the effect of  $n$  on the size of the path, distance, and path-distance oracles. Figure 9a–c shows that as  $n$  varies, the number of PCPs that make up these oracles, nor-



**Figure 9: Effective separation factor  $s_{\text{eff}}$  of a) path, b) distance, and c) path-distance oracles for  $\epsilon = 0.2$  on road networks of varying sizes. d) A comparison of the sizes of path oracles and SILC framework for varying  $n$**

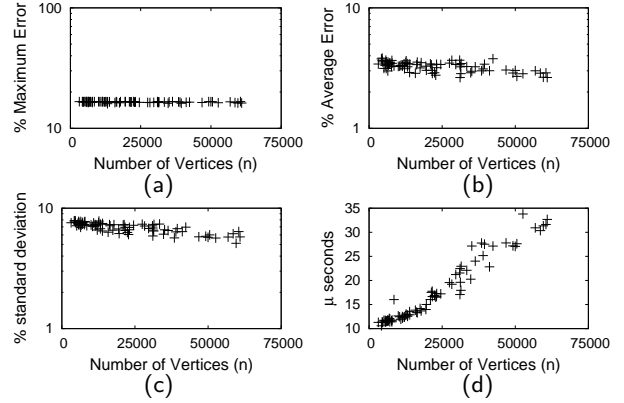
malized by  $n$ , is more or less a constant value. This value, known as the effective separation factor  $s_{\text{eff}}$ , depends only on the nature of the spatial network and not on  $n$ . It is seen that the  $s_{\text{eff}}$  values for all three oracles are more or less constant, independent of  $n$ , indicating that the size of the oracles is linear in the size of the road network. The  $s_{\text{eff}}$  values of the path oracle in Figure 9a lies between 8 and 15 (average: 12), the distance oracle in Figure 9b for  $\epsilon = 0.2$  lies between 5 and 8 (average: 6), and the path-distance oracle in Figure 9c for  $\epsilon = 0.2$  lies between 8 and 15 (average: 12). Note that, although incidental, the path and path-distance oracles for  $\epsilon=0.2$  are more or less of the same size, which means that path oracles associated with approximate distance information will operate at 20% error rates without incurring significant additional space, but the quality can be further improved by expending more storage. We also compared the size of the path oracle with that of the SILC framework in Figure 9d for various road networks of comparable sizes. Observe that the size of path oracle, normalized by  $n$ , is more or less constant as  $n$  varies, while the size of the SILC framework, normalized by  $n$ , steadily increases with  $n$ . In particular, the size of a path oracle for the EAS dataset is less than 50% of the SILC framework. Much higher savings are expected when we apply these two techniques on larger network datasets such as the road network of US as the storage complexity of the SILC framework is  $O(n\sqrt{n})$ , while our representation is linear in  $n$ .



**Figure 10: Observed a) maximum, b) average, and c) standard deviation of the errors of distance oracles and their average access times for  $\epsilon=0.2$  and varying values of  $n$**

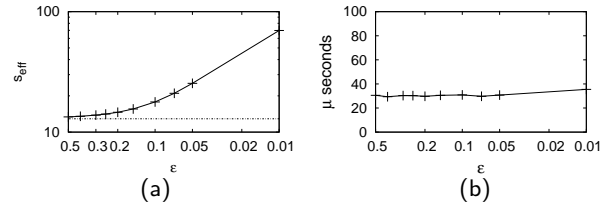
We now study the quality of the distance approximations pro-

duced by our use of the distance oracles. Figure 10a–c shows the observed maximum, average and standard deviation errors computed over 100,000  $\epsilon$ -approximate network distance queries on distance oracles with  $\epsilon = 0.2$  for varying values of  $n$ . Note that the distance oracles (produced by Algorithm 1) do not store any path information. Distance oracles for  $\epsilon = 0.2$  (20%) have a maximum observed error of about 16%, an average observed error of around 3%, and an observed standard deviation of about 6%. Moreover, Figure 10d shows that an  $\epsilon$ -approximate network distance between a given source and destination can be retrieved in 10–35 $\mu$  seconds using our distance oracle.



**Figure 11: Observed a) maximum, b) average, and c) standard deviation of the errors in the network distance when using the path-distance oracles and their average access times for  $\epsilon=0.2$  and varying values of  $n$**

Next, we studied the observed errors in the network distances associated with the use of the path-distance oracle. Recall that a path-distance oracle combines a path oracle with a distance oracle which means that given a source  $s$  and a destination  $w$ , the path-distance oracle provides an intermediate vertex in the shortest path from  $s$  to  $w$  as well as an  $\epsilon$ -approximation of the network distance along this path. Figure 11a–c shows the observed maximum, average and standard deviation errors obtained due the use of the path-distance oracle with  $\epsilon = 0.2$  measured over 100,000  $\epsilon$ -approximate network distance queries for varying values of  $n$ . The observed maximum error was about 16%, the average error was around 3%, and the standard deviation was about 6%. Moreover, Figure 11d shows that the access time lies in the range of 10–40 $\mu$  seconds which gracefully increased with  $n$ .

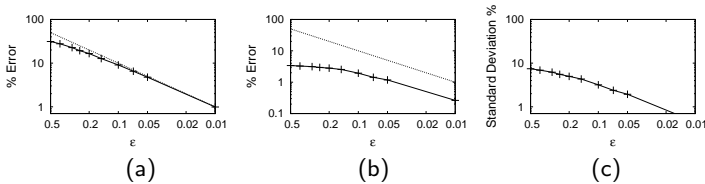


**Figure 12: a) Effective separation factor  $s_{\text{eff}}$  and the b) average access times of path-distance oracles on the EAS dataset for varying values of  $\epsilon$**

Figure 12 shows the effect of varying  $\epsilon$  on the size and the access times when applying the path-distance oracles to the EAS dataset. Recall that this dataset contains all the important roads in the eastern seaboard states of the US, consisting of 91,113 vertices and 114,176 edges. From Theorem 4.12, we know that the size of the path-distance oracle is given by  $O(n \cdot \max(s^2, \frac{1}{\epsilon^2}))$ . In Figure 12a,

we show  $s_{\text{eff}}$ , the effective separation factor, of the path-distance oracle for various values of  $\epsilon$  between 0.5 (50%) and 0.01 (1%). Note that the process of computing the path-distance and path oracles in Algorithm 1 depends on the satisfaction of both the path and distance constraints. In order to evaluate the additional decomposition incurred due to the increased precision  $\epsilon$  required by the path-distance oracle, Figure 12a separates the costs of these two oracles. We use a broken line to depict the cost of the path oracle which is seen to be constant as there is no decomposition due to the distance constraint. On the other hand, the path-distance oracle requires more space as the precision increases (i.e.,  $\epsilon$  decreases) although little extra space is required for values of  $\epsilon$  between 0.5 (50%) and 0.3 (30%). We note that from our experimental analysis that the size of the path oracle for road networks is roughly captured by  $O(n \cdot \max(12^2, \frac{2.5^2}{\epsilon}))$ . Moreover, the average access time, shown in Figure 12b, to obtain an intermediate vertex as well as an  $\epsilon$ -approximate network distance lies in the range of 30–35 $\mu$  seconds that stays more or less the same with decreasing  $\epsilon$ .

Finally, we examine the errors resulting from the use of the path-distance oracle for the EAS dataset for varying values of  $\epsilon$ . Figure 13 shows the observed maximum, average, and standard deviation errors of path-distance oracles obtained by varying the values of  $\epsilon$  between 0.5 (50%) and 0.01 (1%). Figure 13a–b shows that the observed maximum and average errors are less than  $\epsilon$  (depicted by a broken line) and the standard deviation in Figure 13c is relatively low. For example, the path-distance oracle for the EAS dataset with  $\epsilon=0.05$  (5%), has an observed maximum error of about 4.7%. Moreover, from Figure 13d, we see that we can answer queries in about 30 $\mu$  seconds, on the average, with an observed average error of 1.1% and a standard deviation of 1.9%.



**Figure 13: Observed a) maximum, b) average, and c) standard deviation of the errors of path-distance oracles and d) their average access times for varying values of  $\epsilon$  on the EAS dataset**

## 6. CONCLUDING REMARKS

In this paper, we introduced two linear-sized constructs termed path and path-distance oracles that represent the  $n^2$  shortest paths in a spatial network concisely. The key idea is to exploit the observed *coherence* between the spatial positions of vertices and their interconnectivity. This enabled the decomposition of a spatial network into groups of source and destination vertices, called path-coherent pairs, that share common vertices in their shortest paths. With the aid of the well-separated pair decomposition method, we constructed a path oracle that takes  $O(s^2n)$  space ( $s > 2 + \frac{2}{\delta-1}$  for  $\delta > 1$  and empirically is  $12^2n$ ), but which can retrieve an intermediate link in the shortest path between a source and a destination vertex in  $O(\log n)$  time using a B-tree. We also introduced another representation called a path-distance oracle that takes  $O(n \cdot \max(s^2, \frac{1}{\epsilon^2}))$  space (empirically  $n \cdot \max(12^2, \frac{2.5^2}{\epsilon})$ ) but which can yield an intermediate link in the shortest path as well as an  $\epsilon$ -approximation of the network distance in  $O(\log n)$  time using a B-tree. The average access time for the oracles was on the order of tens of microseconds. We can further reduce the access time of the path and path-distance oracles to  $O(1)$  using a hash table of size  $O(s^2n \log n)$  for

the path oracle and  $O(n \log n \cdot \max(s^2, \frac{1}{\epsilon^2}))$  for the path-distance oracle [12]. The result of our work is that now shortest paths can be retrieved by making repeated SQL SELECT operations on the oracle relations which are stored in a database. Even more complex query processing scenarios on spatial networks can be performed by using the initial  $\epsilon$ -approximation of the network distances, and performing subsequent *refinement* operations on the distances to improve the approximation. Such a strategy was adopted in the SILC framework [11] which is also applicable to the path and path-distance oracles. Future work includes investigating the performance of various operations, such as region search, nearest neighbor finding and distance joins, on road networks in the context of a database system using our path-distance oracles.

## 7. REFERENCES

- [1] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. of SODA*, pp. 291–300, Jan. 1993.
- [2] Z. Chen, H. T. Shen, X. Zhou, and J. Yu. Monitoring path nearest neighbor in road networks. In *Proc. of SIGMOD*, June 2009. to appear.
- [3] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to CNN queries in a road network. *Proc. of VLDB*, pp. 865–876, Sep. 2005.
- [4] J. Fischer and S. Har-Peled. Dynamic well-separated pair decomposition made easy. In *Proc. of CCCG*, pp. 235–238, Aug. 2005.
- [5] A. V. Goldberg and C. Harrelson. Computing the shortest path: A\* search meets graph theory. In *Proc. of SODA*, pp. 156–165, Jan. 2005.
- [6] N. Jing, Y.-W. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: an optimal model and its performance evaluation. *TKDE*, 10(3):409–432, May 1998.
- [7] M. R. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *Proc. of VLDB*, pp. 840–851, Sep. 2004.
- [8] H.-P. Kriegel, P. Kröger, M. Renz, and T. Schmidt. Hierarchical graph embedding for efficient query processing in very large traffic networks. In *Proc. of SSDBM*, pp. 150–167, July 2008, vol. 5069 of LNCS.
- [9] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proc. of VLDB*, pp. 802–813, Sep. 2003.
- [10] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, 2006.
- [11] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *Proc. of SIGMOD*, pp. 43–54, June 2008.
- [12] J. Sankaranarayanan. *Scalable query processing on spatial networks*. PhD thesis, UMD, College Park, MD, Apr. 2008.
- [13] J. Sankaranarayanan and H. Samet. Distance oracles for spatial networks. In *Proc. of ICDE*, pp. 652–663, Apr. 2009.
- [14] M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. of STOC*, pp. 183–192, 2001.
- [15] U.S. Census Bureau. TIGER/Line Files, Census 2000. <http://www.census.gov/geo/www/tiger/>.
- [16] U.S. Geological Survey. Major Roads of the United States. <http://nationalatlas.gov/atlasftp.html>.
- [17] D. Wagner and T. Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In *Proc. of ESA*, pp. 776–787, Sep. 2003, vol. 2832 of LNCS.