SILC: Efficient Query Processing on Spatial Networks

Hanan Samet

hjs@cs.umd.edu

Department of Computer Science

University of Maryland

College Park, MD 20742, USA

Joint work with Jagan Sankaranarayanan and Houman Alborzi

Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems (ACM GIS), November 2005, 200–209, Bremen, Germany

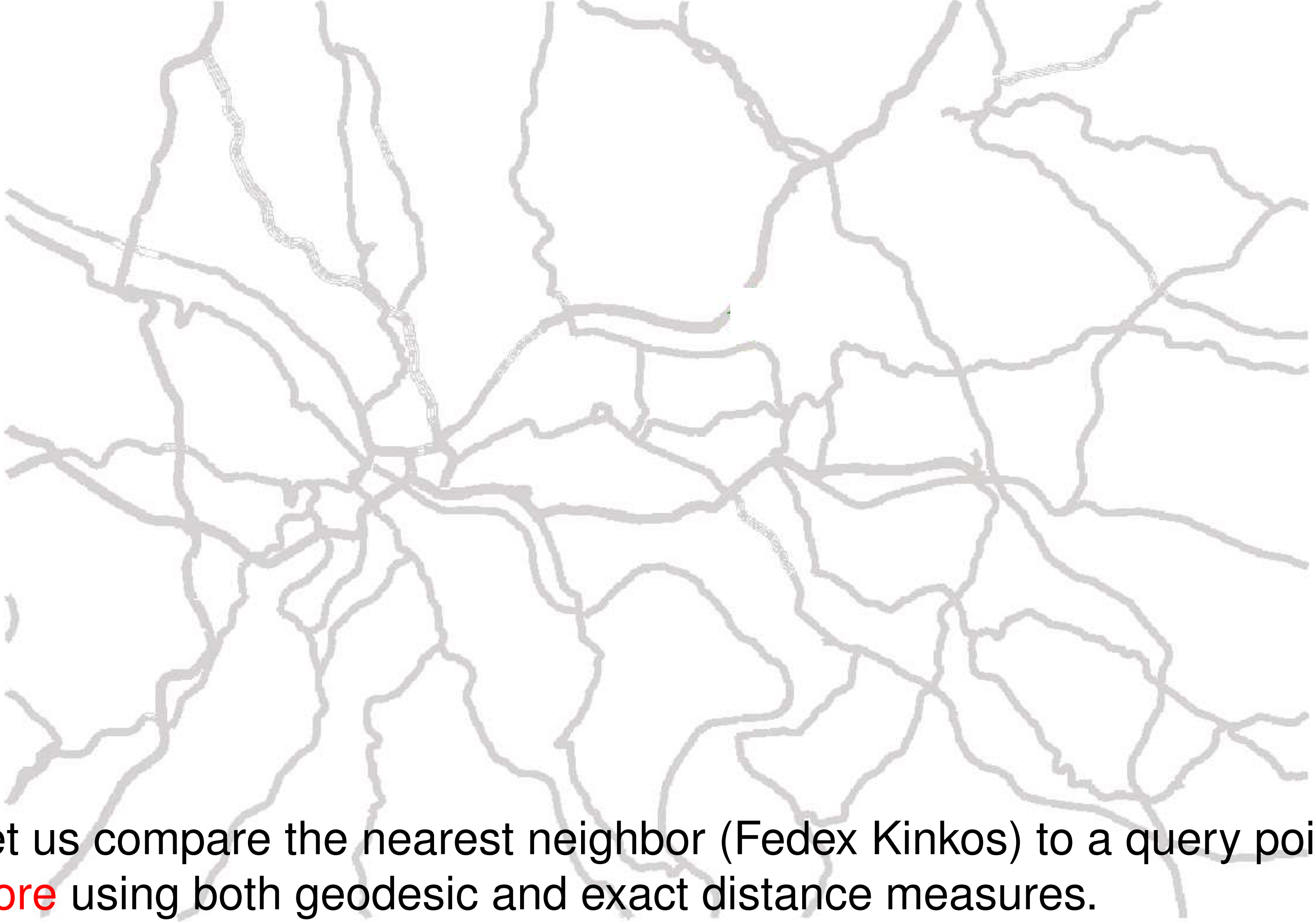
Spatial Networks

- A spatial network is a graph with spatial components at vertices and/or edges.
- Most transportation networks can be modeled as spatial networks. e.g.,
  - Road networks
    - Each intersection is a vertex of the graph, the position of the intersection is associated with the vertex.
    - Each edge of the graph corresponds to a road segment. The weight of an edge corresponds to the cost of travel (i.e., distance or time) along the corresponding road segment.
  - Airline routes
  - Waterways

Motivation

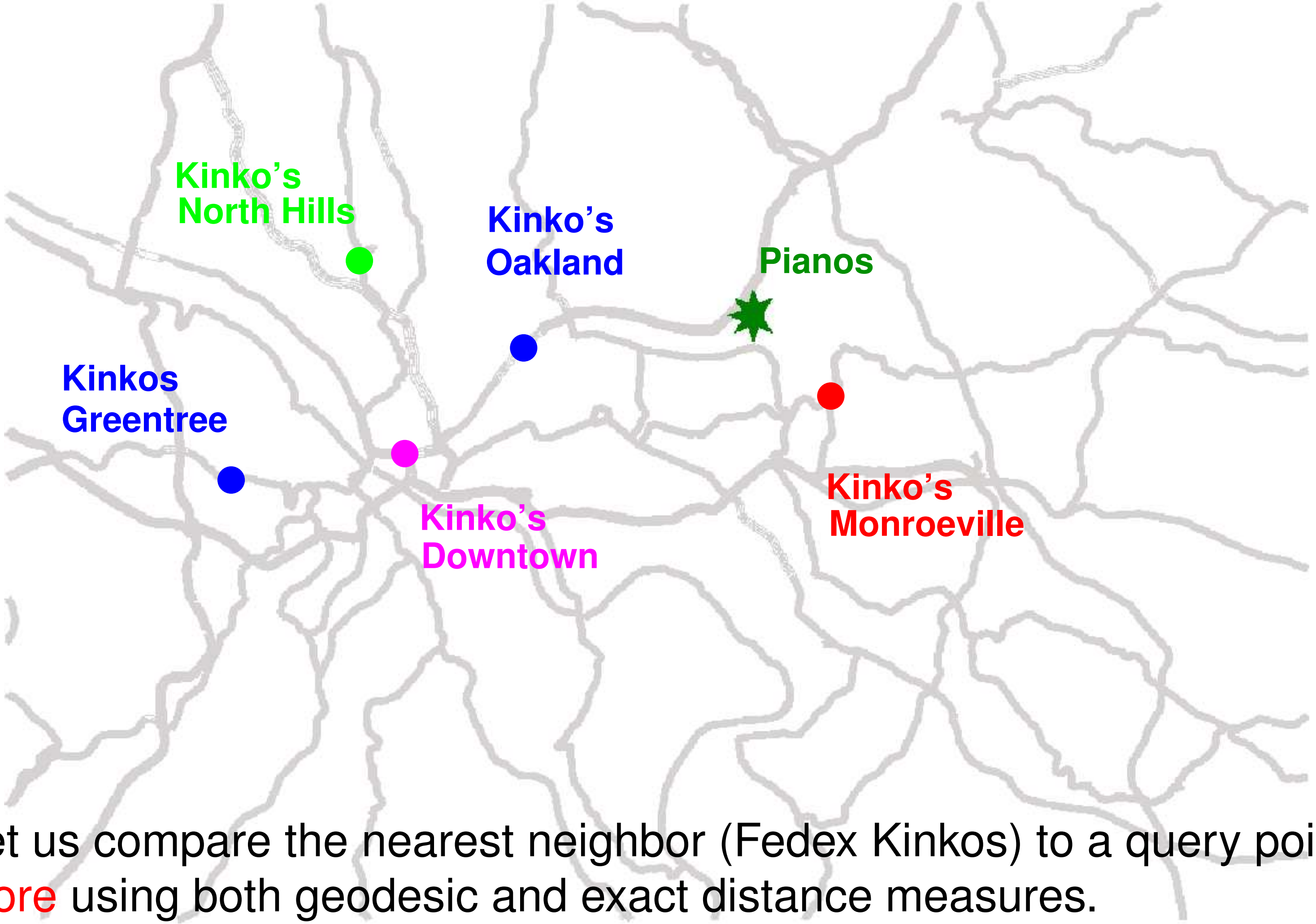
- Challenge
  - Process spatial queries on spatial networks.
  - Real-time processing.
- Applications
  - Location-based services
    - Find all the restaurants reachable within 10 minutes from AV Williams.
    - Find all the gas stations along the trip route.
  - Locational analysis
    - Find optimal location for a new store based on demography.
    - Find optimal location for a new warehouse based on customer locations.
  - Trip planning

Application – Find the closest Kinko's



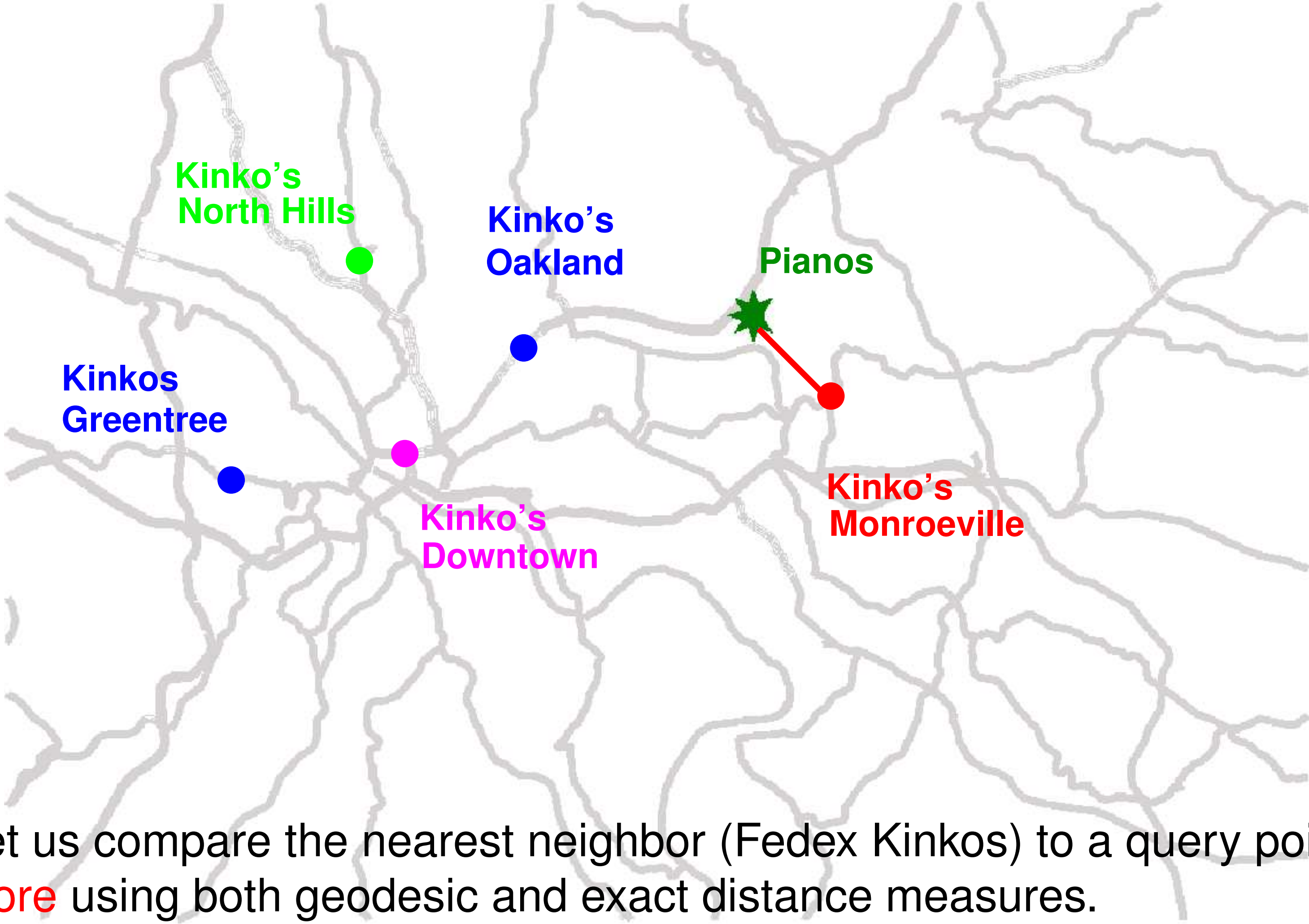
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.

Application – Find the closest Kinko's



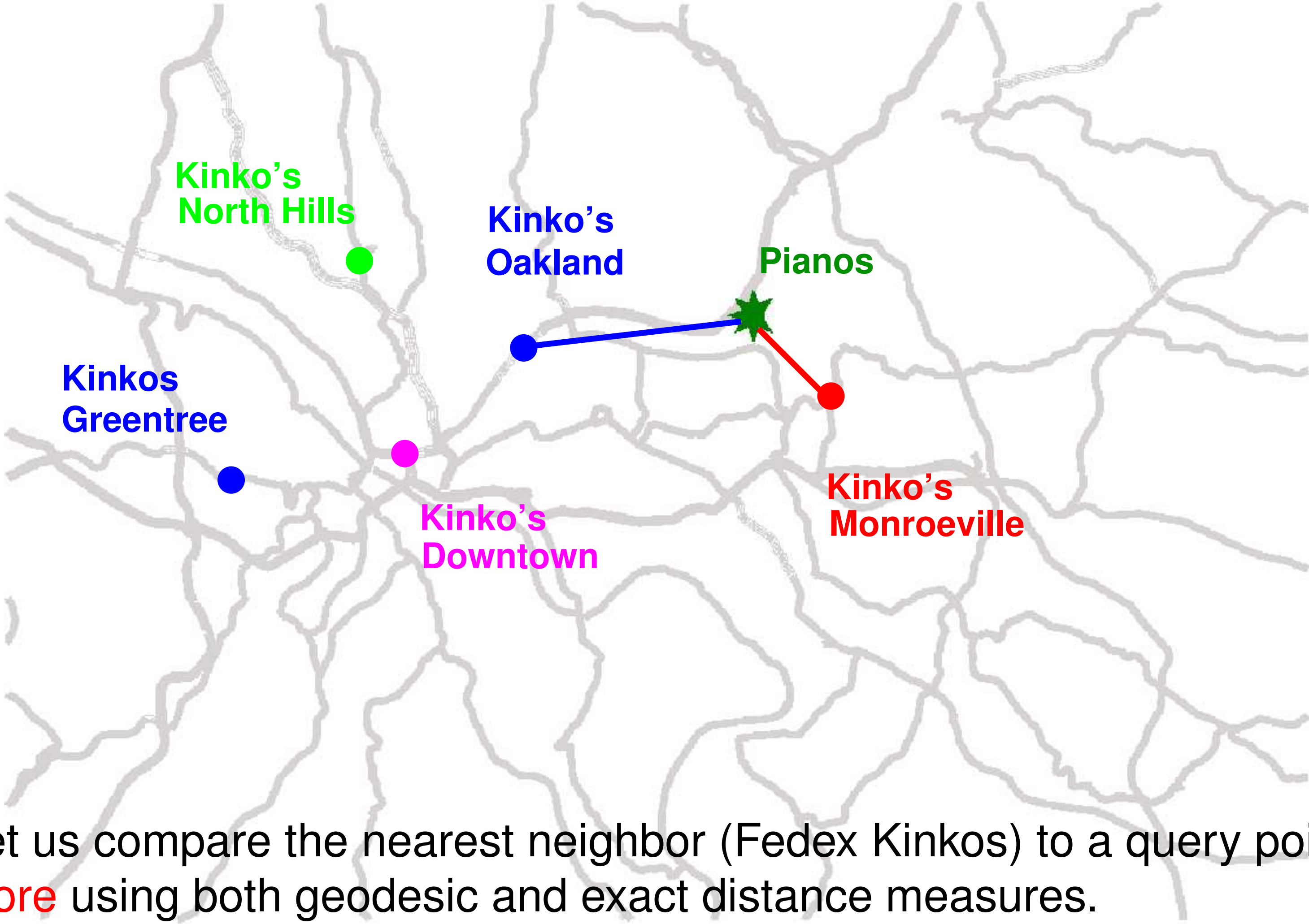
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.

Application – Find the closest Kinko's



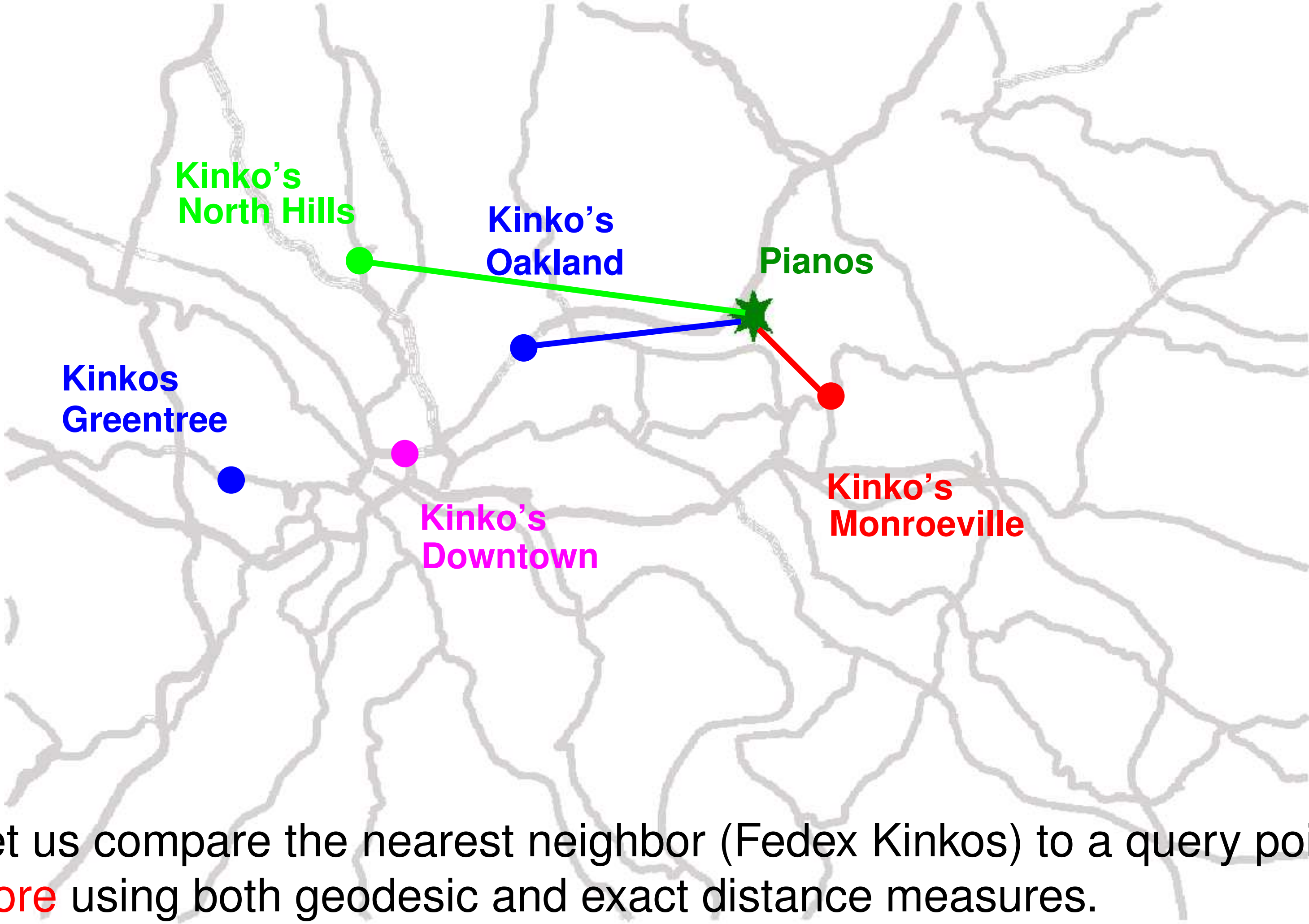
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M

Application – Find the closest Kinko's



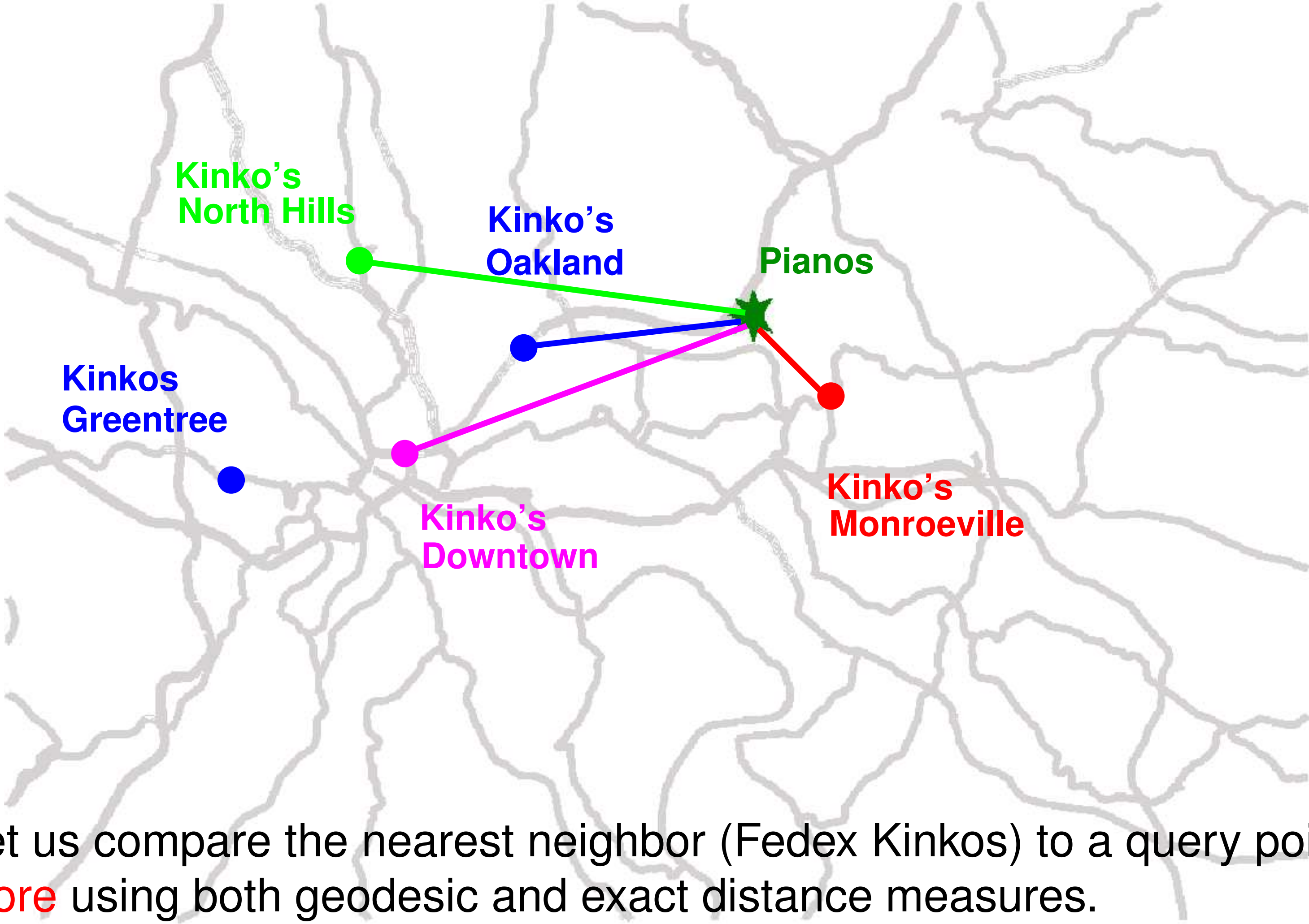
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O

Application – Find the closest Kinko's



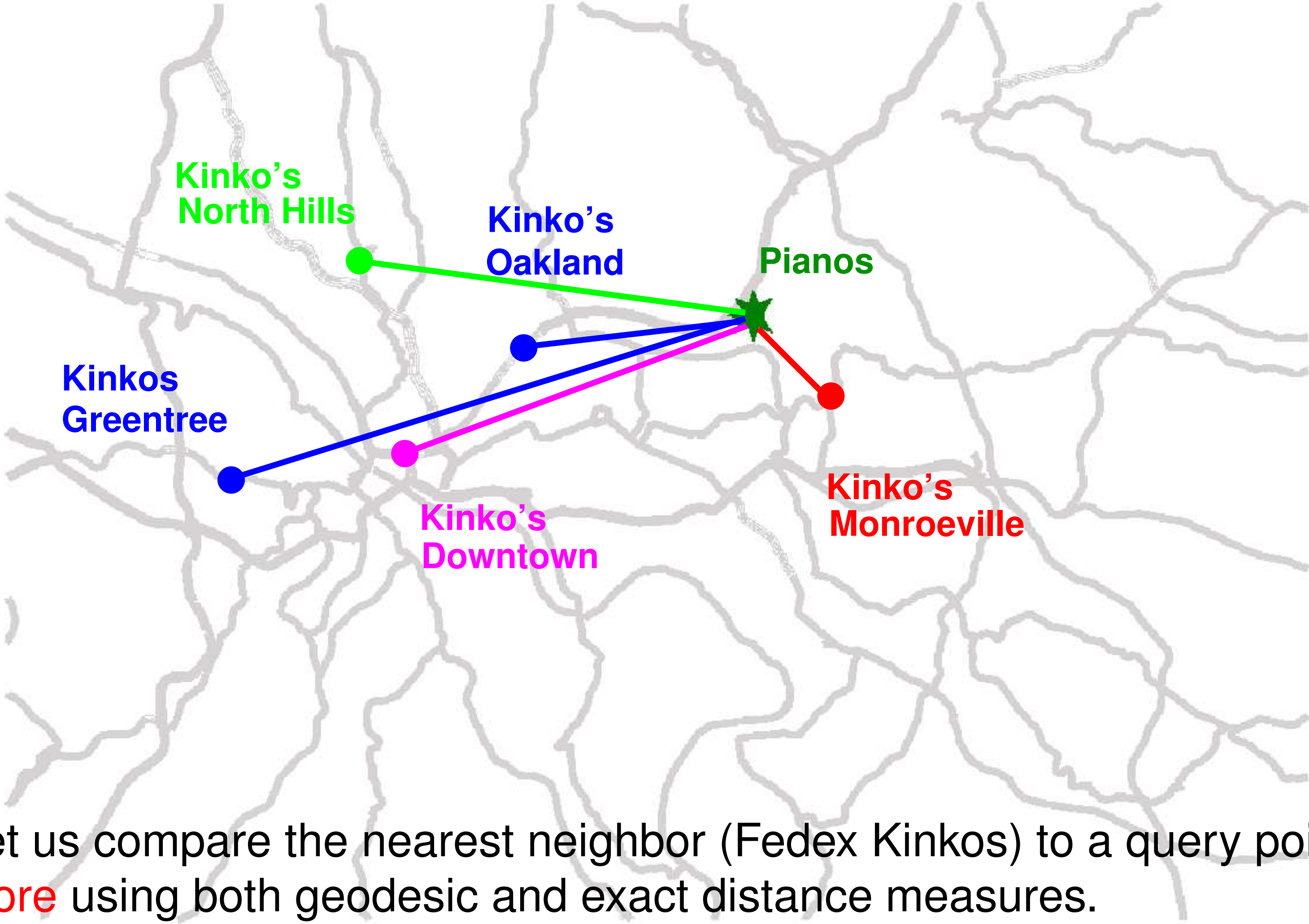
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N

Application – Find the closest Kinko's



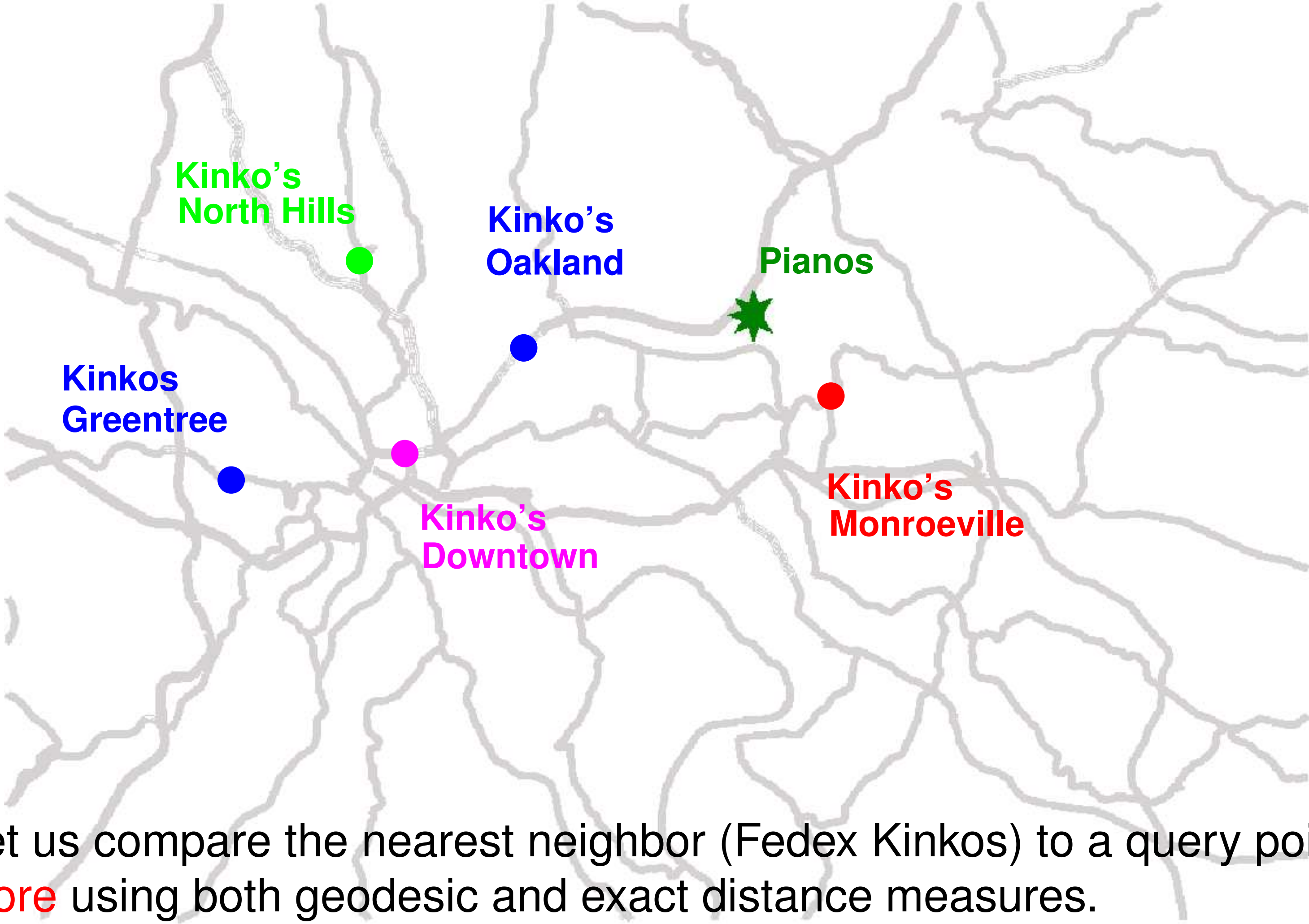
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D

Application – Find the closest Kinko's



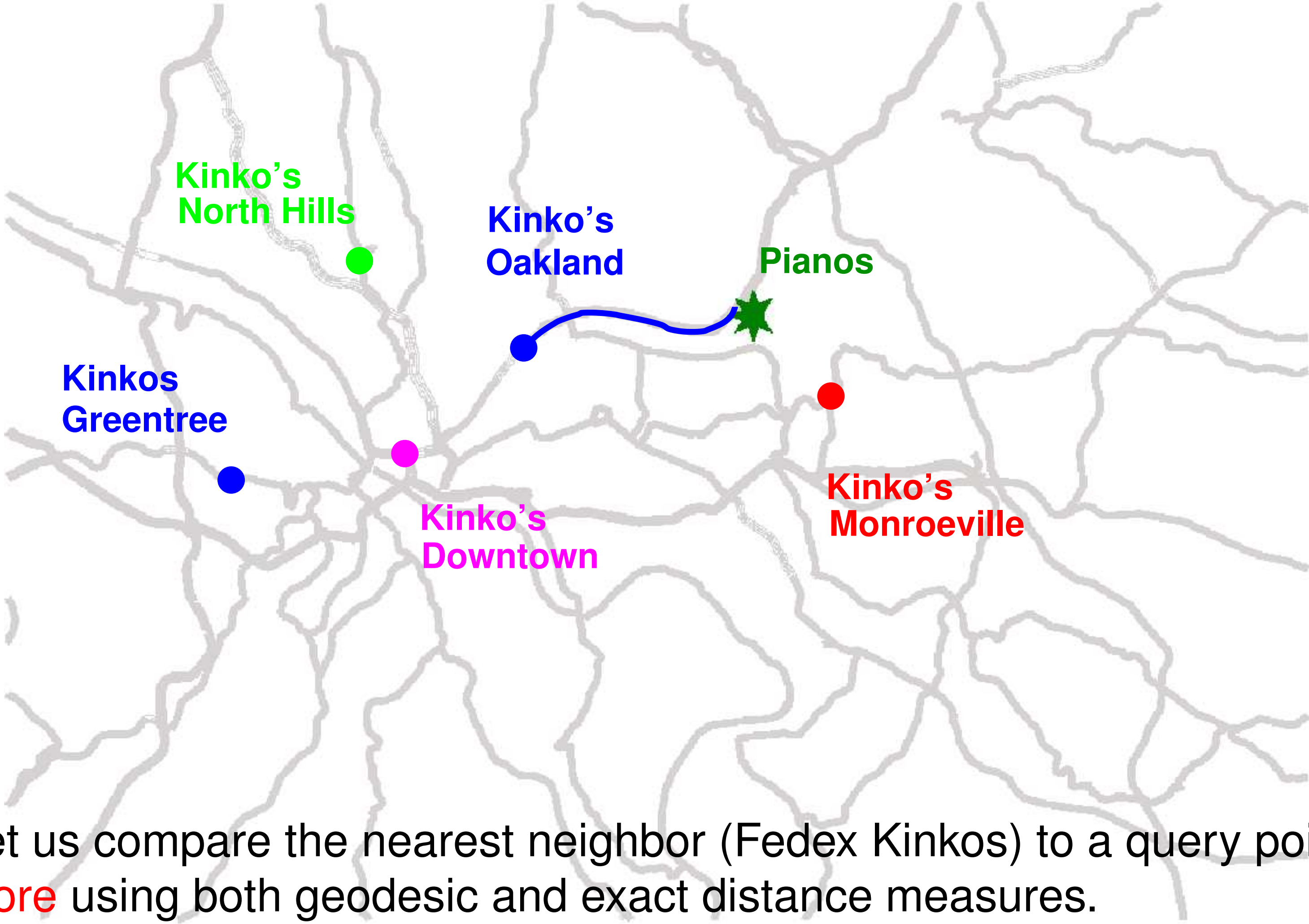
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G

Application – Find the closest Kinko's



- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering

Application – Find the closest Kinko's

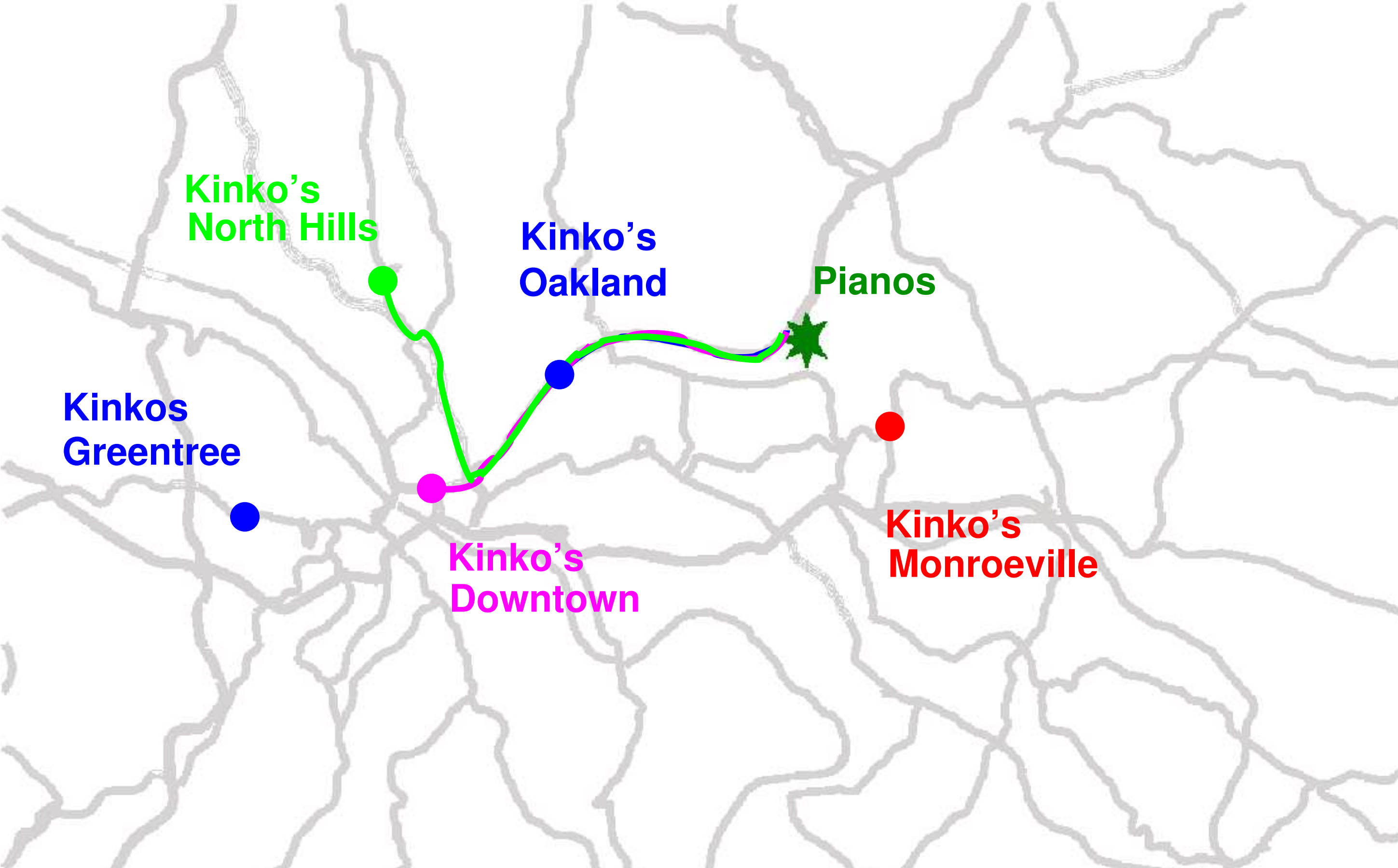


- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O

Application – Find the closest Kinko's

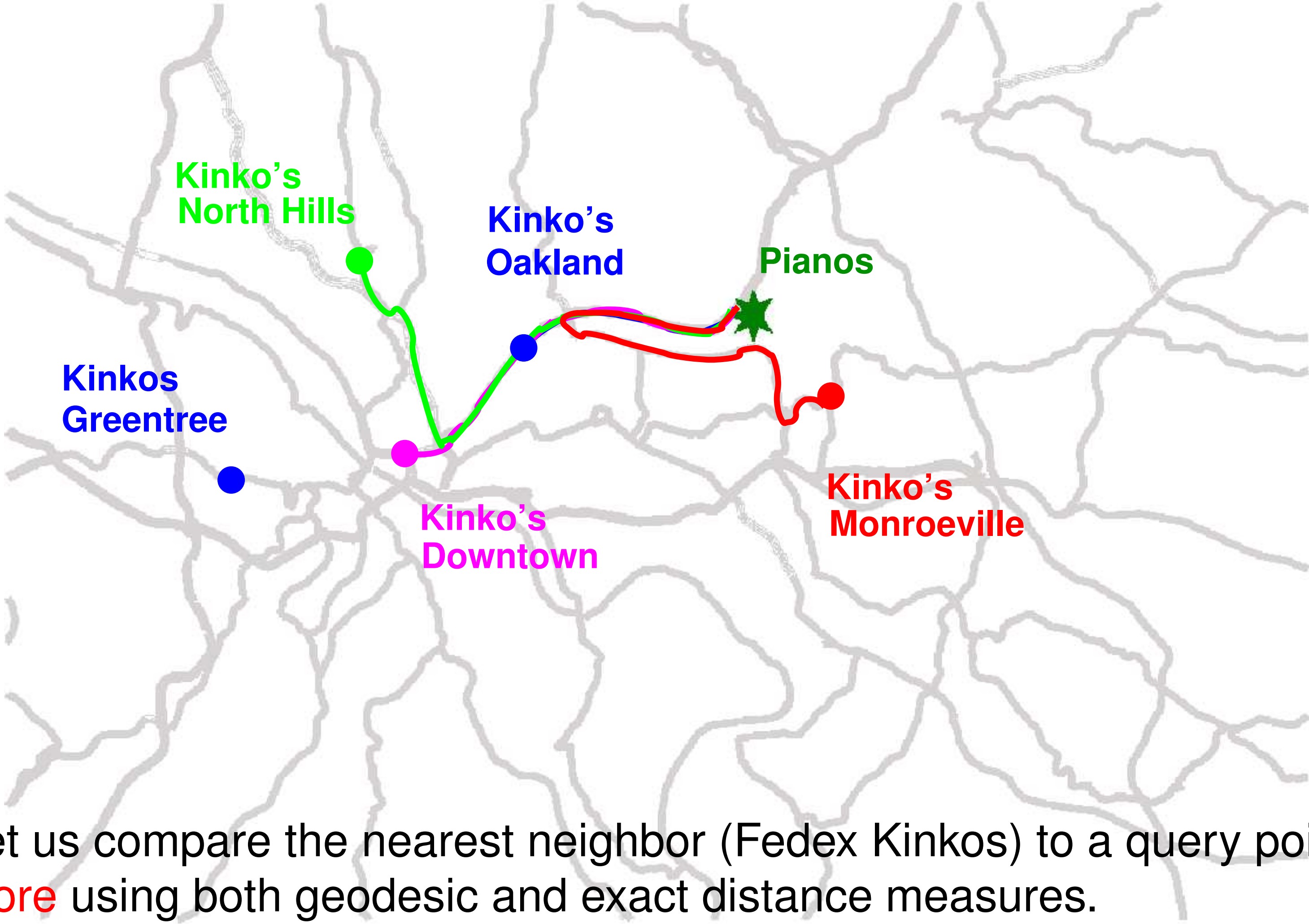
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D

Application – Find the closest Kinko's



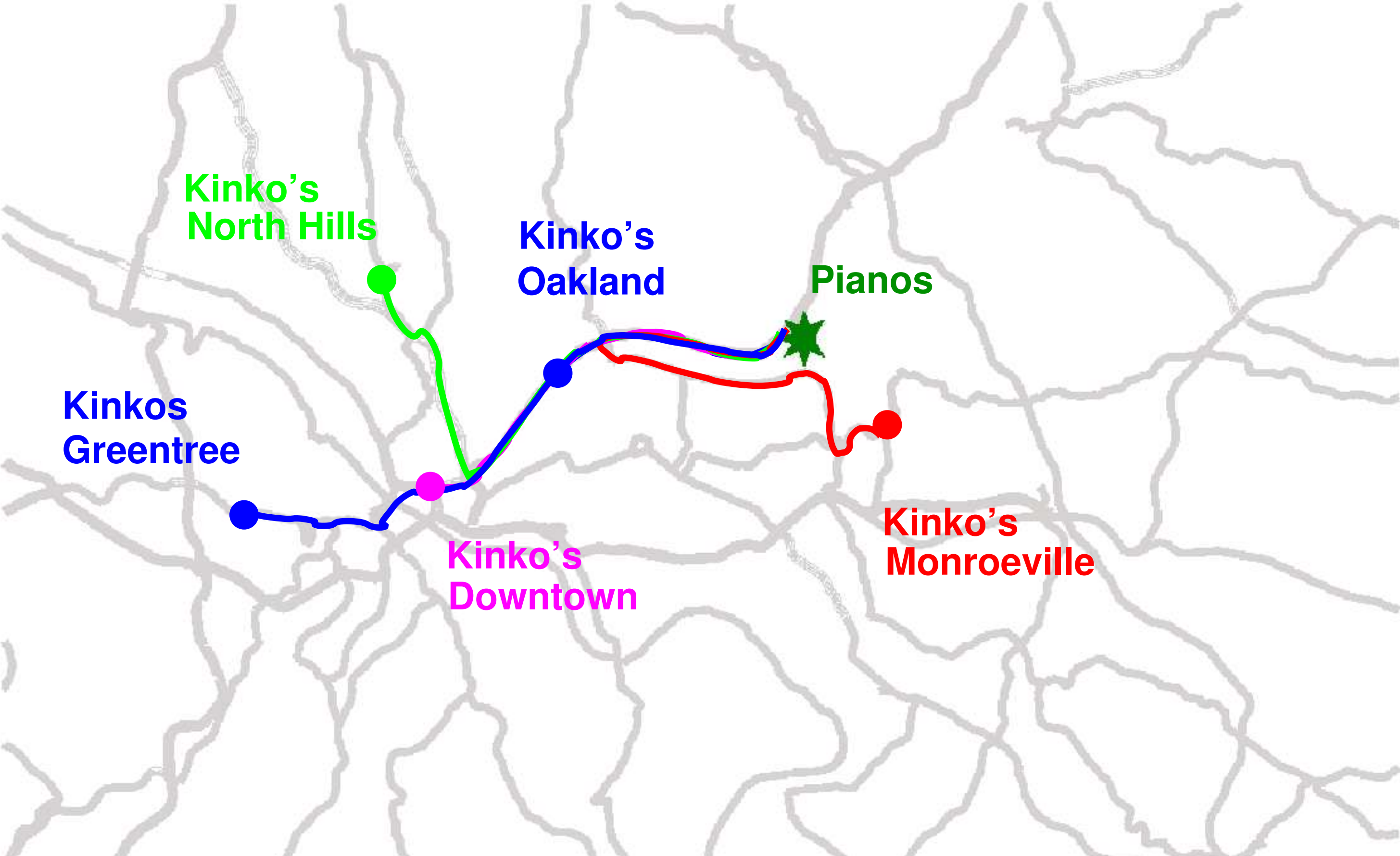
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N

Application – Find the closest Kinko's



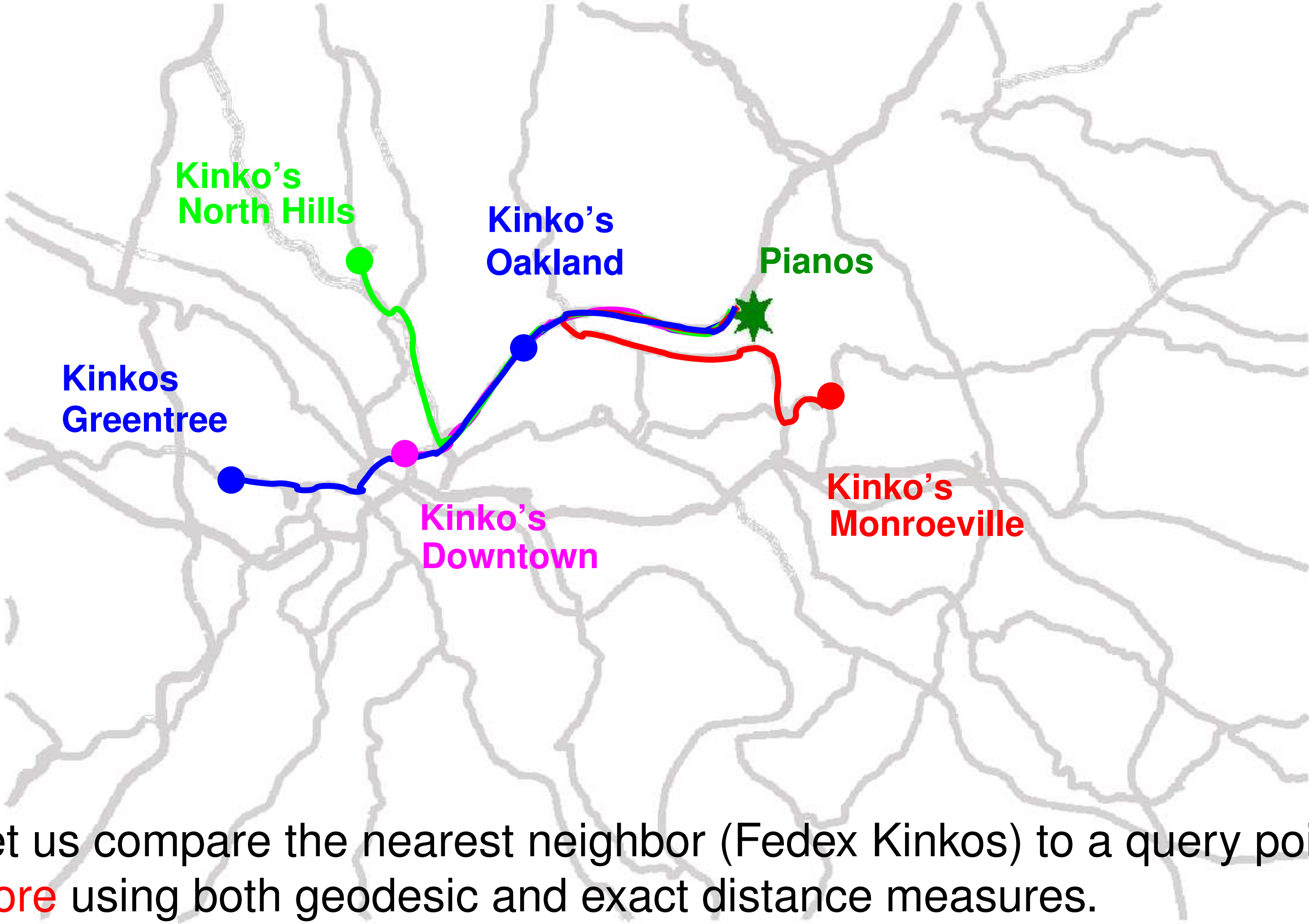
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M

Application – Find the closest Kinko's



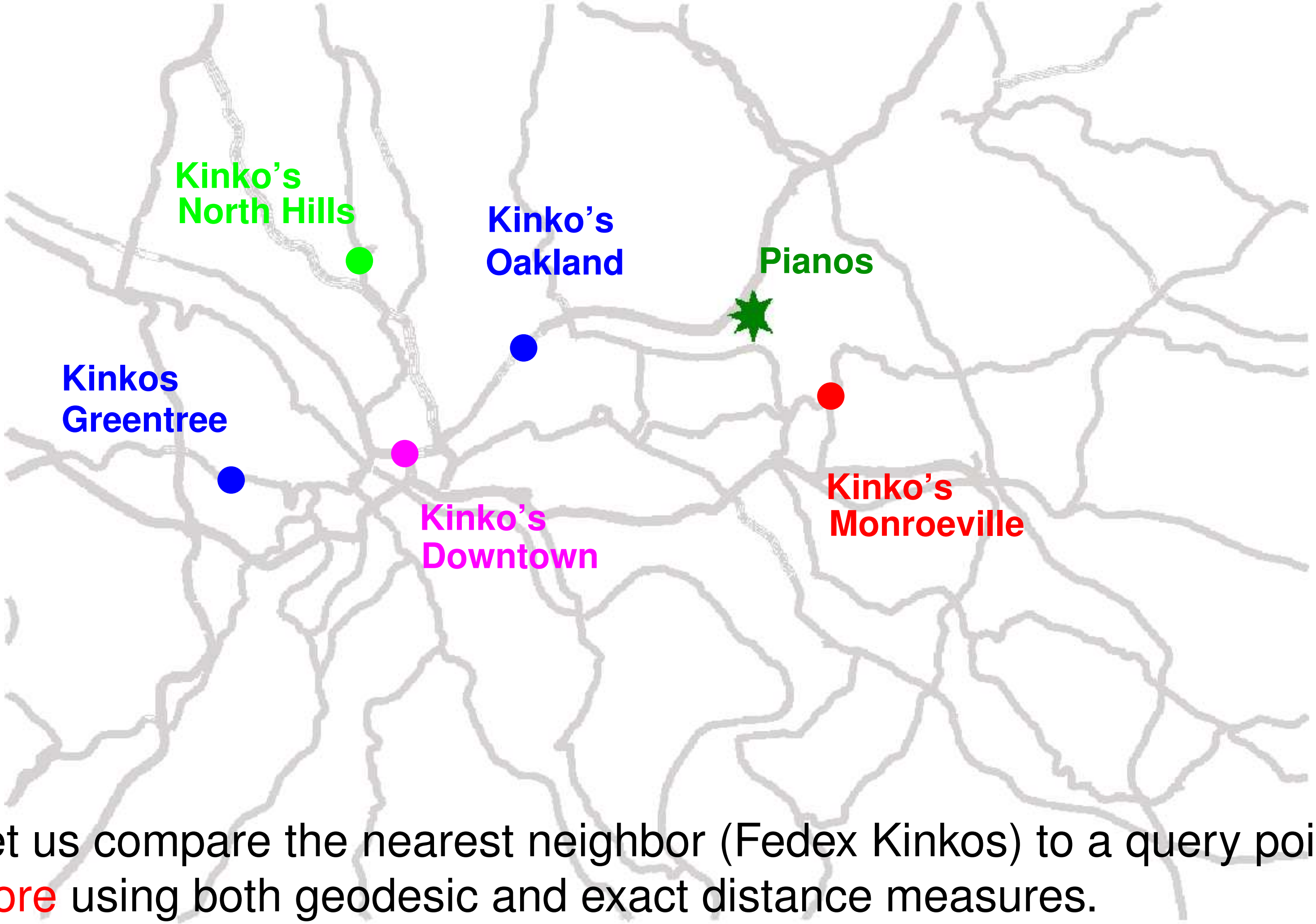
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G

Application – Find the closest Kinko's



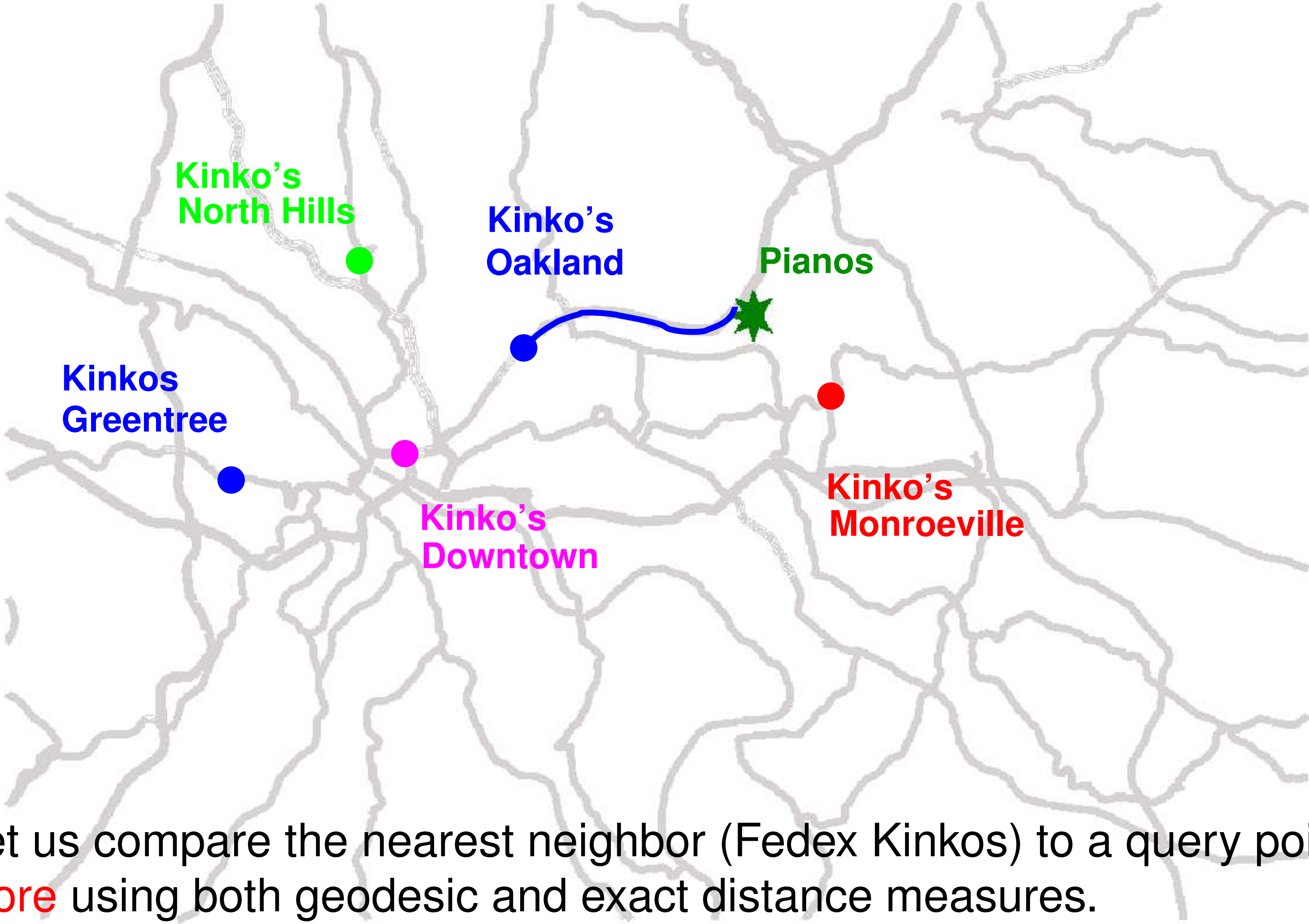
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)

Application – Find the closest Kinko's



- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)
  - round-trip time ordering

Application – Find the closest Kinko's



- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)
  - round-trip time ordering O

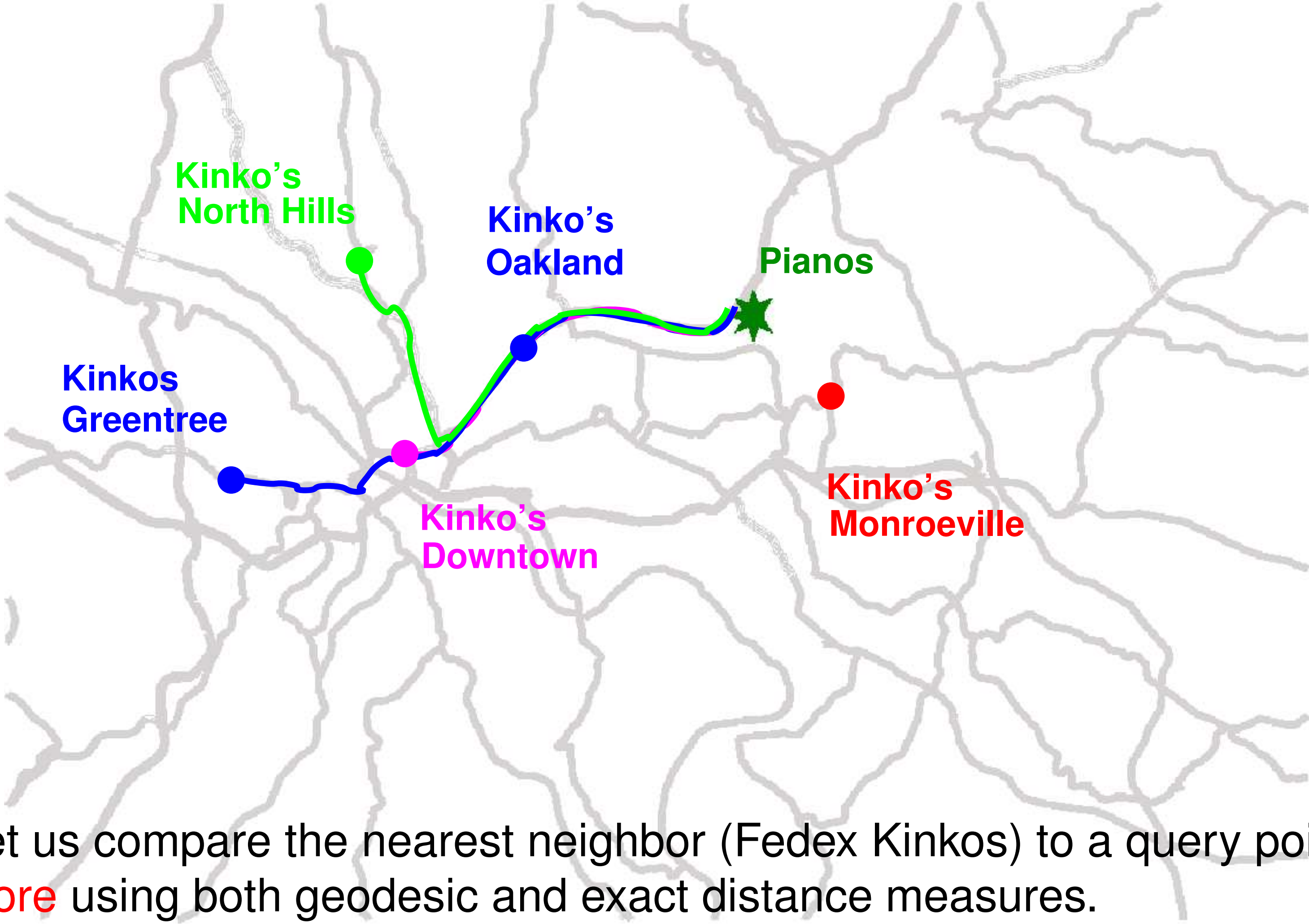
Application – Find the closest Kinko's

- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)
  - round-trip time ordering O D

Application – Find the closest Kinko's

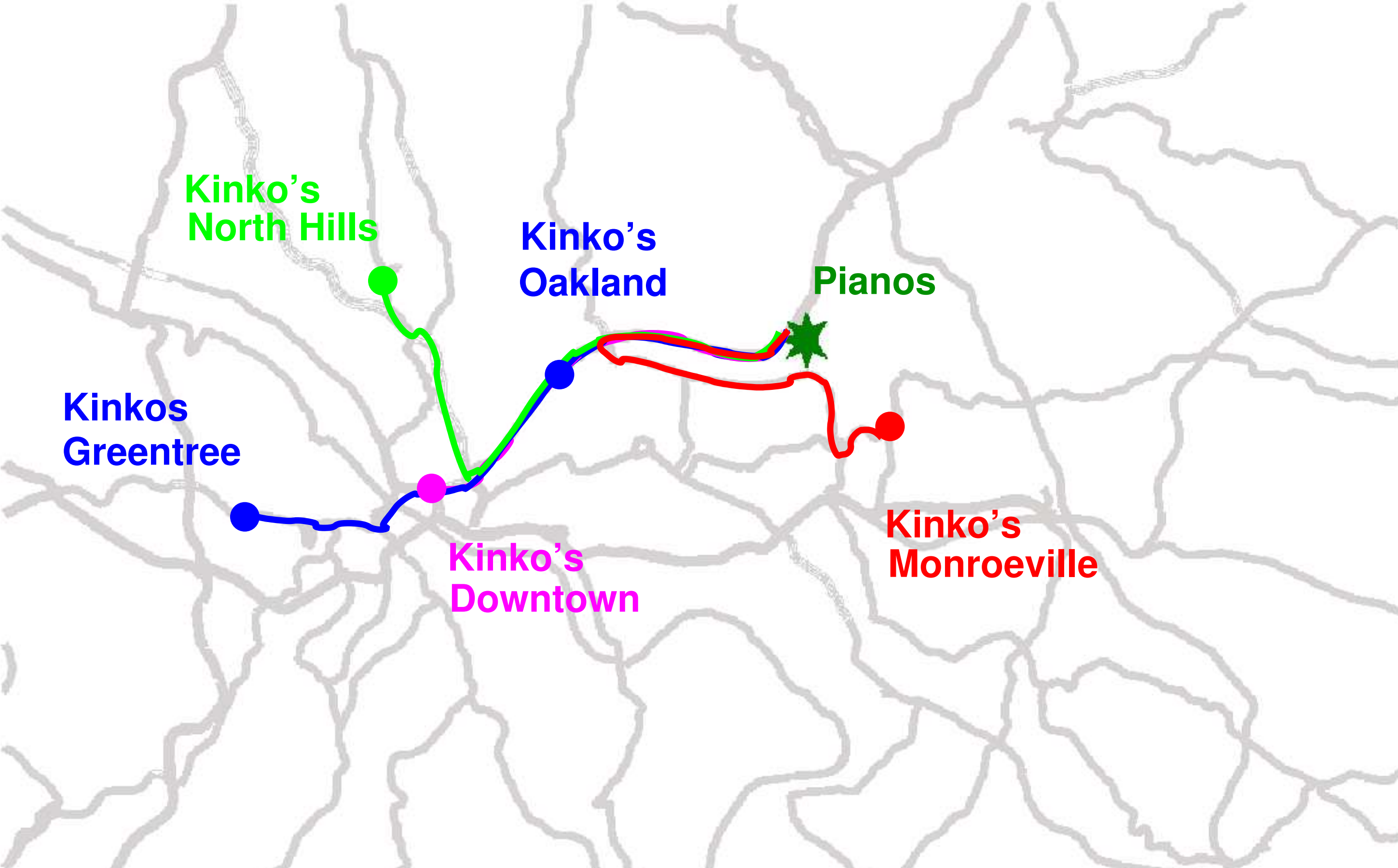
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)
  - round-trip time ordering O D G

Application – Find the closest Kinko’s



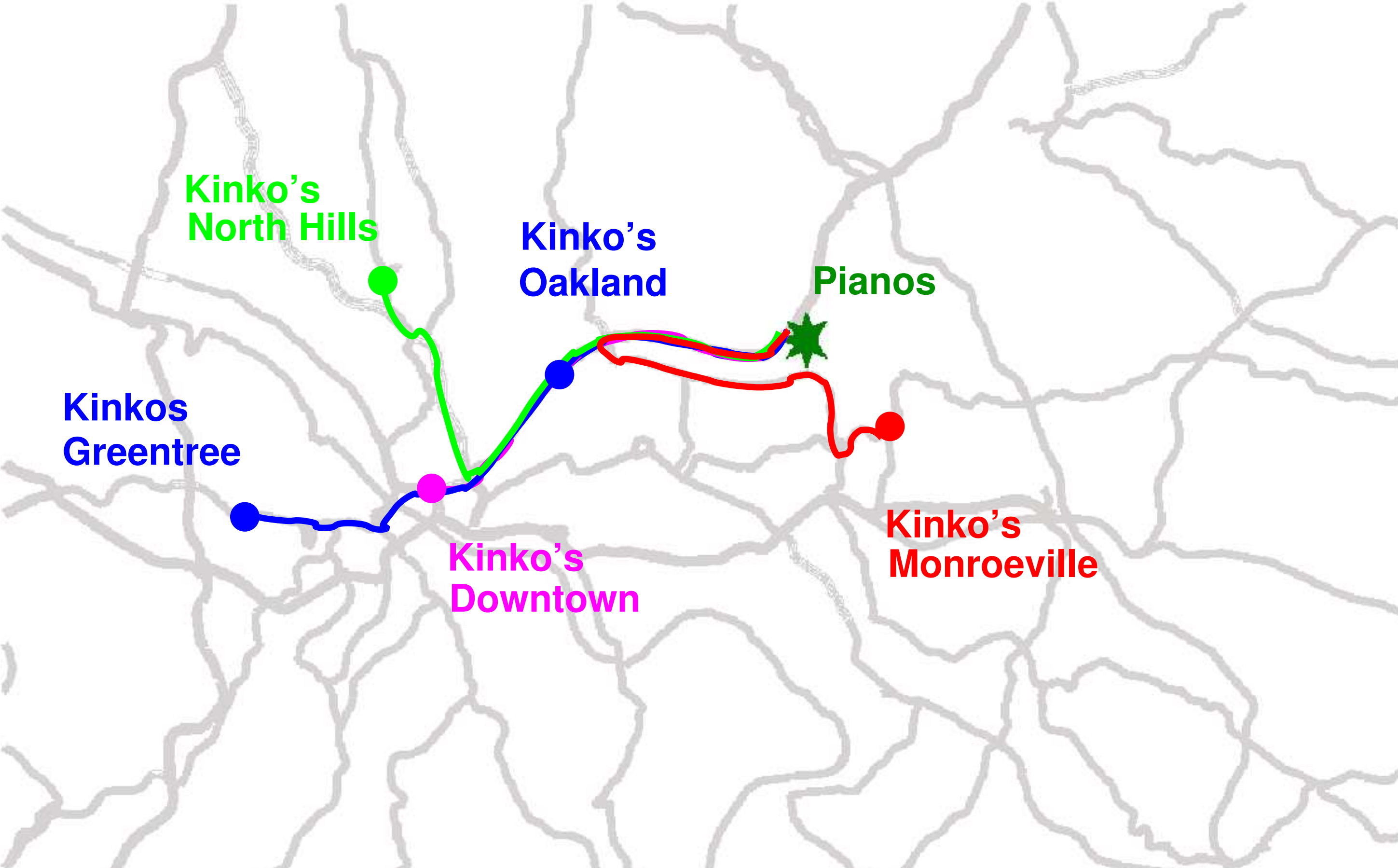
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)
  - round-trip time ordering O D G N

Application – Find the closest Kinko's



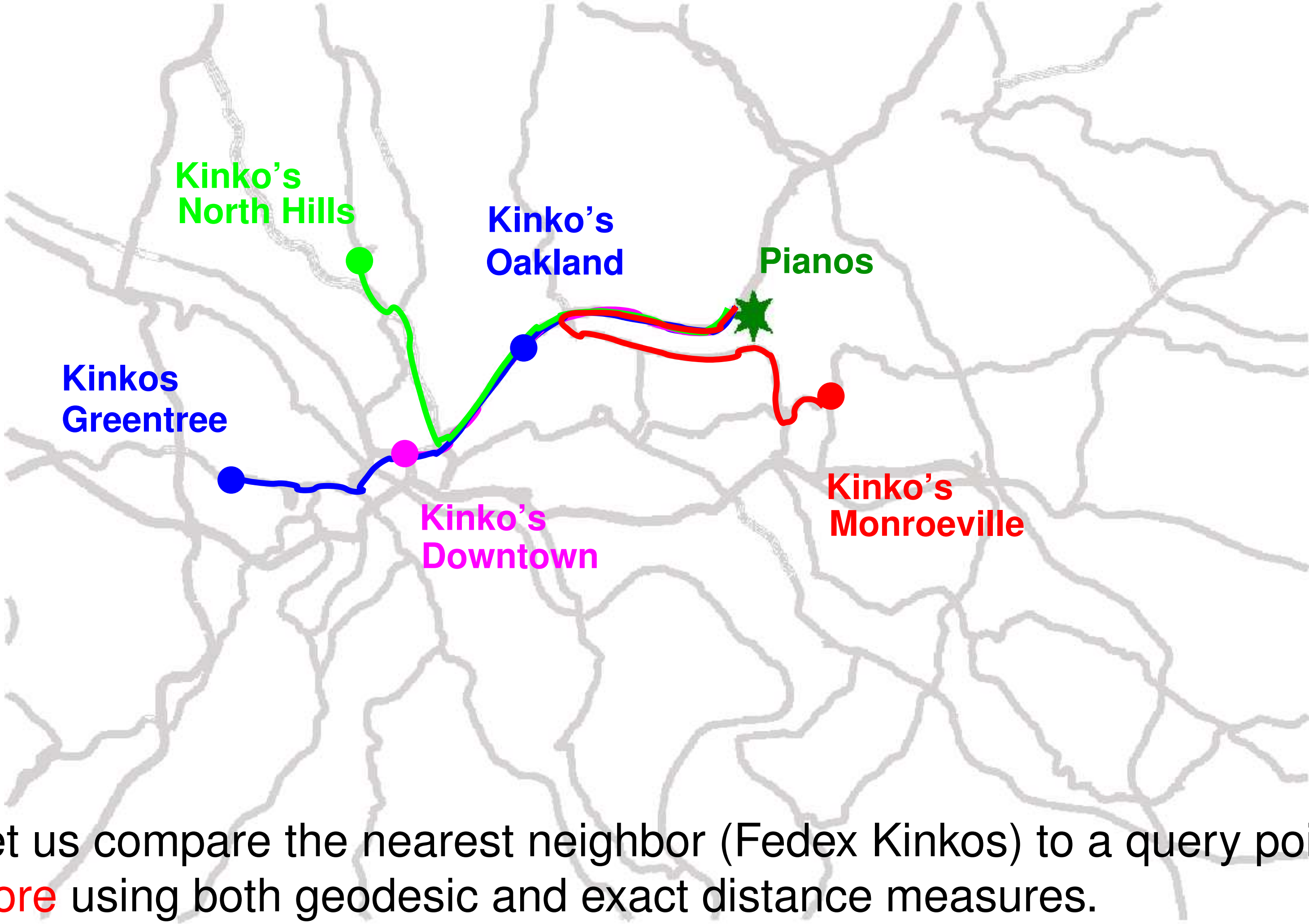
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)
  - round-trip time ordering O D G N M

Application – Find the closest Kinko's



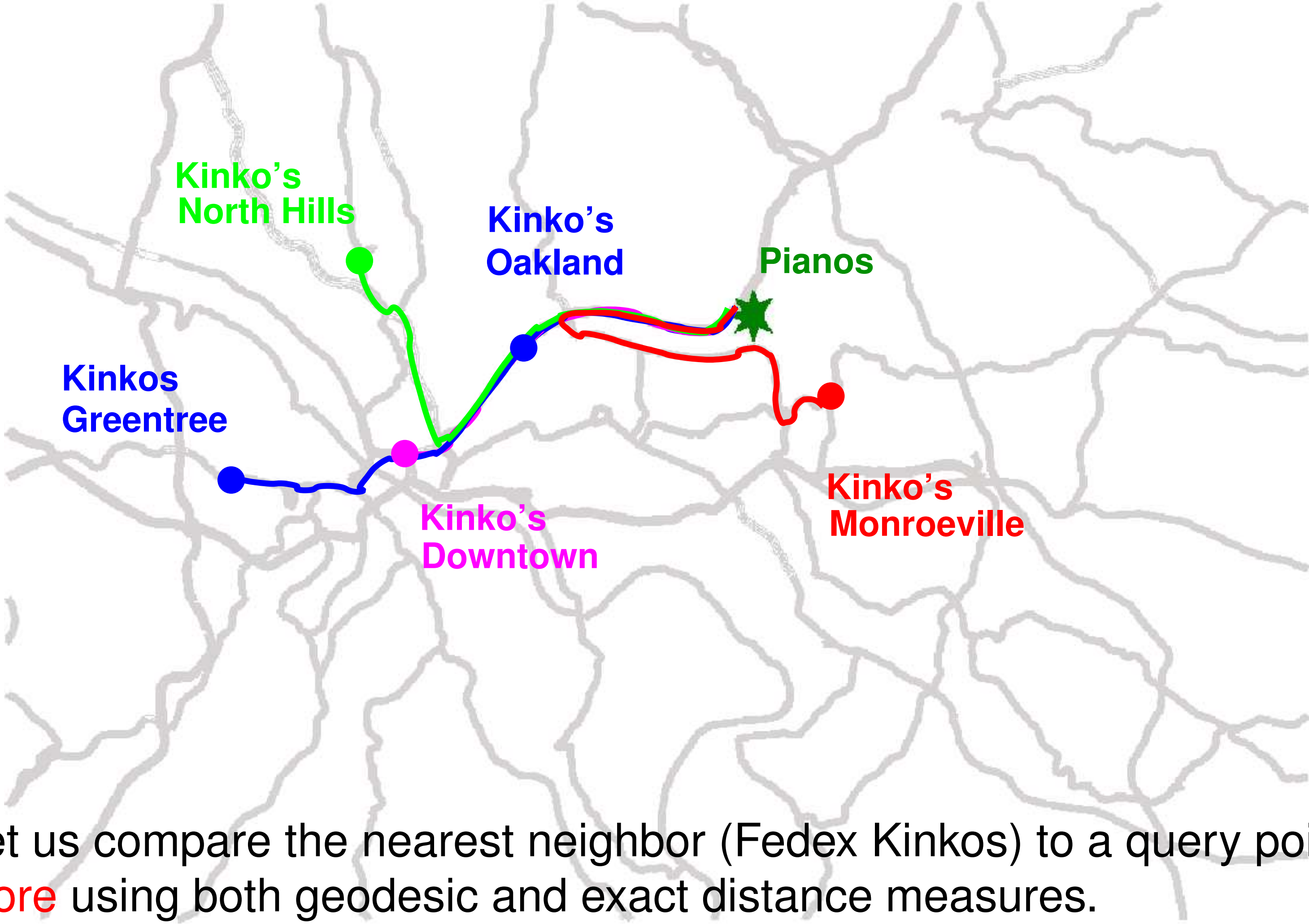
- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)
  - round-trip time ordering O D G N M (Error: +32 minutes)

Application – Find the closest Kinko's



- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)
  - round-trip time ordering O D G N M (Error: +32 minutes)
    - variant of the "traveling salesman" problem.

Application – Find the closest Kinko's



- Let us compare the nearest neighbor (Fedex Kinkos) to a query point Piano store using both geodesic and exact distance measures.
  - geodesic ordering M O N D G
  - network distance ordering O D N M G (Error: +26 miles)
  - round-trip time ordering O D G N M (Error: +32 minutes)
    - variant of the "traveling salesman" problem.
- Challenge: Real time + exact queries

Contributions of our work

The SILC framework:

- Query processing on spatial networks using existing spatial database techniques – nearest neighbor queries, range queries, and spatial joins.
- Real time processing of both approximate and exact spatial queries on spatial networks.
- We precompute and store the shortest path between all pairs of vertices in the spatial network. Storage made feasible using path coherence.
- We introduce the general concept of *progressive refinement* of distances.

Spatial Network Queries

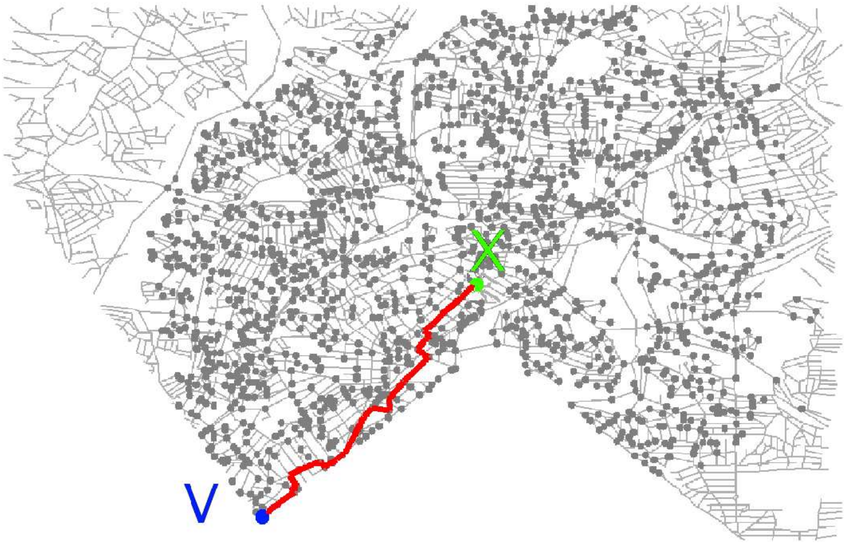
- Types of queries desired, with examples from an emergency response scenario.
- Path and distance queries: Compute the shortest path and distance between two locations on a spatial network.
  - e.g., Find the distance and the path from the accident scene to the hospital.
- Range queries: Find all locations that exist within a distance of r from a specified query point.
  - e.g., Find all hospitals that are within one mile road distance – or equally, can be reached within five minutes of driving – from the accident scene.
- Incremental nearest neighbors: Incrementally retrieve the nearest neighbors to a query point.
  - e.g., Find the nearest hospitals to the accident scene in the increasing order of the trip time.

Spatial Network Queries (Contd.)

- Types of joins desired, with location analysis scenarios.
- Distance join: Given two sets of spatial objects, S and R, incrementally retrieve the closest pair of objects.
  - e.g., Given a set of stores and another set of warehouses, incrementally retrieve the closest pair containing a store and a warehouse, in the increasing order of the trip time.
- Distance semi-join: Distance semi-join requires that objects from S appear only once in the output.
  - The distance semi-join finds the closest warehouse to each store.

Shortest Path Computation

- Shortest path computation is a primitive operation.
- However, it is not feasible in real time computation for large spatial networks.
- Dijkstra's algorithm visits too many vertices during the search process.
- In the example shown alongside, the Dijkstra's algorithm visits 3191 out of a total of 4233 vertices in the spatial network to identify a path comprising 75 vertices between X and V.



- Popular solution: Use "crow flying" (geodesic) distance.

Observation

- By precomputing and storing all the shortest paths, shortest path queries could be answered instantly.
  - How to effectively compute the shortest path ?
  - How to effectively store the shortest paths ?
  - Challenge: Very large network, (24,000,000 vertices).
- Intuition: Path Coherence – Vertices that are spatially close to each other share large portions of their shortest path to far destinations.
- Example:
  - Neighborhood commuters use (congest !) the same roads when traveling to nearby offices.
- Most spatial networks exhibit path coherence.
- Idea: Use path coherence to store and retrieve all shortest paths efficiently.

Path and Distance Encoding: Standard approach

- Path and distance encoding: Storing all pair shortest path and distance.
- Trade-off: Space requirement of encoding shortest paths vs. retrieval time.

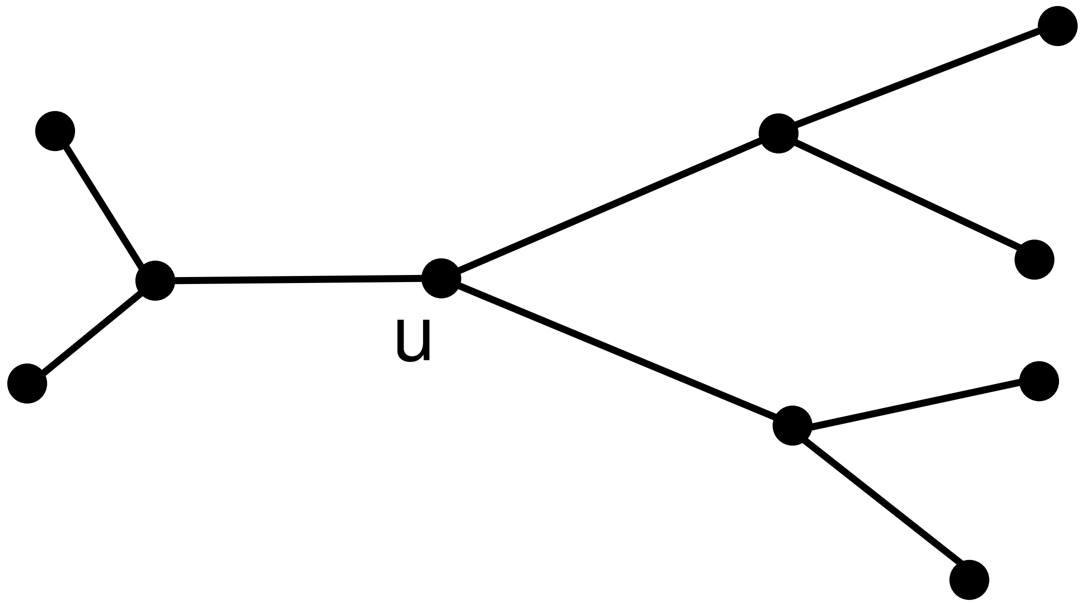
k= length of the shortest path

Approach	Space	Query Time	Query Time
		Path	Distance
Explicit path storage	$O(n^3)$	$O(1)$	$O(1)$
Next-hop storage	$O(n^2)$	$O(k)$	$O(k)$
Dijkstra's	$O(n)$	$O(m \log n)$	$O(m \log n)$
SILC	$O(n \log n)$	$O(k \log n)$	$O(k \log n)$

- Approximate path and distance encoding [Shah03] is not the focus of this work.

SILC Path Encoding

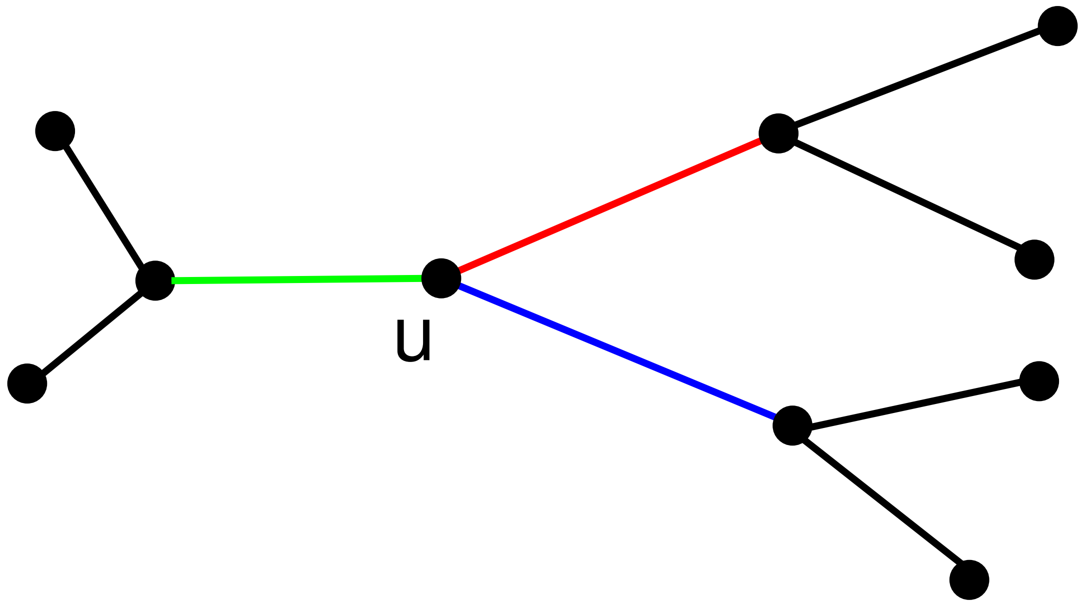
- The SILC path encoding takes advantage of the path coherence
- Assign *colors* to vertices based on first edge in path.



- Source vertex u in a spatial network.

SILC Path Encoding

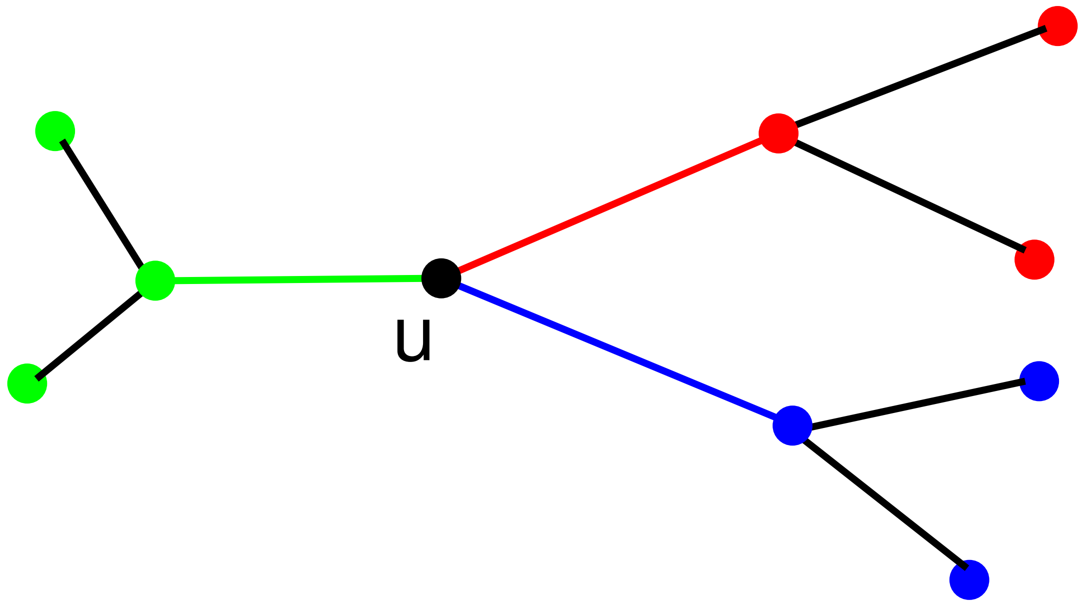
- The SILC path encoding takes advantage of the path coherence
- Assign *colors* to vertices based on first edge in path.



- Source vertex u in a spatial network.
- Assign colors to the outgoing edges of u.

SILC Path Encoding

- The SILC path encoding takes advantage of the path coherence
- Assign *colors* to vertices based on first edge in path.



- Source vertex u in a spatial network.
- Assign colors to the outgoing edges of u.
- Color vertex based on their first edge in the shortest path to u.

SILC Path Encoding (Contd.)

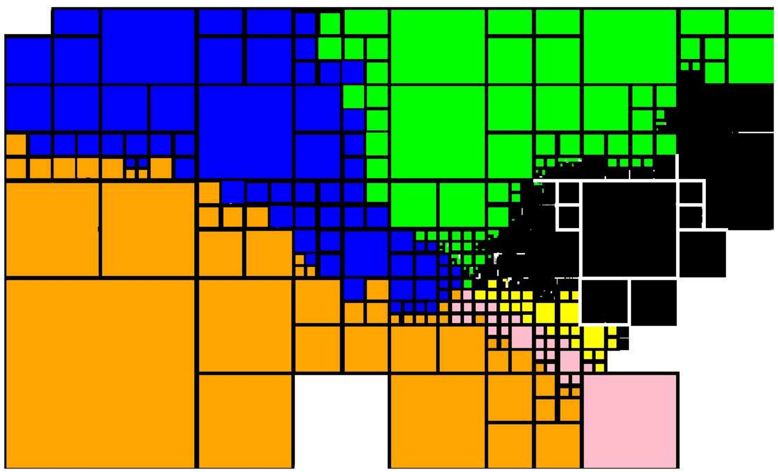
- Coloring results in spatially contiguous colored regions.
- Spatial contiguity of colored regions arises from path coherence.

a. Source vertex u highlighted in red.

b. Remaining vertices are assigned colors based on their shortest path from u through one of the six adjacent vertices of u.

Storing Colored regions

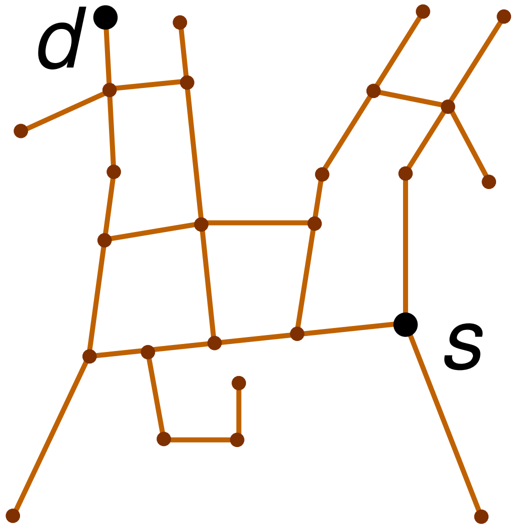
- Minimum bounding boxes [Wagn03]
- Disjoint decomposition: region quadtree.
- A disjoint decomposition is preferable as overlaps are avoided.
- Region quadtree stored as a collection of Morton blocks.
- Proposed encoding strategy leverages the dimensionality reduction property of quadtrees.
  - The required storage cost to represent a region R in a region quadtree is $O(p)$, where p is the perimeter of R.

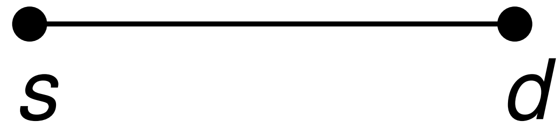


Quadtree corresponding to the regions formed due to the coloring operation.

Path Retrieval

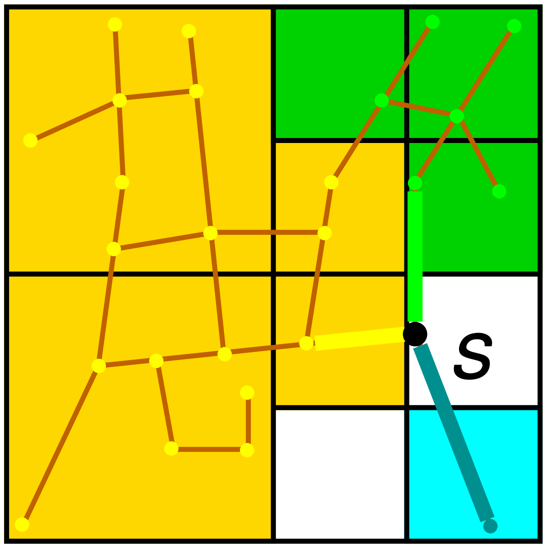
- Problem: How to retrieve the shortest path between source s and destination d.





Path Retrieval

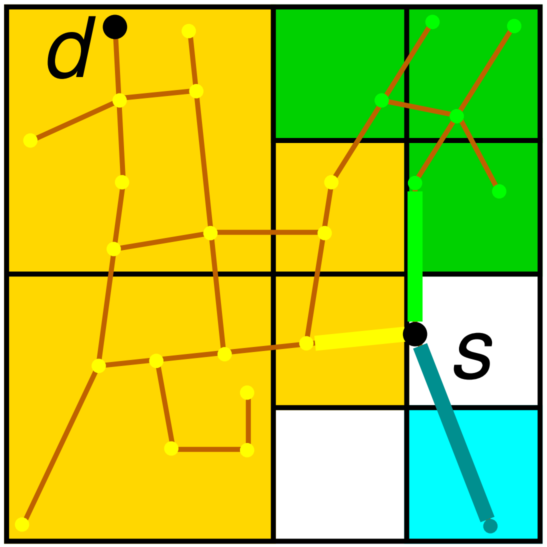
- Problem: How to retrieve the shortest path between source s and destination d.

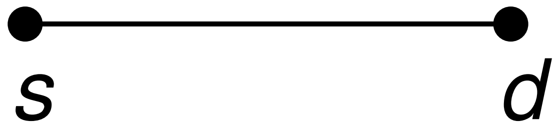


  - Retrieve the quadtree Q_s corresponding to s.

Path Retrieval

- Problem: How to retrieve the shortest path between source s and destination d.

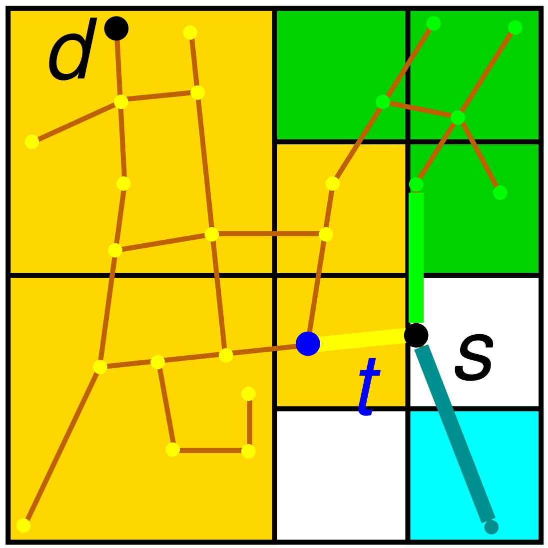


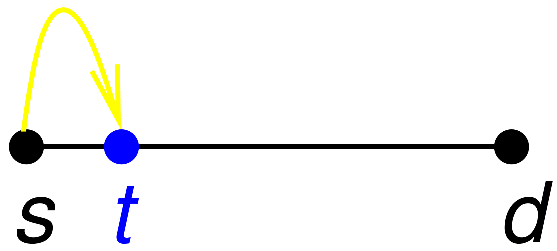


- Retrieve the quadtree Q_s corresponding to s.
- Find the colored region that contains d in Q_s.

Path Retrieval

- Problem: How to retrieve the shortest path between source s and destination d.

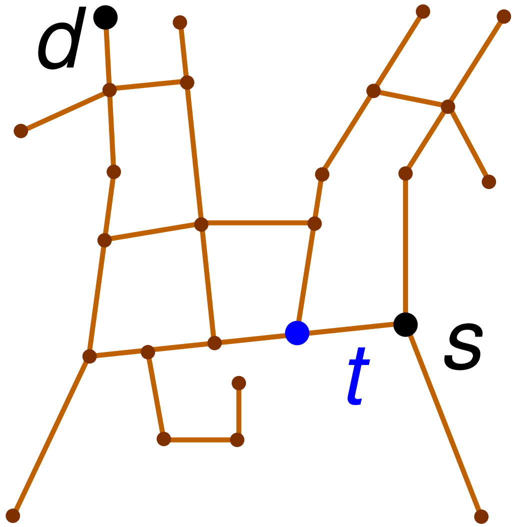


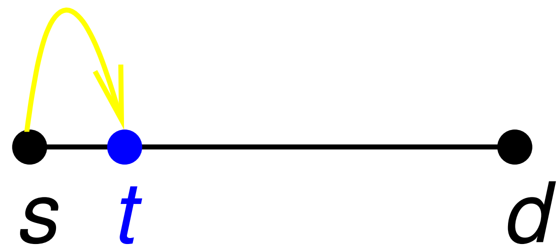


  - Retrieve the quadtree Q_s corresponding to s.
  - Find the colored region that contains d in Q_s.
  - Retrieve the vertex t connected to s in the region containing d in Q_s.

Path Retrieval

- Problem: How to retrieve the shortest path between source s and destination d.

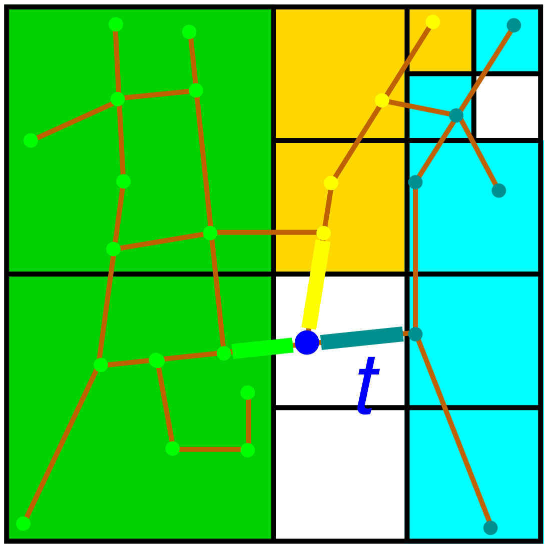


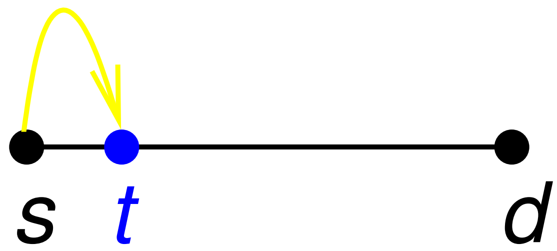


- Retrieve the quadtree Q_s corresponding to s.
- Find the colored region that contains d in Q_s.
- Retrieve the vertex t connected to s in the region containing d in Q_s.

Path Retrieval

- Problem: How to retrieve the shortest path between source s and destination d.

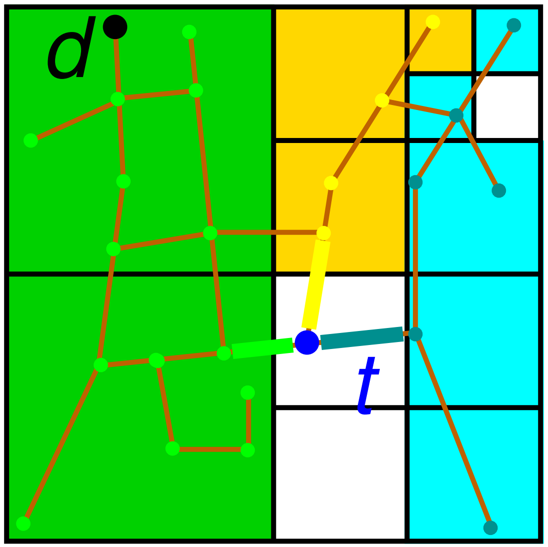


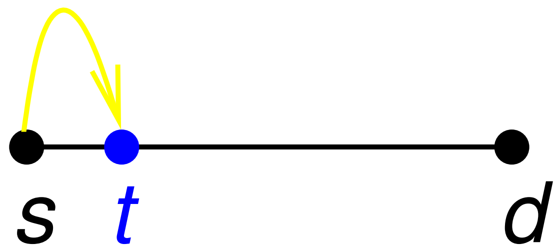


  - Retrieve the quadtree Q_s corresponding to s.
  - Find the colored region that contains d in Q_s.
  - Retrieve the vertex t connected to s in the region containing d in Q_s.
  - Retrieve the quadtree Q_t corresponding to t.

Path Retrieval

- Problem: How to retrieve the shortest path between source s and destination d.

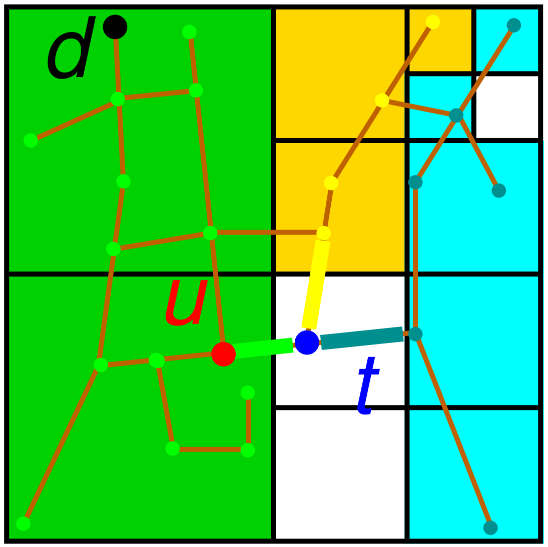


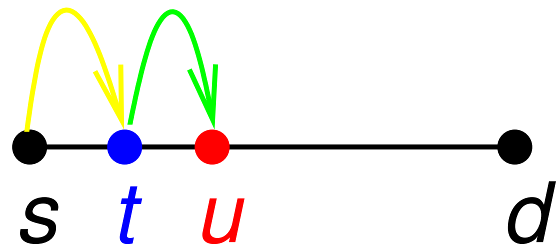


- Retrieve the quadtree Q_s corresponding to s.
- Find the colored region that contains d in Q_s.
- Retrieve the vertex t connected to s in the region containing d in Q_s.
- Retrieve the quadtree Q_t corresponding to t.
- Find the colored region that contains d in Q_t.

Path Retrieval

- Problem: How to retrieve the shortest path between source s and destination d.

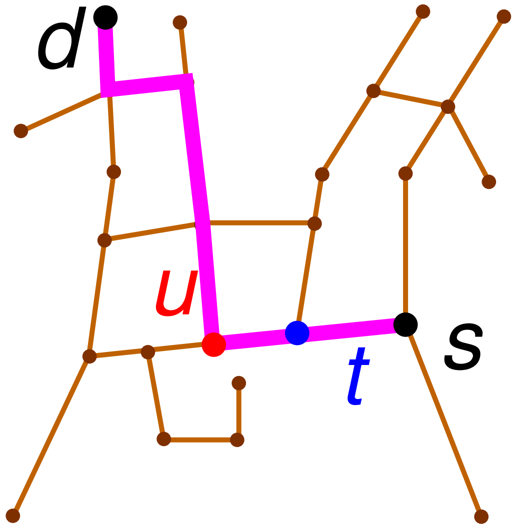


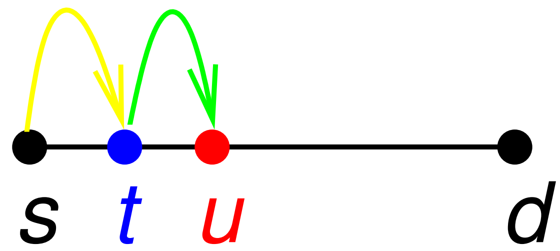


  - Retrieve the quadtree Q_s corresponding to s.
  - Find the colored region that contains d in Q_s.
  - Retrieve the vertex t connected to s in the region containing d in Q_s.
  - Retrieve the quadtree Q_t corresponding to t.
  - Find the colored region that contains d in Q_t.
  - Retrieve the vertex u connected to t in the region containing d in Q_t.

Path Retrieval

- Problem: How to retrieve the shortest path between source s and destination d.

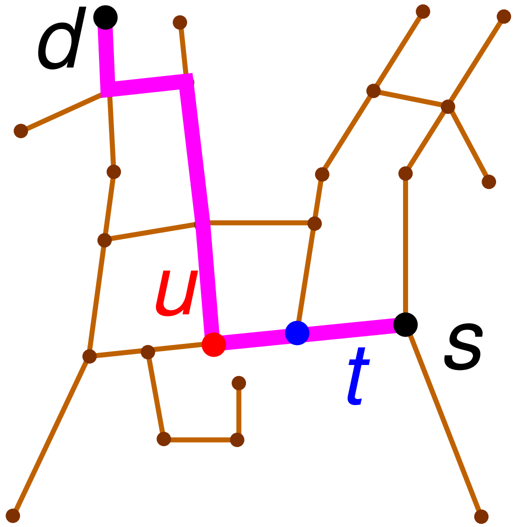


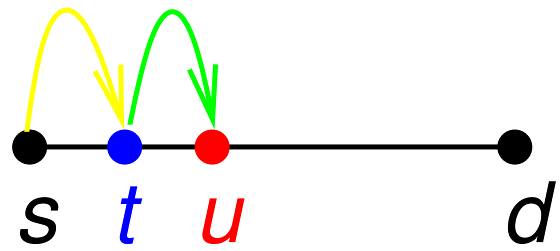


  - Retrieve the quadtree Q_s corresponding to s.
  - Find the colored region that contains d in Q_s.
  - Retrieve the vertex t connected to s in the region containing d in Q_s.
  - Retrieve the quadtree Q_t corresponding to t.
  - Find the colored region that contains d in Q_t.
  - Retrieve the vertex u connected to t in the region containing d in Q_t.
- The entire shortest path between s and d can be retrieved in (size of path) steps.

Path Retrieval

- Problem: How to retrieve the shortest path between source s and destination d.

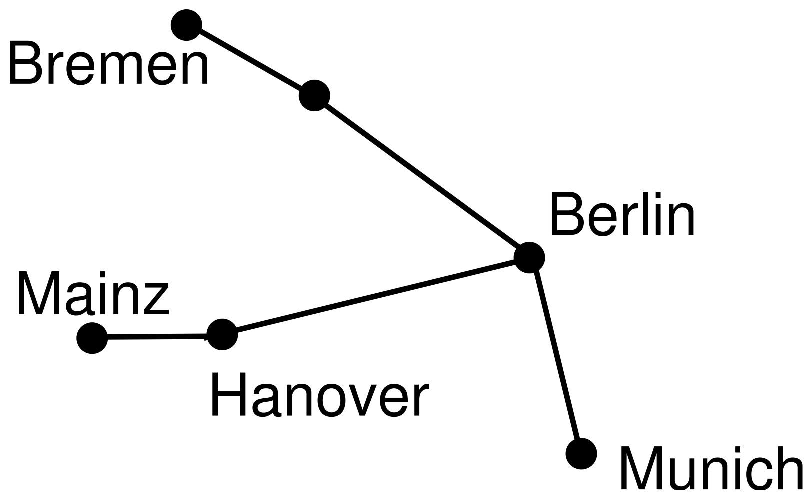




  - Retrieve the quadtree Q_s corresponding to s.
  - Find the colored region that contains d in Q_s.
  - Retrieve the vertex t connected to s in the region containing d in Q_s.
  - Retrieve the quadtree Q_t corresponding to t.
  - Find the colored region that contains d in Q_t.
  - Retrieve the vertex u connected to t in the region containing d in Q_t.
- The entire shortest path between s and d can be retrieved in (size of path) steps.
- The distance information is immediately obtained from shortest path.

Progressive Refinements

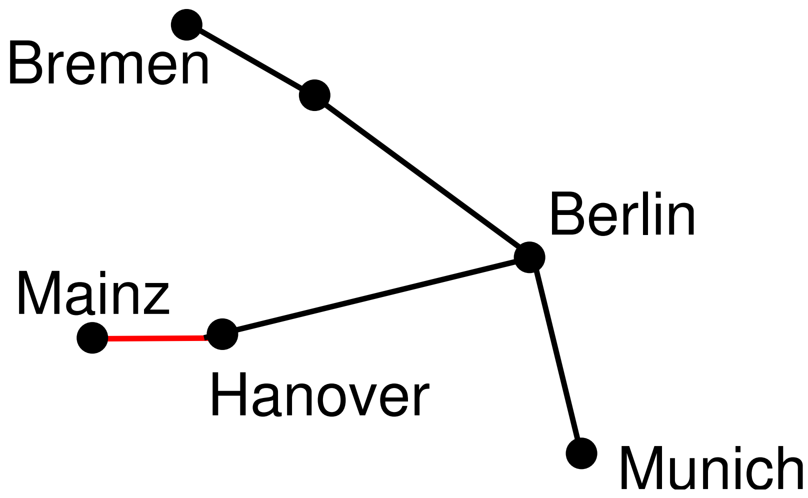
- Many queries require distance comparison primitives.
  - Example: Is Munich closer to Mainz than Bremen ?
- Avoid full shortest path retrievals using Progressive Refinement.
- Idea: Use distance intervals instead of the exact distance.
- Progressive refinement: Improve interval if query cannot be answered.
  - Associate Min/Max distance information with each Morton block.
  - Refinement involves finding the next link in the shortest path.
  - Worst case: retrieve entire shortest path to answer query.



	Munich	Bremen
Mainz	[10,20]	[15,30]

Progressive Refinements

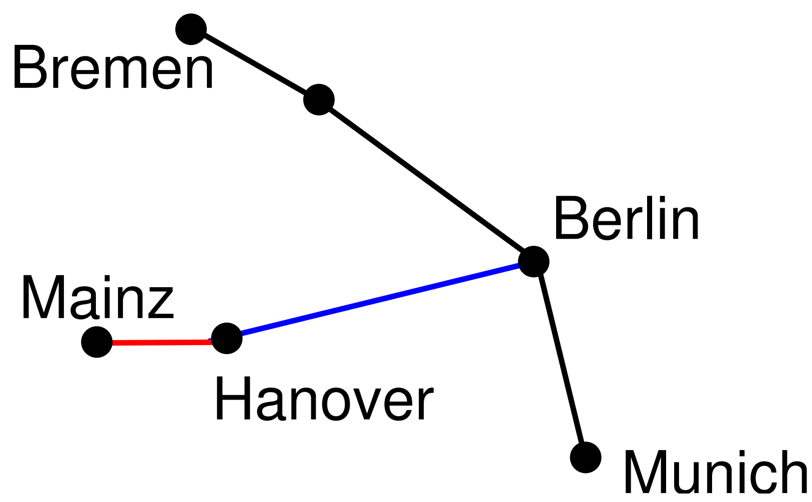
- Many queries require distance comparison primitives.
  - Example: Is Munich closer to Mainz than Bremen ?
- Avoid full shortest path retrievals using Progressive Refinement.
- Idea: Use distance intervals instead of the exact distance.
- Progressive refinement: Improve interval if query cannot be answered.
  - Associate Min/Max distance information with each Morton block.
  - Refinement involves finding the next link in the shortest path.
  - Worst case: retrieve entire shortest path to answer query.



	Munich	Bremen
Mainz	[10,20]	[15,30]
Hanover	[12,18]	[17,20]

Progressive Refinements

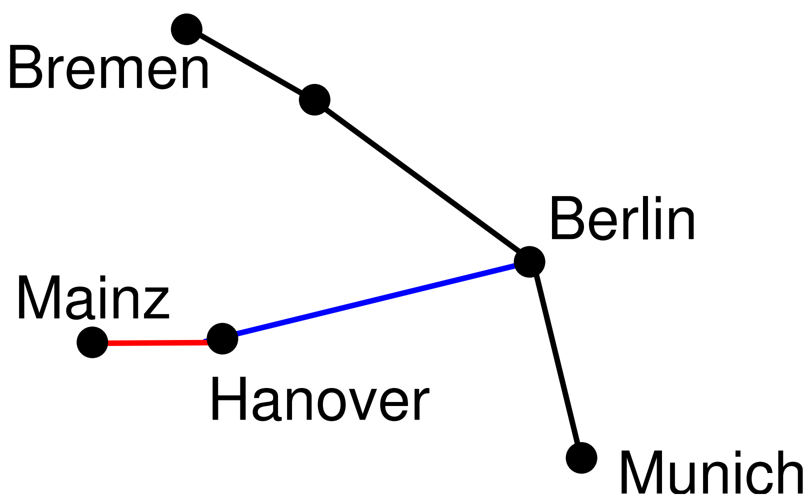
- Many queries require distance comparison primitives.
  - Example: Is Munich closer to Mainz than Bremen ?
- Avoid full shortest path retrievals using Progressive Refinement.
- Idea: Use distance intervals instead of the exact distance.
- Progressive refinement: Improve interval if query cannot be answered.
  - Associate Min/Max distance information with each Morton block.
  - Refinement involves finding the next link in the shortest path.
  - Worst case: retrieve entire shortest path to answer query.



	Munich	Bremen
Mainz	[10,20]	[15,30]
Hanover	[12,18]	[17,20]
Berlin	[13,15]	[18,19]

Progressive Refinements

- Many queries require distance comparison primitives.
  - Example: Is Munich closer to Mainz than Bremen ?
- Avoid full shortest path retrievals using Progressive Refinement.
- Idea: Use distance intervals instead of the exact distance.
- Progressive refinement: Improve interval if query cannot be answered.
  - Associate Min/Max distance information with each Morton block.
  - Refinement involves finding the next link in the shortest path.
  - Worst case: retrieve entire shortest path to answer query.



	Munich	Bremen
Mainz	[10,20]	[15,30]
Hanover	[12,18]	[17,20]
Berlin	[13,15]	[18,19]

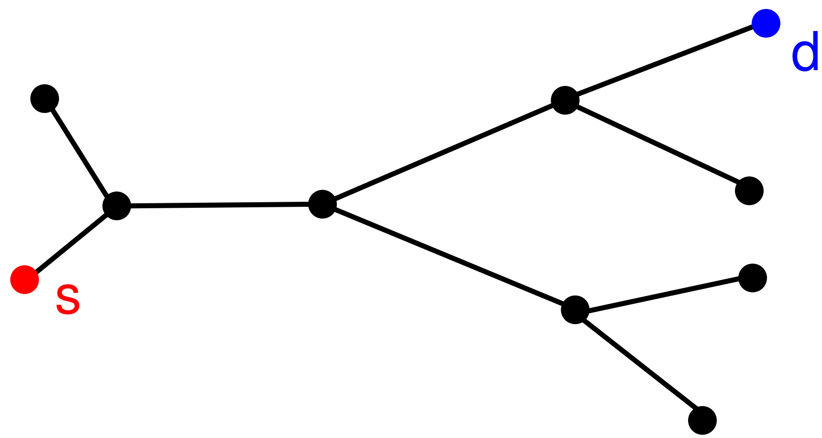
- Munich is closer as distance interval via Berlin does not intersect distance interval to Bremen via Berlin.

Distance Computation in SILC

- Distance primitives computed using progressive refinements.

Distance Computation in SILC

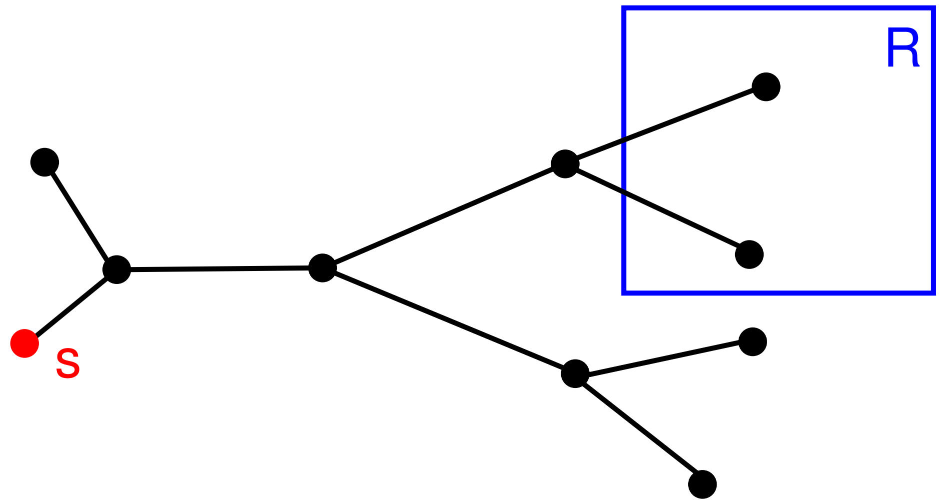
- Distance primitives computed using progressive refinements.



  - DISTANCE_INTERVAL(object,object): used in incremental nearest neighbor (INN)

Distance Computation in SILC

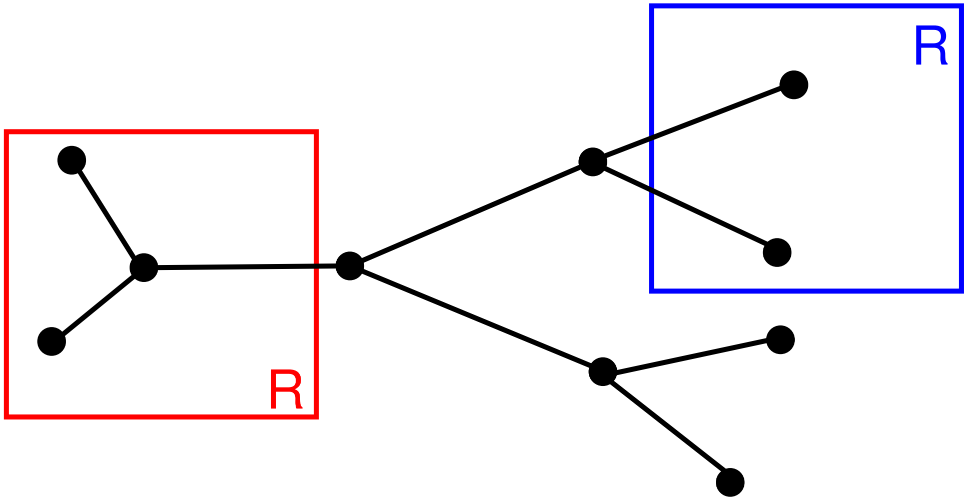
- Distance primitives computed using progressive refinements.



- DISTANCE_INTERVAL(object,object): used in incremental nearest neighbor (INN)
- DISTANCE_INTERVAL(object,Region): only use lower bound in INN

Distance Computation in SILC

- Distance primitives computed using progressive refinements.



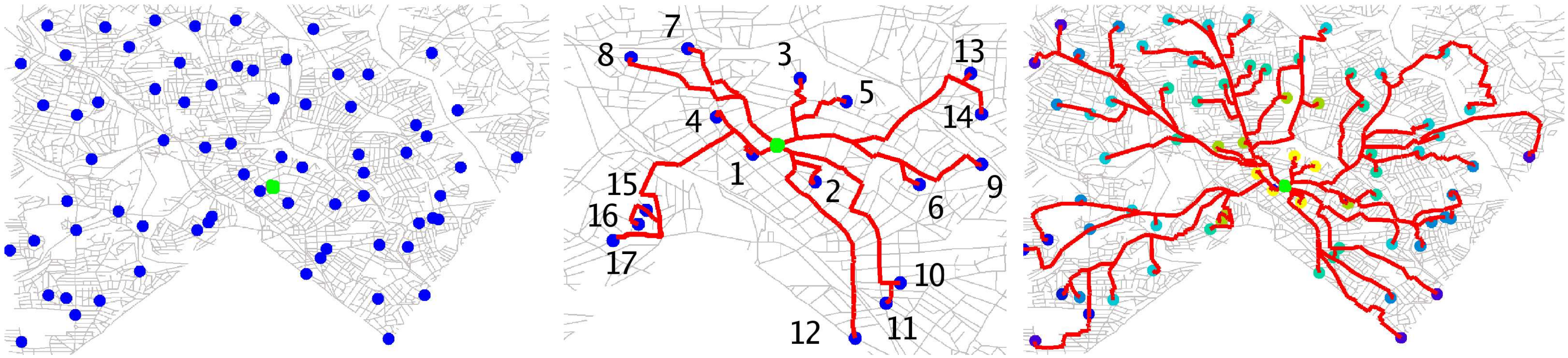
- DISTANCE_INTERVAL(object,object): used in incremental nearest neighbor (INN)
- DISTANCE_INTERVAL(object,Region): only use lower bound in INN
- DISTANCE_INTERVAL(Region,Region): used in incremental join

SILC query Processing

- Inputs to the SILC query processing engine
  - Set of objects (with spatial information).
  - A spatial data structure (*e.g.*, a quadtree or R-tree) on objects.
  - Paths and distances are precomputed and stored using the SILC encoding which means that the Morton blocks contain distance intervals.
- Primitive operations using Progressive Refinement
  - DISTANCE_INTERVAL(object,object)
  - DISTANCE_INTERVAL(object,Region)
  - DISTANCE_INTERVAL(Region,Region)
  - REFINE_INTERVAL($\cdot, \cdot$)

Incremental Nearest Neighbor

- Modified Best First Search (BFS) ([Hjal95]) to use progressive refinement.



(a) (b) (c)

- Mechanics of a nearest neighbor search on a road network.
  a. Initial configuration: A query object (in green) and a set of locations in blue.
  b. Query progression: Partial result of ranking the dataset of objects based on the length of their shortest path from the query object.
  c. Final result: All objects have been ranked by their network distance to the query object.

Incremental Nearest Neighbor

- The algorithm takes three inputs
  - A pointer T to the root of a hierarchical spatial data structure containing the set of objects from which the neighbors are drawn (*e.g.*, a set of hospitals)
  - a query object q
  - shortest path region quadtree corresponding to q.
- The algorithm uses a priority queue Q of objects and blocks.
- The network distance interval $[\delta^-, \delta^+]$ of objects from q is stored in the priority queue Q.
- Additionally, a few additional pieces of information are stored along with each object o in Q.

Incremental Nearest Neighbor

- The algorithm takes three inputs
  - A pointer T to the root of a hierarchical spatial data structure containing the set of objects from which the neighbors are drawn (*e.g.*, a set of hospitals)
  - a query object q
  - shortest path region quadtree corresponding to q.
- The algorithm uses a priority queue Q of objects and blocks.
- The network distance interval $[\delta^-, \delta^+]$ of objects from q is stored in the priority queue Q.
- Additionally, a few additional pieces of information are stored along with each object o in Q.
  - an intermediate vertex u in the shortest path from q to o

Incremental Nearest Neighbor

- The algorithm takes three inputs
  - A pointer T to the root of a hierarchical spatial data structure containing the set of objects from which the neighbors are drawn (*e.g.*, a set of hospitals)
  - a query object q
  - shortest path region quadtree corresponding to q.
- The algorithm uses a priority queue Q of objects and blocks.
- The network distance interval $[\delta^-, \delta^+]$ of objects from q is stored in the priority queue Q.
- Additionally, a few additional pieces of information are stored along with each object o in Q.
  - an intermediate vertex u in the shortest path from q to o
  - the network distance d from q to u

Incremental Nearest Neighbor

- The algorithm takes three inputs
  - A pointer T to the root of a hierarchical spatial data structure containing the set of objects from which the neighbors are drawn (*e.g.*, a set of hospitals)
  - a query object q
  - shortest path region quadtree corresponding to q.
- The algorithm uses a priority queue Q of objects and blocks.
- The network distance interval $[\delta^-, \delta^+]$ of objects from q is stored in the priority queue Q.
- Additionally, a few additional pieces of information are stored along with each object o in Q.
  - an intermediate vertex u in the shortest path from q to o
  - the network distance d from q to u
- Objects are retrieved from Q in an increasing δ^- ordering from q

Incremental Nearest Neighbor

1. The priority queue Q is initialized by inserting the root T.
2. At each iteration of the algorithm, the top element p in the queue is retrieved.

Incremental Nearest Neighbor

1. The priority queue Q is initialized by inserting the root T.
2. At each iteration of the algorithm, the top element p in the queue is retrieved.
3. If p is a LEAF block, then it is replaced with all the objects contained within it and with their associated network distance intervals from their shortest path quadtrees.

Incremental Nearest Neighbor

1. The priority queue Q is initialized by inserting the root T.
2. At each iteration of the algorithm, the top element p in the queue is retrieved.
3. If p is a LEAF block, then it is replaced with all the objects contained within it and with their associated network distance intervals from their shortest path quadtrees.
4. If p is a NON-LEAF block, then all its children blocks are inserted into Q with their associated minimum network distance obtained from their shortest path quadtrees.

Incremental Nearest Neighbor

1. The priority queue Q is initialized by inserting the root T.
2. At each iteration of the algorithm, the top element p in the queue is retrieved.
3. If p is a LEAF block, then it is replaced with all the objects contained within it and with their associated network distance intervals from their shortest path quadtrees.
4. If p is a NON-LEAF block, then all its children blocks are inserted into Q with their associated minimum network distance obtained from their shortest path quadtrees.
5. If p is a OBJECT, then the distance interval of p is checked with the *current* top element in Q for possible *collisions*

Incremental Nearest Neighbor

1. The priority queue Q is initialized by inserting the root T.
2. At each iteration of the algorithm, the top element p in the queue is retrieved.
3. If p is a LEAF block, then it is replaced with all the objects contained within it and with their associated network distance intervals from their shortest path quadtrees.
4. If p is a NON-LEAF block, then all its children blocks are inserted into Q with their associated minimum network distance obtained from their shortest path quadtrees.
5. If p is a OBJECT, then the distance interval of p is checked with the *current* top element in Q for possible *collisions*
   - A collision occurs if the distance interval of p intersects with the distance interval of the current top element in Q

Incremental Nearest Neighbor

1. The priority queue Q is initialized by inserting the root T.
2. At each iteration of the algorithm, the top element p in the queue is retrieved.
3. If p is a LEAF block, then it is replaced with all the objects contained within it and with their associated network distance intervals from their shortest path quadtrees.
4. If p is a NON-LEAF block, then all its children blocks are inserted into Q with their associated minimum network distance obtained from their shortest path quadtrees.
5. If p is a OBJECT, then the distance interval of p is checked with the *current* top element in Q for possible *collisions*
   - A collision occurs if the distance interval of p intersects with the distance interval of the current top element in Q
   - Collision:

Incremental Nearest Neighbor

1. The priority queue Q is initialized by inserting the root T.
2. At each iteration of the algorithm, the top element p in the queue is retrieved.
3. If p is a LEAF block, then it is replaced with all the objects contained within it and with their associated network distance intervals from their shortest path quadtrees.
4. If p is a NON-LEAF block, then all its children blocks are inserted into Q with their associated minimum network distance obtained from their shortest path quadtrees.
5. If p is a OBJECT, then the distance interval of p is checked with the *current* top element in Q for possible *collisions*
   - A collision occurs if the distance interval of p intersects with the distance interval of the current top element in Q
   - Collision: improve network distance interval of p by applying Refinement operation and reinsert p into Q with new intermediate vertex and distance from q. Go to Step 2.

Incremental Nearest Neighbor

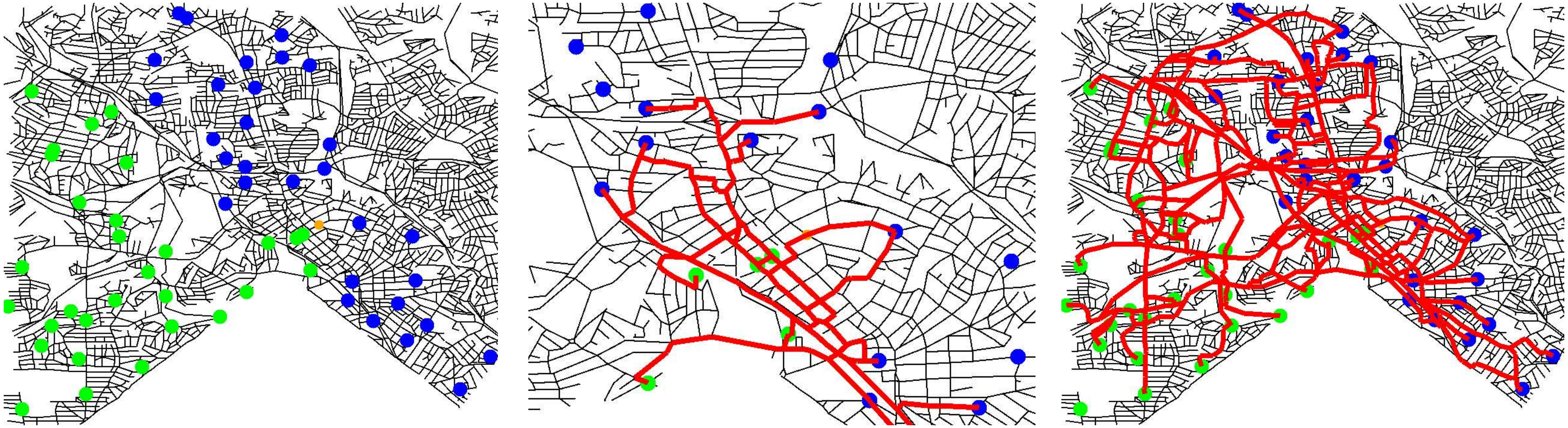
1. The priority queue Q is initialized by inserting the root T.
2. At each iteration of the algorithm, the top element p in the queue is retrieved.
3. If p is a LEAF block, then it is replaced with all the objects contained within it and with their associated network distance intervals from their shortest path quadtrees.
4. If p is a NON-LEAF block, then all its children blocks are inserted into Q with their associated minimum network distance obtained from their shortest path quadtrees.
5. If p is a OBJECT, then the distance interval of p is checked with the *current* top element in Q for possible *collisions*
   - A collision occurs if the distance interval of p intersects with the distance interval of the current top element in Q
   - Collision: improve network distance interval of p by applying Refinement operation and reinsert p into Q with new intermediate vertex and distance from q. Go to Step 2.
   - No collision:

Incremental Nearest Neighbor

1. The priority queue Q is initialized by inserting the root T.
2. At each iteration of the algorithm, the top element p in the queue is retrieved.
3. If p is a LEAF block, then it is replaced with all the objects contained within it and with their associated network distance intervals from their shortest path quadtrees.
4. If p is a NON-LEAF block, then all its children blocks are inserted into Q with their associated minimum network distance obtained from their shortest path quadtrees.
5. If p is a OBJECT, then the distance interval of p is checked with the *current* top element in Q for possible *collisions*
   - A collision occurs if the distance interval of p intersects with the distance interval of the current top element in Q
   - Collision: improve network distance interval of p by applying Refinement operation and reinsert p into Q with new intermediate vertex and distance from q. Go to Step 2.
   - No collision: report p as next nearest neighbor of q and return to caller for possible reinvocation for incremental nearest neighbor.

Distance Joins

- Modified the join and distance semi-join algorithms ([Hjal98]) to use progressive refinement.



(a) (b) (c)

- Mechanics of an incremental distance join [Hjal98] on a road network.
  a. Initial configuration: The road network and two sets of locations.
  b. Query progression: At each step, the distance join fetches the next closest pair of locations, one drawn from either of the sets of locations.
  c. Final result: All pairs of locations obtained by the distance join and the shortest paths between them.

Other "Incremental" Methods

1. IER method [Papa03]: not an incremental network distance algorithm
   - Use incremental nearest neighbor algorithm to find k nearest neighbors using Euclidean distance
   - Find the network distance of these k nearest neighbors using Dijkstra's algorithm and sort in increasing order
   - Apply incremental nearest neighbor algorithm using Euclidean distance until obtaining an object whose Euclidean distance is greater than the farthest of the current k nearest network distance neighbors
2. INE method [Papa03]: k-nearest neighbor network distance algorithm
   - Really Dijkstra's algorithm with a buffer L containing the k nearest neighbors seen so far in terms of network distance.
   - Halt when current neighbor is farther than the farthest of the k nearest neighbors in L.
3. Advantage of our method is that Dijkstra's Algorithm is only applied once regardless of the number of queries instead of once for each query.

Related Work

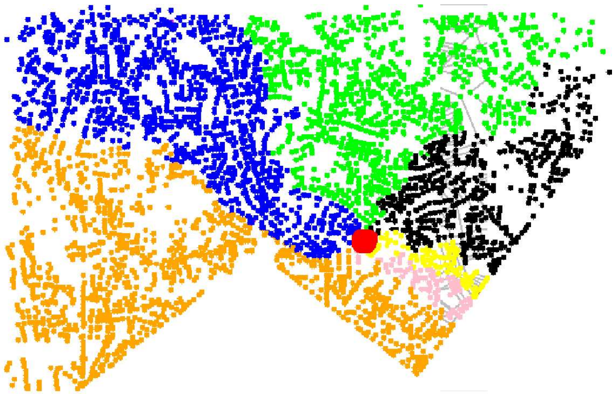
- The hierarchal graph representation of Jing *et al.* [Jing98] and Filho and Samet [Filh02]
- The Road Network Embedding (RNE) technique proposed by Shahabi et al. [Shah03] uses a Lipschitz embedding where subsets of vertices serve as coordinates and hence a mapping into a higher dimensional space resulting in an approximate shortest path
- The ALT method by Goldberg and Harrelson [Gold05a] utilizes *landmarks* for speeding up shortest path computation.
- Geometric speedup approach of Wagner and Willhalm [Wagn03]

Geometric Speedup Approach

- Wagner and Willhalm [Wagn03] represent the individual regions in colored map for a vertex by their minimum bounding boxes
  - Bounding boxes are organized by an index for an object hierarchy such as an R-tree.

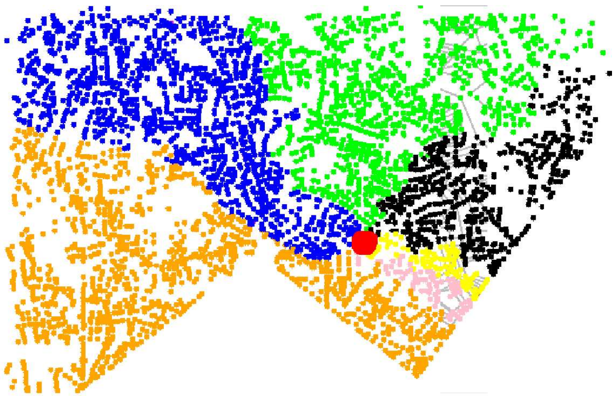
Geometric Speedup Approach

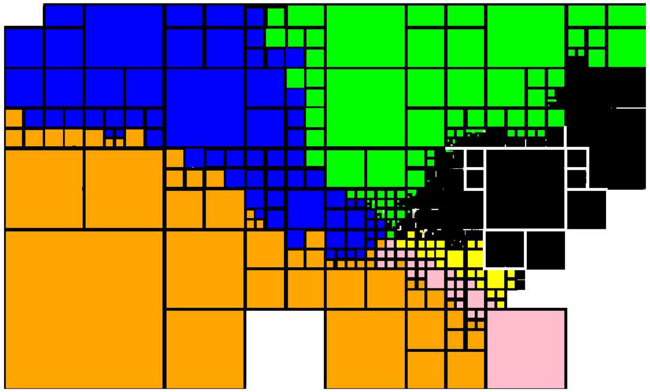
- Wagner and Willhalm [Wagn03] represent the individual regions in colored map for a vertex by their minimum bounding boxes
  - Bounding boxes are organized by an index for an object hierarchy such as an R-tree.



Geometric Speedup Approach

- Wagner and Willhalm [Wagn03] represent the individual regions in colored map for a vertex by their minimum bounding boxes
  - Bounding boxes are organized by an index for an object hierarchy such as an R-tree.

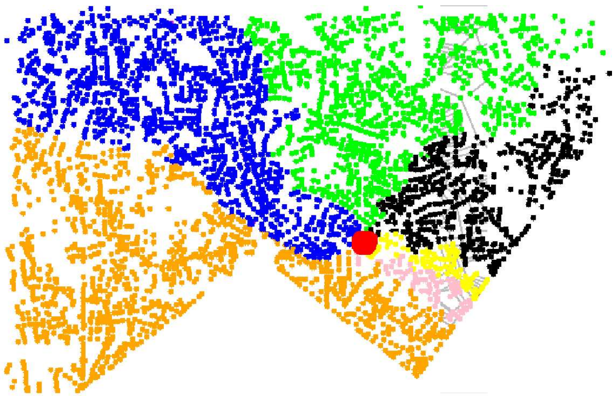


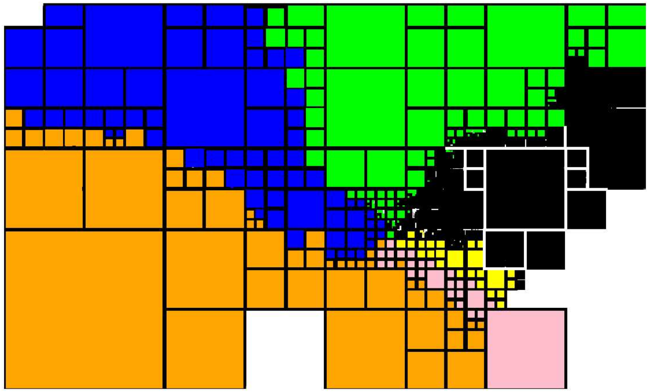


SILC -disjoint decomposition

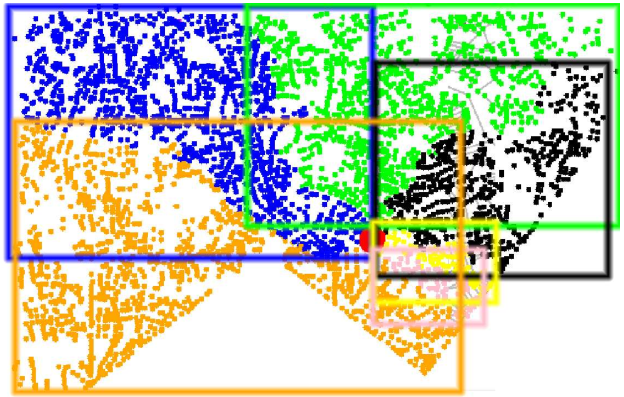
Geometric Speedup Approach

- Wagner and Willhalm [Wagn03] represent the individual regions in colored map for a vertex by their minimum bounding boxes
  - Bounding boxes are organized by an index for an object hierarchy such as an R-tree.





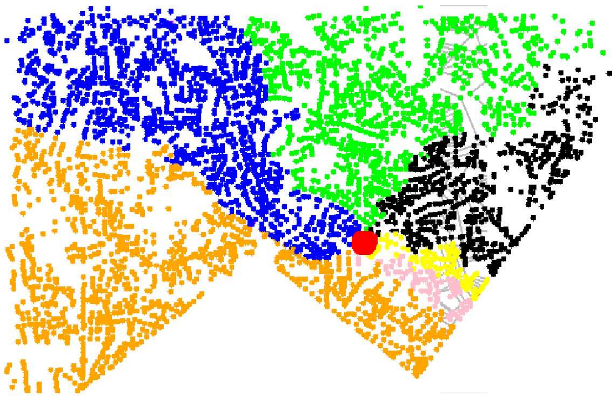
SILC -disjoint decomposition

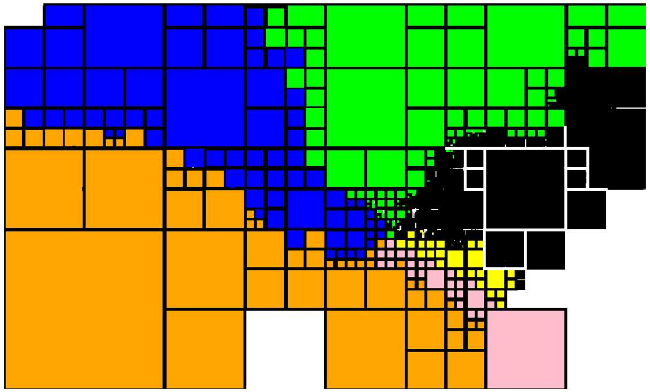


Bounding boxes with overlap

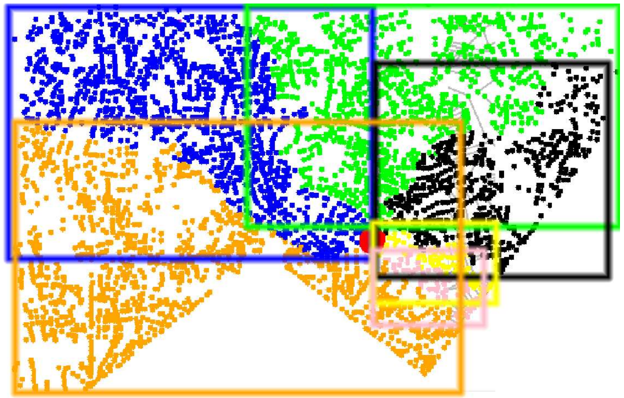
Geometric Speedup Approach

- Wagner and Willhalm [Wagn03] represent the individual regions in colored map for a vertex by their minimum bounding boxes
  - Bounding boxes are organized by an index for an object hierarchy such as an R-tree.





SILC -disjoint decomposition



Bounding boxes with overlap

- The drawback of the approach is that the bounding boxes are not disjoint, which complicates future searches.

Conclusion

- SILC contributions:
  - Use path coherence to encode paths efficiently.
  - Use progressive refinements to process queries.
  - Exploits existing spatial database techniques for spatial networks.
- Future work:
  - Complexity analysis
  - Handle dynamic spatial networks
    - Road networks with real-time traffic information.
    - Route planning with link failures.

References

1. [Same05] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, CA, USA, 2005.
2. [Filh02] G. G. Filho and H. Samet. A linear iterative approach for hierarchical shortest path finding. Computer Science Department CS-TR-4417, University of Maryland, College Park, MD, Nov. 2002.
3. [Gold05a] A. V. Goldberg and C. Harrelson. Computing the shortest path: A^* search meets graph theory. In *SODA '05: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, Vancouver, BC, Canada, Jan. 2005.
4. [Hjal95] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *SSD '95: Proceedings of the 4th International Symposium on Large Spatial Databases*, pages 83–95, Portland, ME, USA, Aug. 1995.
5. [Hjal98] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *SIGMOD '98: Proceedings of the International Conference on Management of Data*, pages 237–248, Seattle, WA, USA, June 1998.

References

6. [Jing98] N. Jing, Y.-W. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: an optimal model and its performance evaluation. ons of Multidimensional and Metric Data Structures. Morgan-Kaufmann, San Francisco, CA, USA, 2005.
7. [Papa03] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB'03: Proceedings of the 29th International Conference on Very Large Databases*, pages 802–813, Berlin, Germany, Sept. 2003.
8. [Shah03] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. *GeoInformatica*, 7(3):255–273, Sept. 2003.
9. [Shek03] S. Shekhar and J. S. Yoo. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In *GIS '03: Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems*, pages 9–16, New Orleans, LA, USA, 2003.
10. [Wagn03] D. Wagner and T. Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In *ESA '03: Proceedings of the 11th Annual European Symposium on Algorithms*, pages 776–787, 2003.