USC Viterbi
School of Engineering

Scalable Network Distance Browsing in Spatial Database

Instructor: Cyrus Shahabi

Outline

- Motivation
- Overview
- Proposed Approach
- Evaluation
- Questions

Motivation

- Growing Popularity of Online Mapping Services

Challenges

- Real-time response for point-to-point shortest path computation
- Calculating all pairs shortest path is costly
- Storing pre-computed shortest paths is not easy
- Scalability

Contribution:

- Greedy Algorithm Independent: Avoids applying Dijkstra's algorithm for each query which visits all vertices on the shortest path to the destination
  - **Shortest path computation belongs the databases.**
  - **Complexity of the SP computation is reduced to number of nodes in the actual SP path.**
- Pre-computes shortest paths between all vertices in spatial network
  - **Reduces cost of storing shortest paths between all pairs of N vertices from $O(N^3)$ to $O(N^{1.5})$**
- Decouples shortest path and nearest neighbor computation processes
  - **Efficient k-NN techniques rely on partitioning the space (i.e.,road network) based on the data objects**

USC Viterbi
School of Engineering

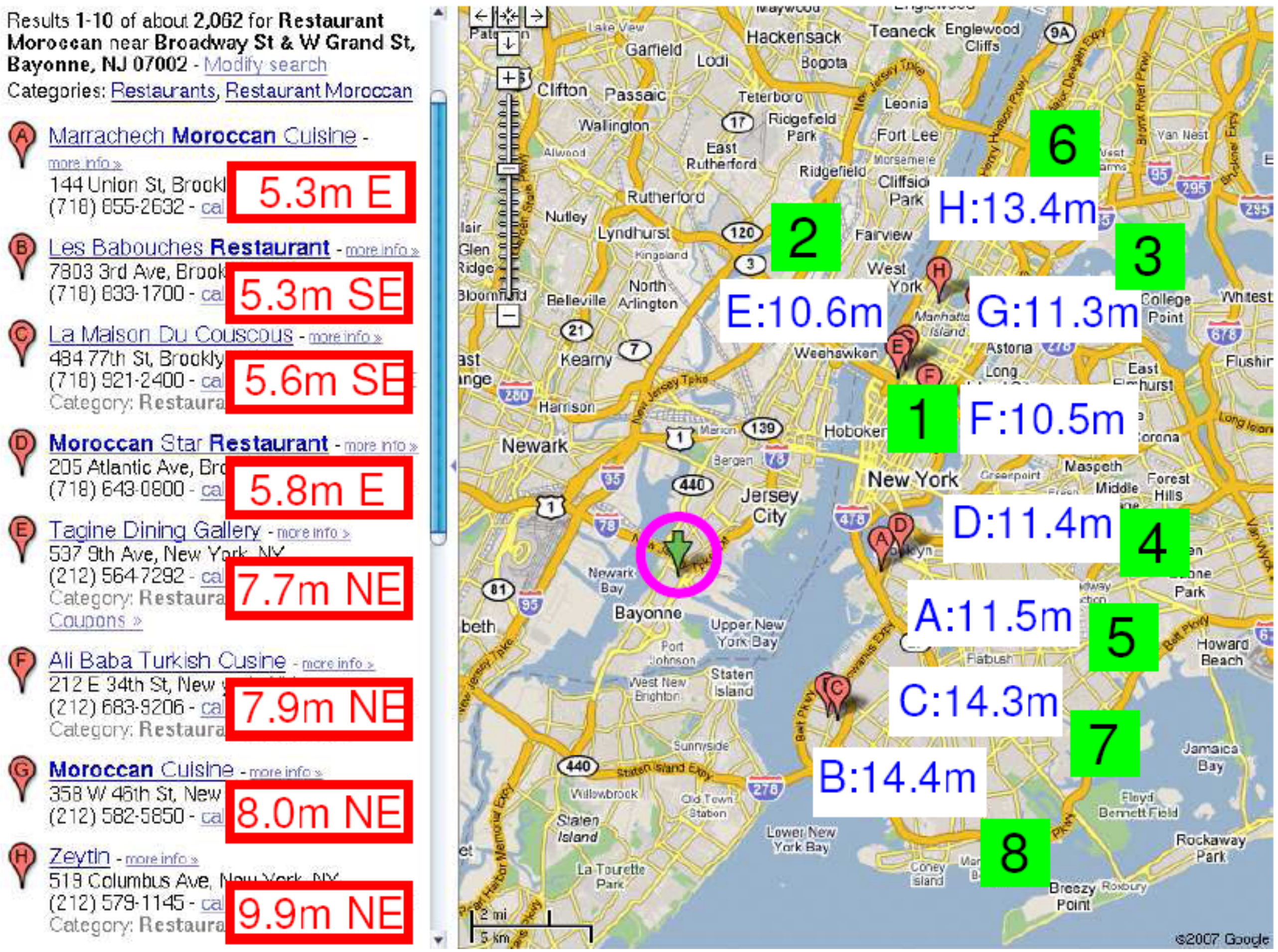
Shortest Path Computation

- Usually based on Dijkstra's or A* shortest path algorithm
  - Not feasible in real time for large spatial networks
  - Algorithm visits too many vertices during the search process

%72 of the vertices are visited.

- Popular solution by online map services is to use Euclidean distance

Even Google!


Results 1-10 of about 2,062 for Restaurant Moroccan near Broadway St & W Grand St, Bayonne, NJ 07002 - Modify search
Categories: Restaurants, Restaurant Moroccan
Marrachech Moroccan Cuisine - more info »
144 Union St, Brookl
(718) 855-2632
5.3m E
Les Babouches Restaurant - more info »
7803 3rd Ave, Brook
(718) 833-1700
5.3m SE
La Maison Du Couscous - more info »
484 77th St, Brookly
(718) 921-2400
Category: Restaura
5.6m SE
Moroccan Star Restaurant - more info »
205 Atlantic Ave, Bro
(718) 643-0800
5.8m E
Tagine Dining Gallery - more info »
537 9th Ave, New York, NY
(212) 564-7292
Category: Restaura
Coupons »
7.7m NE
Ali Baba Turkish Cusine - more info »
212 E 34th St, New
(212) 683-9206
Category: Restaura
7.9m NE
Moroccan Cuisine - more info »
358 W 46th St, New
(212) 582-5850
8.0m NE
Zeytin - more info »
519 Columbus Ave,
(212) 579-1145
Category: Restaura
9.9m NE
6
H:13.4m
2
3
E:10.6m
G:11.3m
1
F:10.5m
D:11.4m
4
A:11.5m
5
C:14.3m
7
B:14.4m
8
Newark
New York
Jersey City
Bayonne
Hoboken
Staten Island
©2007 Google

USC Viterbi School of Engineering

Precomputation of SP

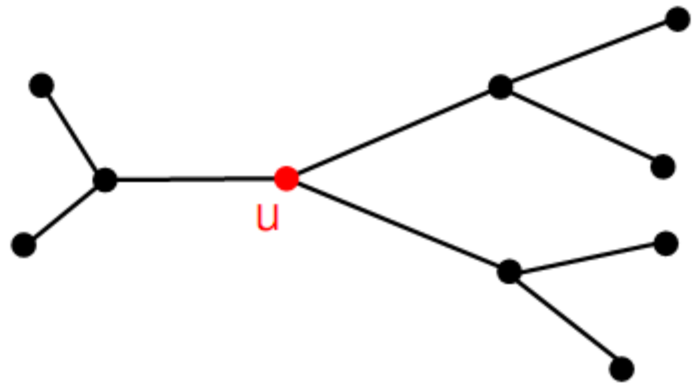
- By precomputing and storing all of the shortest paths, point-to-point SP and nearest neighbor queries could be answered instantly
  - How to effectively compute the shortest path?
  - How to effectively store the shortest path?
  - Challenge: very large network (approximately 45 million nodes in North America)

Method	Space	Retrieval Time
Exhaustive	$O(N^3)$	$O(1)$
Next-Hop	$O(N^2)$	$O(P)$
Dijkstra	$O(N+M)$	$O(M+N\log N)$
SLIC	$O(N^{1.5})$	$O(P \log N)$

- **N:**Nodes, **M:**Edges, **P:**Number of nodes in the path

Path Encoding

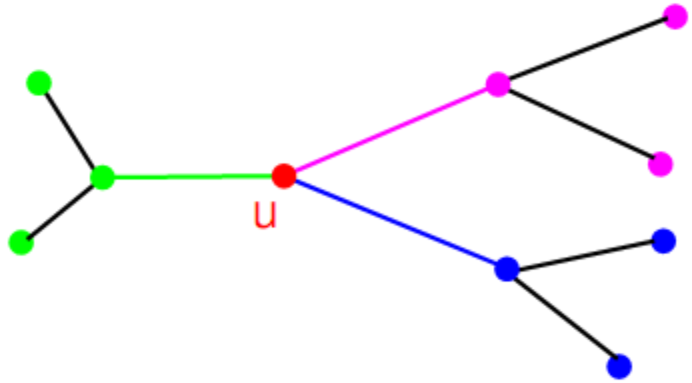
- Path coherence
  - Vertices in proximity share portion of the shortest paths to them from distant sources



USC Viterbi School of Engineering

Path Encoding

- Path coherence
  - Vertices in proximity share portion of the shortest paths to them from distant sources

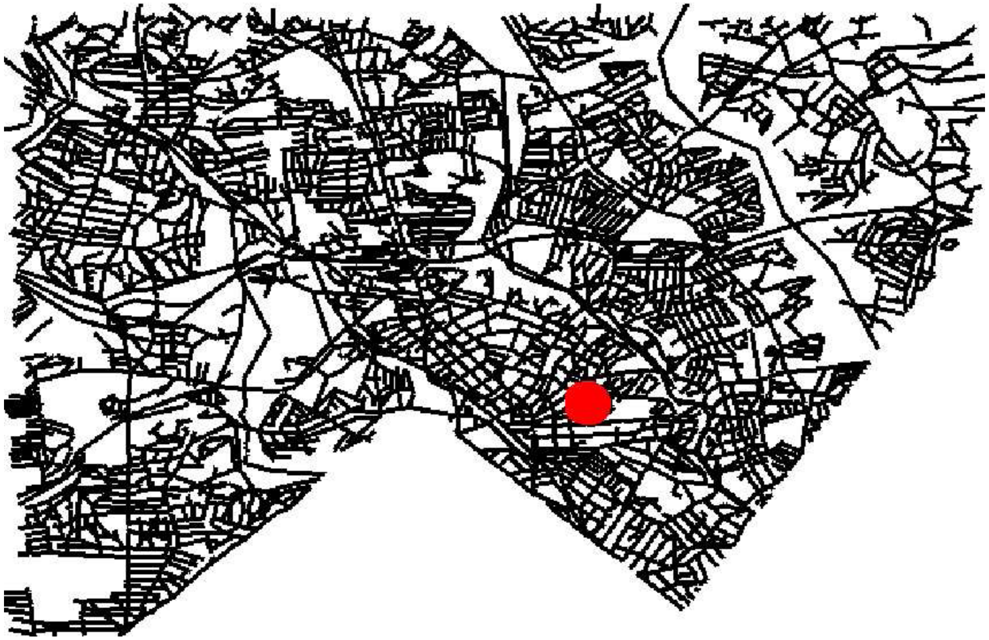


- Source vertex u in a spatial network
- Assign colors to the **outgoing edges** of u
- Color vertex based on the first edge on the shortest path from u

USC Viterbi School of Engineering

Path Encoding

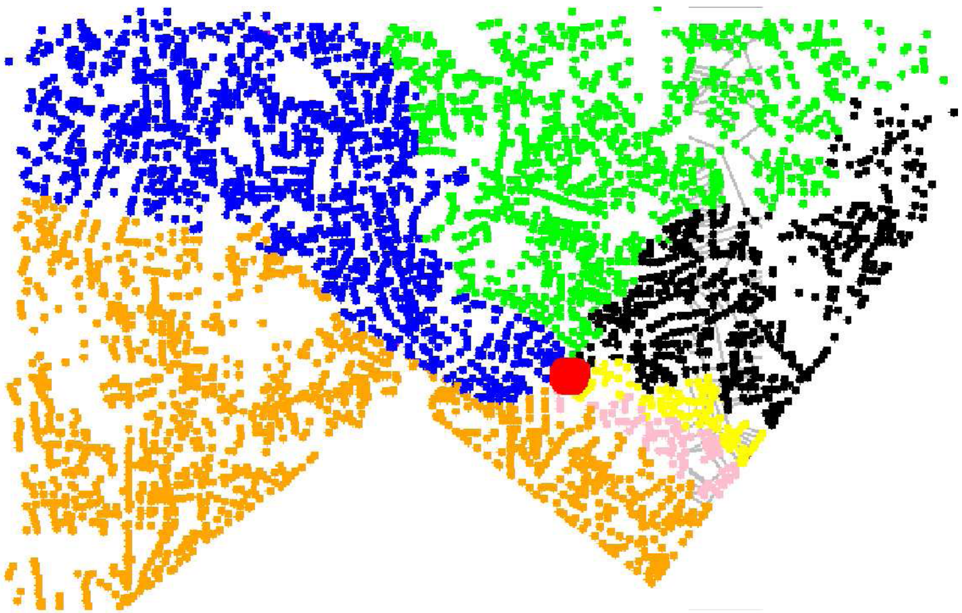
- Path coherence



USC Viterbi School of Engineering

Path Encoding

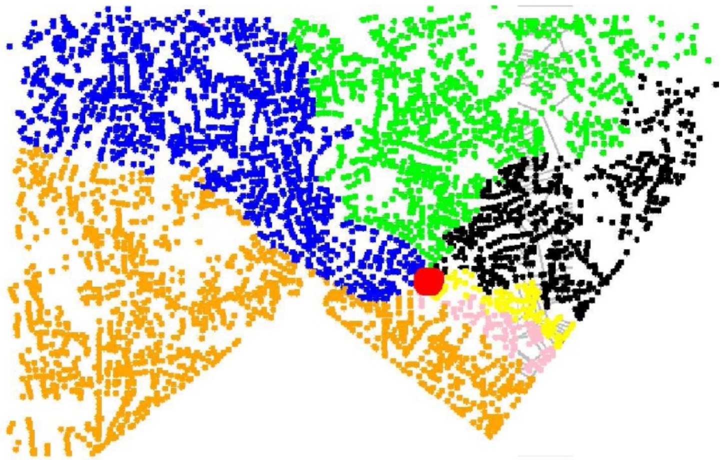
- Path coherence

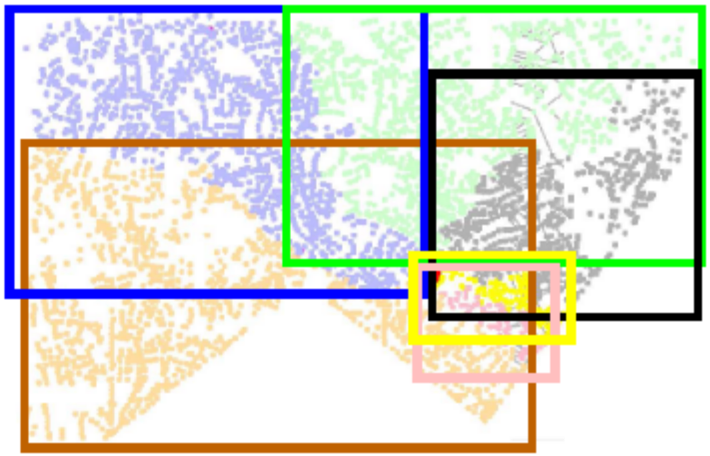


Shortest path map of U

Path Encoding

- **How to store and access colored regions?**



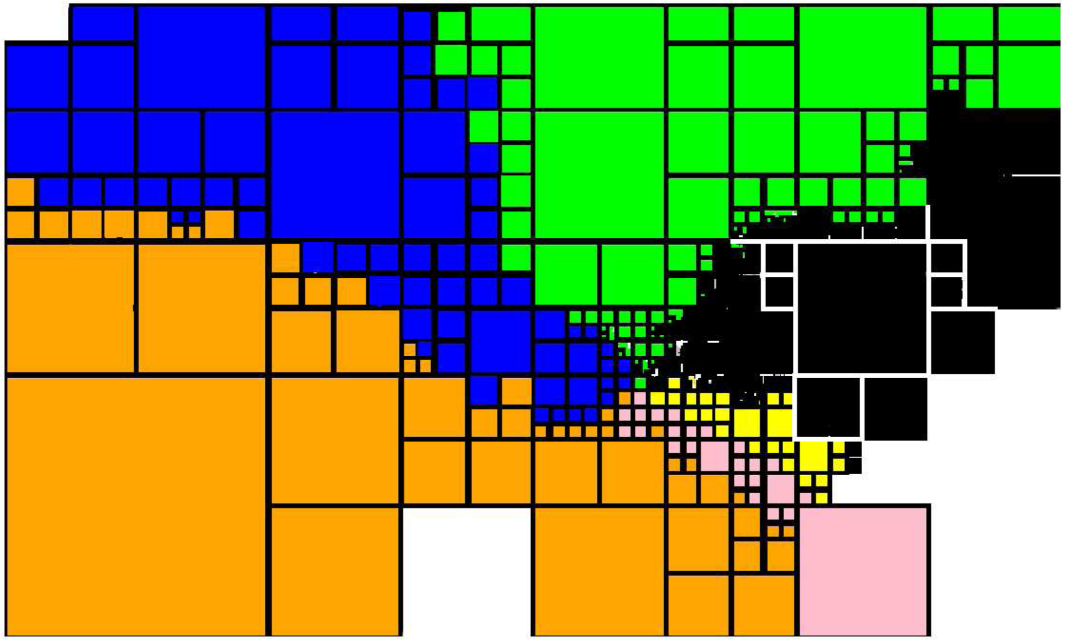


- **Indexing regions with R-tree [Wagner03]**

Path Encoding

- Indexing with Quad-Tree (SILC): Decompose each colored region until all vertices in a block have same color.

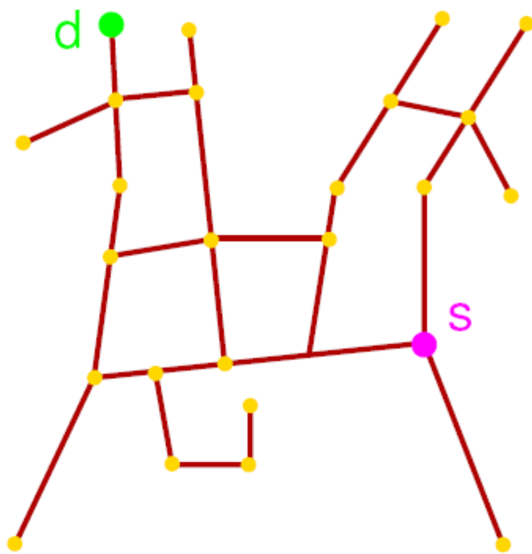
Shortest Path Quad-Tree

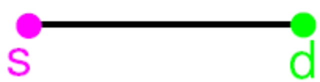


USC Viterbi School of Engineering

Path Retrieval

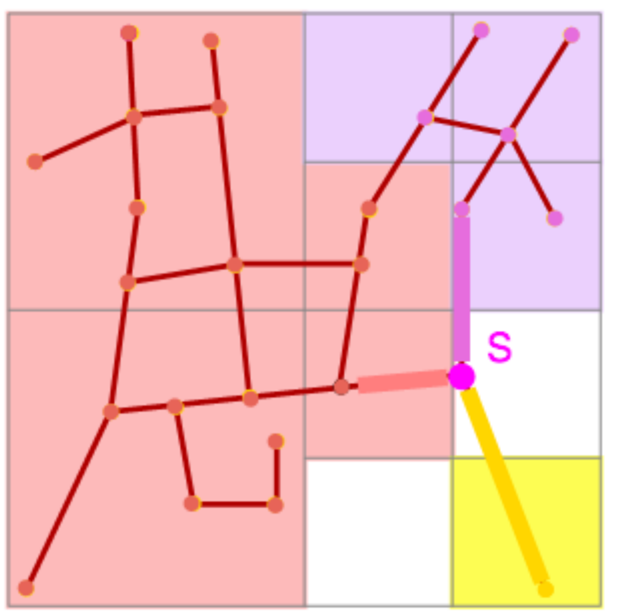
- How to retrieve the shortest path from s to d using the shortest path Quad-tree?





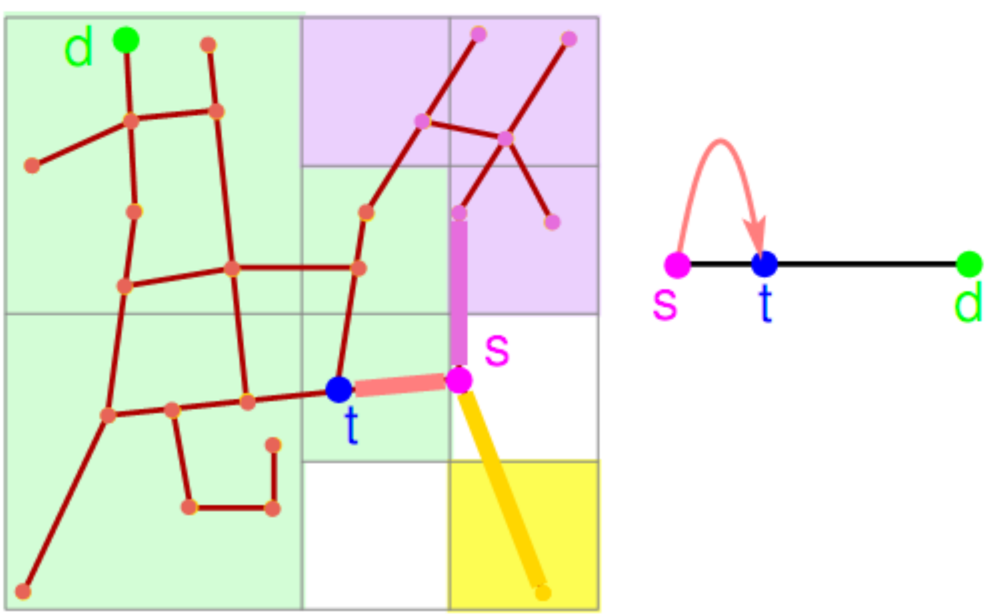
Path Retrieval

- Retrieve the shortest-path quadtree **Q_s** corresponding to s



Path Retrieval

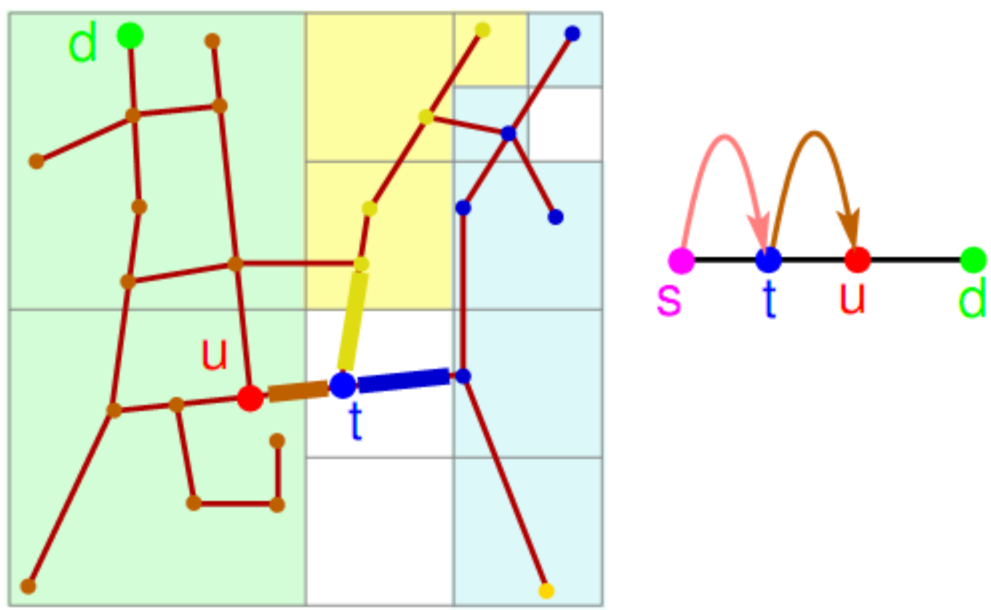
- Find the colored region that contains d in $\mathbf{Q}_s$
- Retrieve the vertex t connected to s in the region containing d in $\mathbf{Q}_s$



USC Viterbi
School of Engineering

Path Retrieval

- Retrieve the shortest-path quadtree $\mathbf{Q_t}$ corresponding to t
- Find the colored region that contains d in $\mathbf{Q_t}$

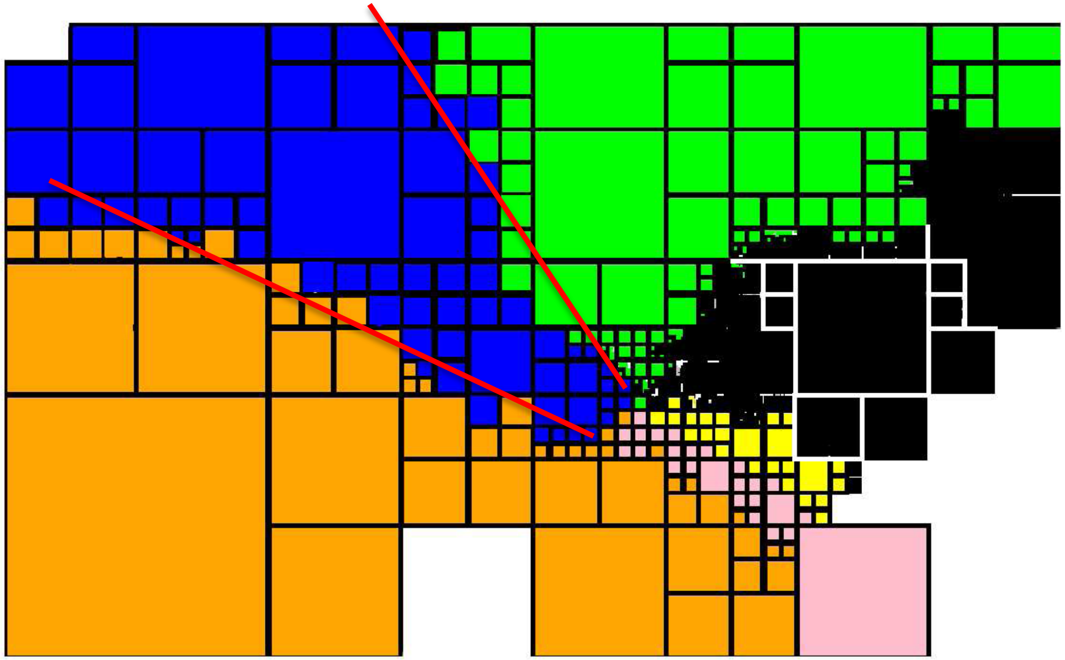


Contribution:

- Greedy Algorithm Independent: Avoids applying Dijkstra's algorithm for each query which visits all vertices on the shortest path to the destination
  - **Shortest path computation belongs the databases.**
  - **Complexity of the SP computation is reduced to number of nodes in the actual SP path.**
- Pre-computes shortest paths between all vertices in spatial network
  - **Reduces cost of storing shortest paths between all pairs of N vertices from $O(N^3)$ to $O(N^{1.5})$**
- Decouples shortest path and nearest neighbor computation processes
  - **Efficient k-NN techniques rely on partitioning the space (i.e.,road network) based on the data objects**

How is space reduced?

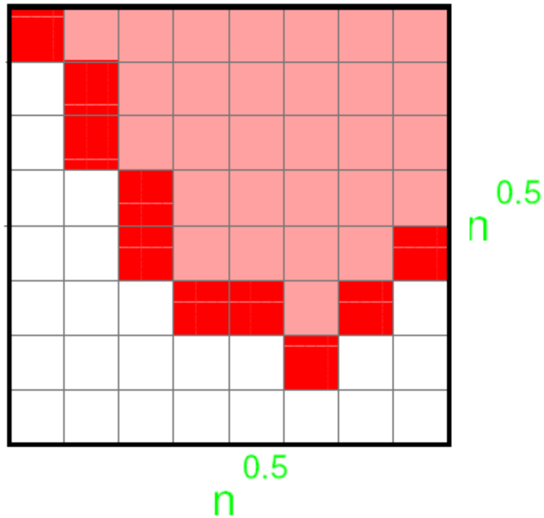
- Only consider boundaries

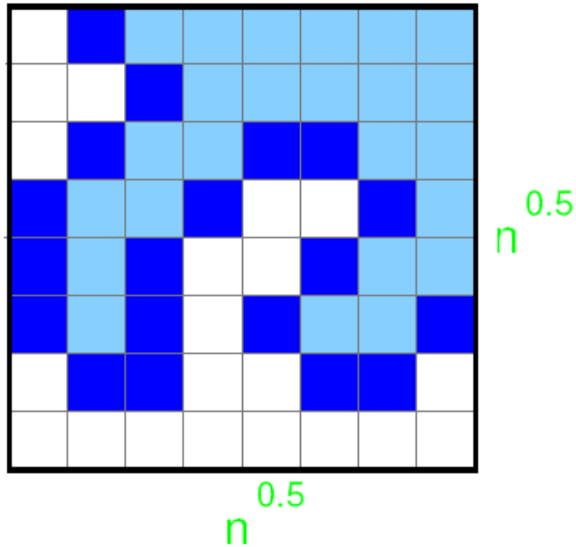


USC Viterbi School of Engineering

How is space reduced?

- Embedding the *N* vertices in a square grid implies that the grid is $N^{0.5} \times N^{0.5}$
- Perimeter of a region with **monotonic** boundary (increasing in each coordinate) is of size $O(N^{0.5})$
- Perimeter of a region with **a non-monotonic** boundary can be of size **O(N)**
- **Assumption with SILC:** Regions of the shortest-path quadtree have ***monotonic** boundaries*
- The space complexity of the shortest path quadtree corresponding to the shortest path map is proportional to the sum of the perimeters of the polygons that make up the shortest path map.
- Size of a shortest-path quadtree of a **vertex u** is $\mathbf{cxN^{0.5}}$, where **c** is the outdegree of u
- **Total storage complexity** of SILC framework is $\mathbf{O(N*N^{0.5})=O(N^{1.5})}$ closely follows empirical results





USC Viterbi School of Engineering

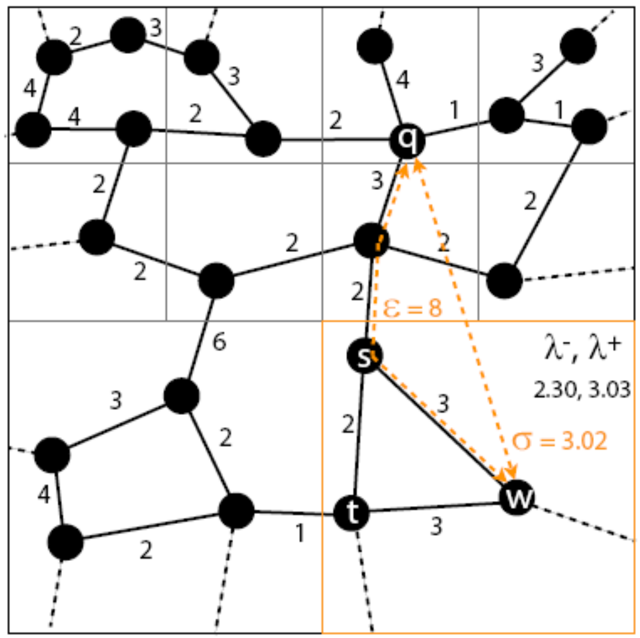
Contribution:

- Greedy Algorithm Independent: Avoids applying Dijkstra's algorithm for each query which visits all vertices on the shortest path to the destination
  - **Shortest path computation belongs the databases.**
  - **Complexity of the SP computation is reduced to number of nodes in the actual SP path.**
- Pre-computes shortest paths between all vertices in spatial network
  - **Reduces cost of storing shortest paths between all pairs of N vertices from $O(N^3)$ to $O(N^{1.5})$**
- Decouples shortest path and nearest neighbor computation processes
  - **Efficient k-NN techniques rely on partitioning the space (i.e.,road network) based on the data objects**

K-NN Algorithm

- Set of objects
- Pre-computed paths (quadtree)

k-NN Algorithm



Distances:

$\varepsilon(q,w) = 8$ (Network)
$\sigma(q,w) = 3.02$ (Spatial)
$\lambda = \varepsilon / \sigma$ (Ratio)
$\lambda(q,w) = 8/3.02 = 2.65$
$\lambda(q,s) = 5/1.65 = 3.03$
$\lambda(q,t) = 7/3.05 = 2.30$
$(\lambda^-, \lambda^+) = (2.30, 3.03)$

Distance Estimate q-w:

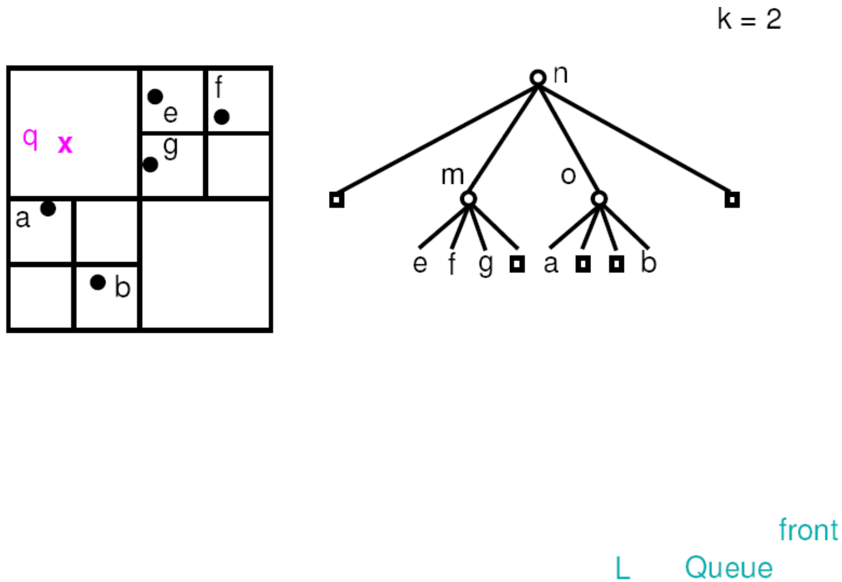
$w(3.02 \times (\lambda^-, \lambda^+)) =$
$w(\delta^-, \delta^+)$
$w(6.95, 9.15)$

δ^- Min. Network Distance
δ^+ Max. Network Distance

Two more properties are stored in each block *b* of the quadtrees: λ^- and λ^+

At the query time, the min and max network distance to specific node w is computed by $w(\sigma \times (\lambda^-, \lambda^+)) = w(\delta^-, \delta^+)$

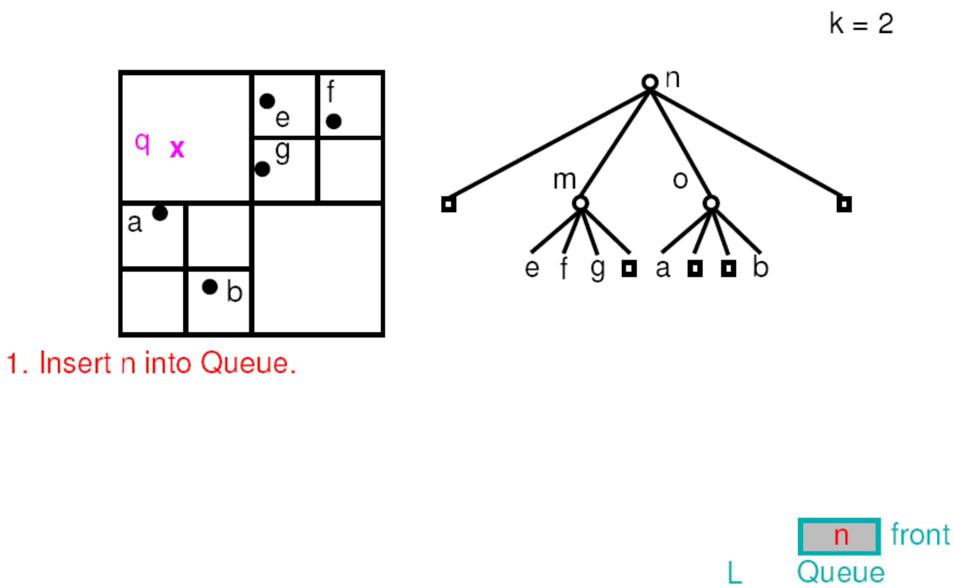
kNN Example



L is ordered based on the maximum network distances, δ^+
Queue is ordered based on the minimum network distances, δ^-

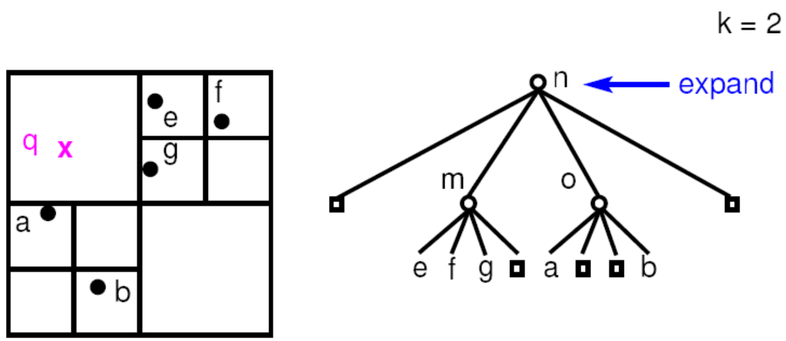
USC Viterbi School of Engineering

kNN Example

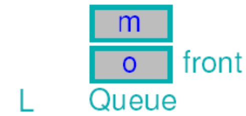


USC Viterbi School of Engineering

kNN Example



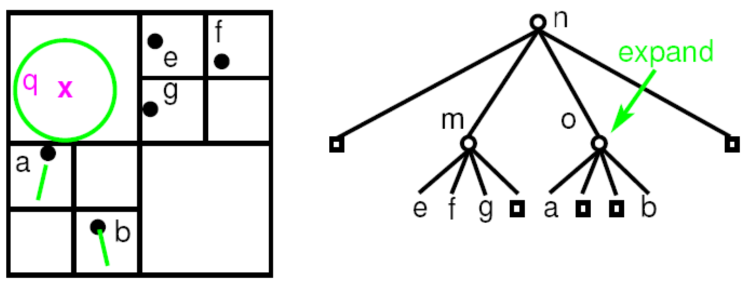
1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.



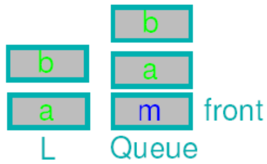
USC Viterbi School of Engineering

kNN Example

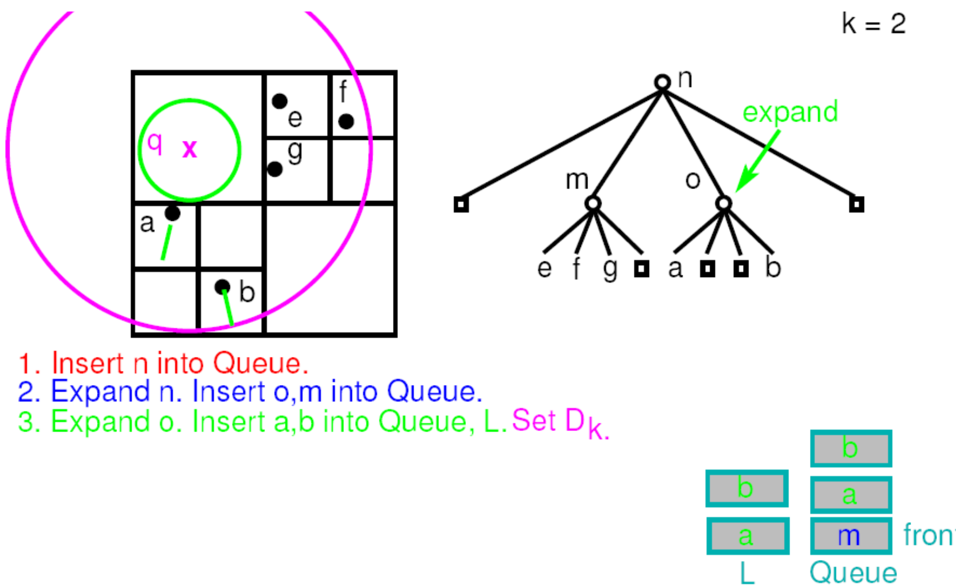
k = 2



1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L.



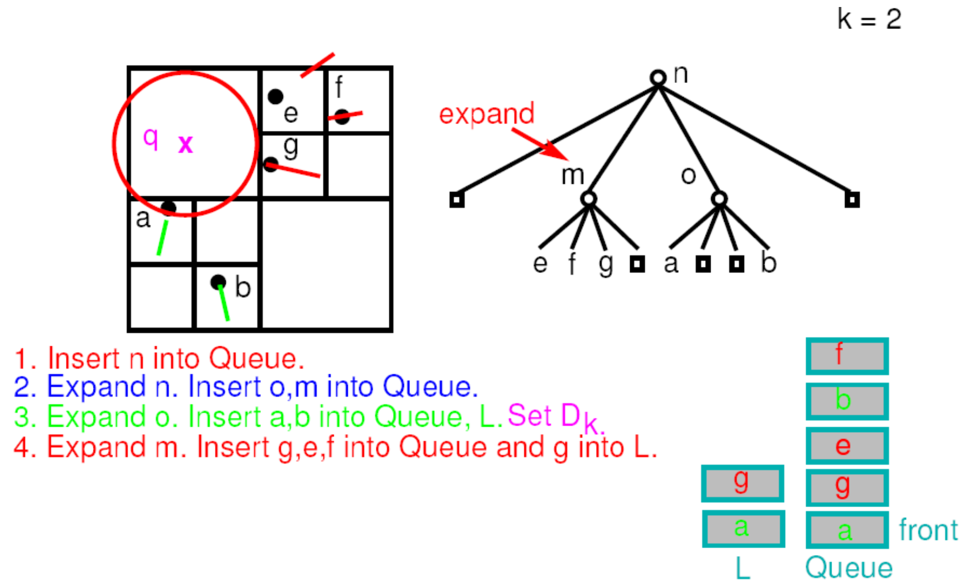
kNN Example



1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_K.

D_k is the maximum network distance of the k th nearest neighbor candidate

kNN Example



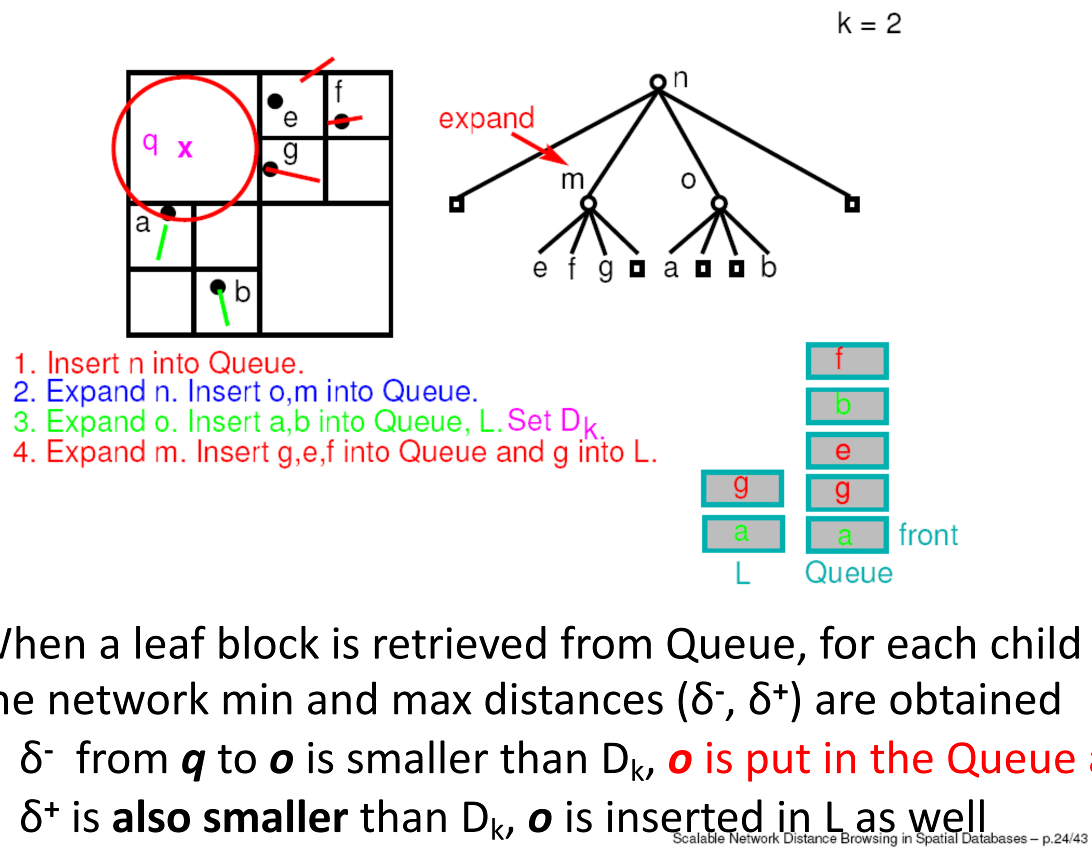
When a leaf block is retrieved from Queue, for each child object ***o*** in it, the network min and max distances (δ^-, δ^+) are obtained

If δ^- from ***q*** to ***o*** is greater than or equal to D_k, ***then*** exit and return *L* as the set of *k* nearest neighbors because *o* and all other objects in *Queue* or in blocks in *Queue* cannot be found at a distance from *q* which is less than D_k.)

Otherwise, ***o*** is put in the Queue again and

USC Viterbi School of Engineering

kNN Example

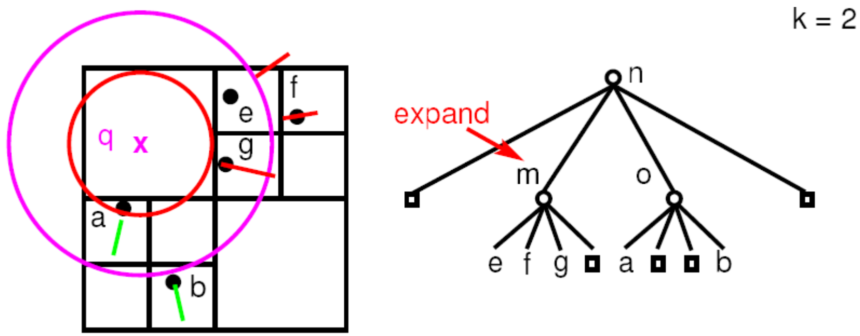


When a leaf block is retrieved from Queue, for each child object ***o*** in it, the network min and max distances (δ^-, δ^+) are obtained

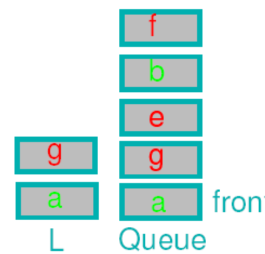
If δ^- from ***q*** to ***o*** is smaller than D_k, ***o* is put in the Queue again, and**

If δ^+ is **also smaller** than D_k, ***o*** is inserted in L as well (otherwise (i.e., $D_k \geq \delta^+$) which means that *o* is one of the *k* nearest neighbors of *q*

kNN Example

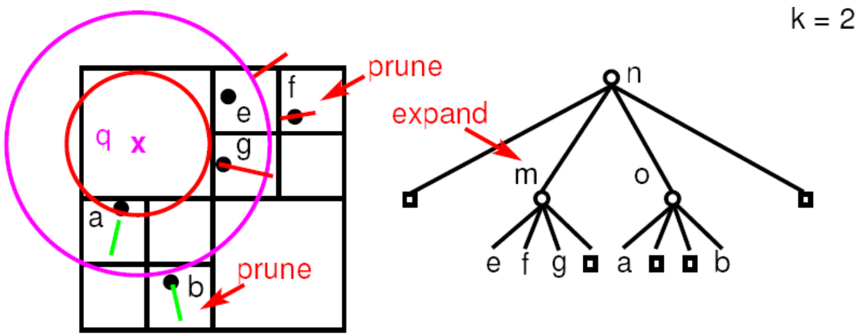


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L. Update D_k.

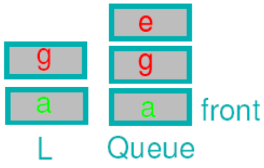


USC Viterbi School of Engineering

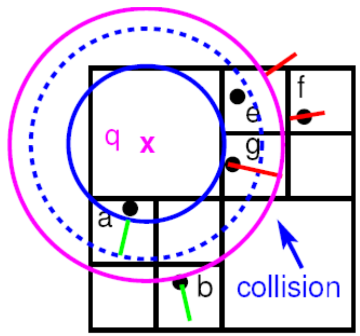
kNN Example

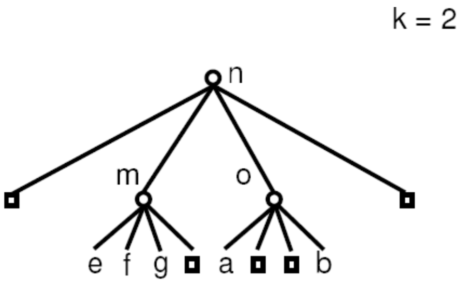


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.

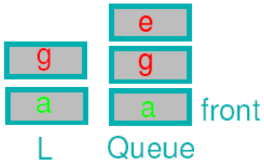


kNN Example

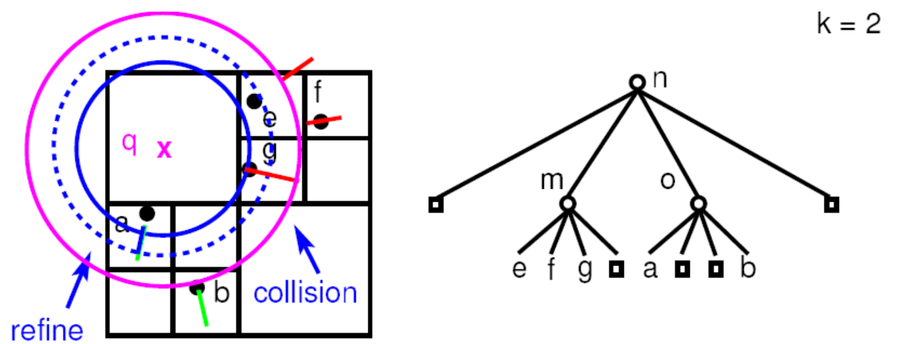




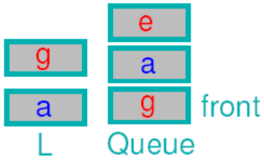
1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L. Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.



kNN Example

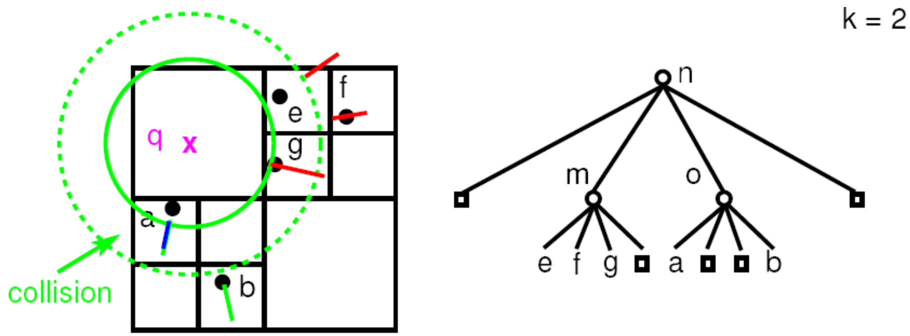


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.

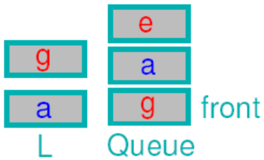


USC Viterbi School of Engineering

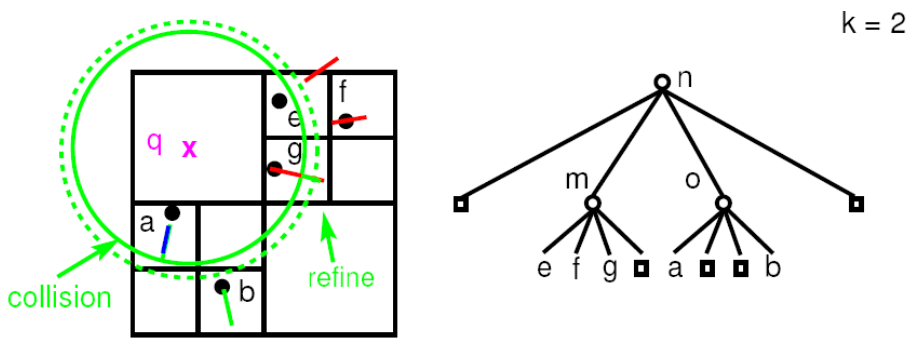
kNN Example



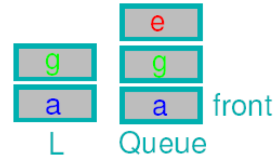
1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L. Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g. Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a.



kNN Example

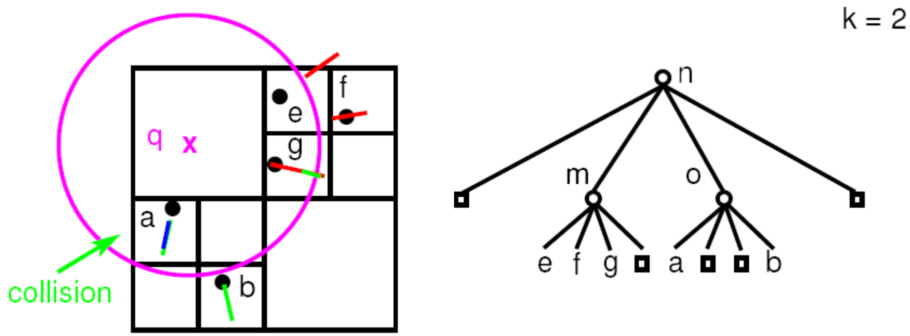


1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L. Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g. Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a. Refine and Reinsert g into Queue and L.

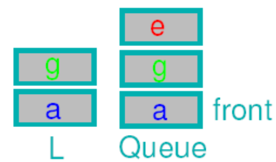


USC Viterbi School of Engineering

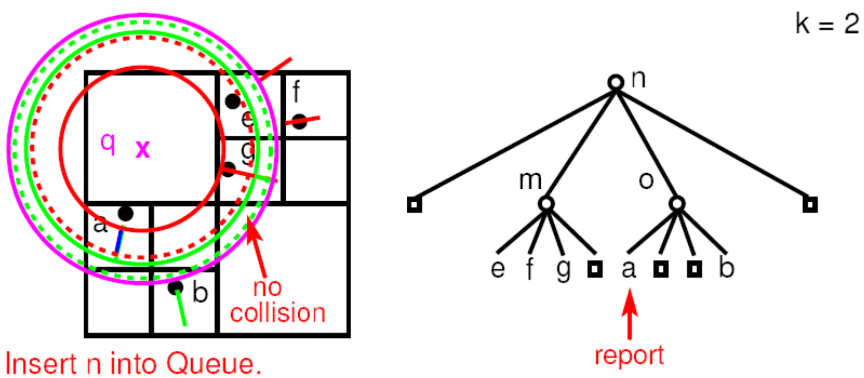
kNN Example



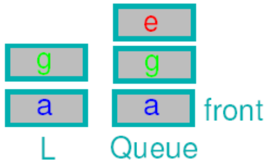
1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L. Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g. Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a. Refine and Reinsert g into Queue and L. Update D_k.



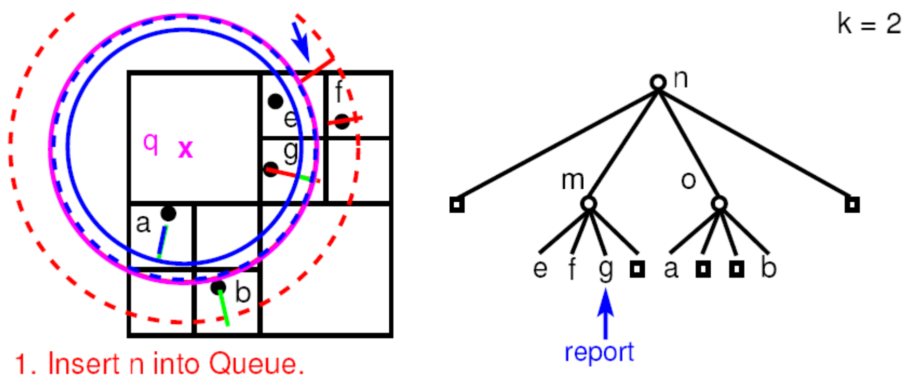
kNN Example



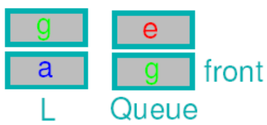
1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a.
   Refine and Reinsert g into Queue and L. Update D_k.
7. Process a. No collision of a with g. No need to refine a further.



kNN Example



1. Insert n into Queue.
2. Expand n. Insert o,m into Queue.
3. Expand o. Insert a,b into Queue, L. Set D_k.
4. Expand m. Insert g,e,f into Queue and g into L.
   Update D_k. Prune f and b from Queue.
5. Process a. Collision of a with g.
   Refine a. Reinsert a into Queue and L.
6. Process g. Collision of g with a.
   Refine and Reinsert g into Queue and L. Update D_k.
7. Process a. No collision of a with g. No need to refine a further.
8. Process g. No collision of g with e.
   No need to refine g further. Report L.

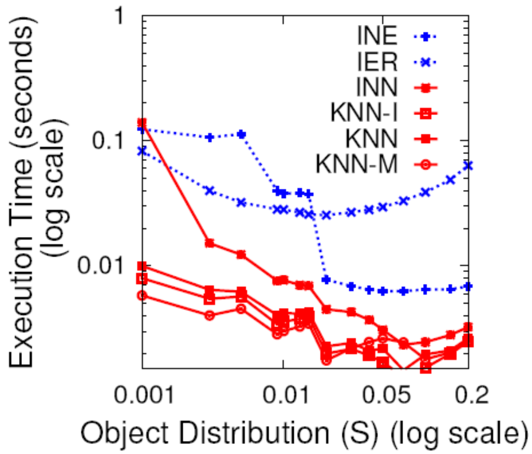


USC Viterbi School of Engineering

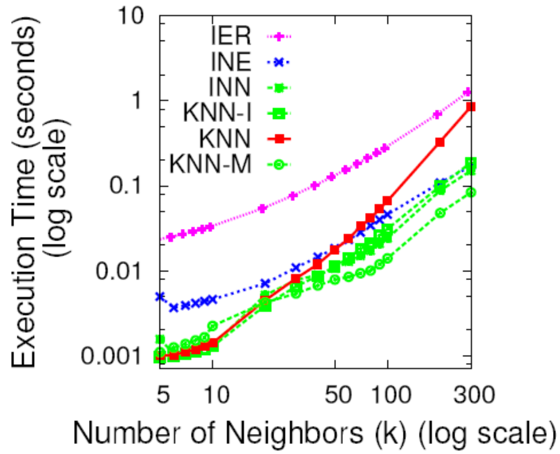
Evaluation

- Compared kNN with other algorithms including variants of kNN
  - INE: Basically Dijkstra's algorithm [Papa03]
  - IER: Using Euclidean distance as a filter [Papa03]
  - INN: Incremental variant of kNN which invokes kNN k times no priority queue, L, or D_k
  - kNN-I: Use L to calculate D_k using first *k* objects
  - kNN-M: Reduce number of renements by dropping need for total ordering k
- Test set is important roads on US eastern seaboard consisting of 91,113 vertices and 114,176 edges
- S is generated at random and stored in a PMR quadtree
- Each query run on at least 50 random input datasets of same size

Evaluation



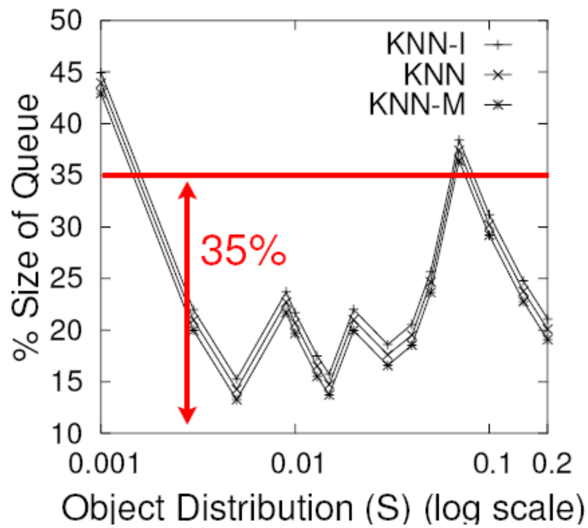
k=10 and varying sizes of S



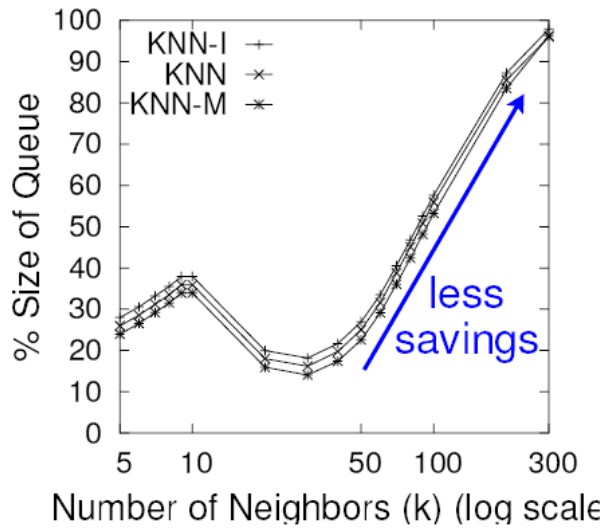
$S = 0.07N$ and varying k

- kNN and Variants are at least one order of magnitude faster than INE and IER for small values of k and moderate values of S
- INE and IER improve relatively for large values of S as easy to find k neighbors around q
- IER always slowest

Evaluation



k=10 and varying sizes of S



$S = 0.07N$ and varying k

- Compared maximum size of priority queue of kNN and variants with INN which cannot use D_k to reduce insertions
- 35% of INN on the average
- As k increases, savings in maximum queue size vanish
    - most likely due to an increase in the number of objects having overlapping distance intervals from q

Conclusion

- Very efficient shortest path and kNN computation in static spatial networks
  - Avoid applying Dijkstra's algorithm for each query which visits all vertices on the shortest path to the destination
- General framework for query processing in spatial networks
- Not restricted to nearest neighbor queries
- Transform solution from a graph-based combinatorial algorithm to a purely geometric one
- Pure database solution and hence can be integrated with DBMS architecture easily
- Reduce cost of storing shortest paths between all pairs of N vertices from $O(N^3)$ to $O(N^{1.5})$
- Scalable

USC Viterbi School of Engineering

Discussion

- Minimalist experiments: Most of the experiments focus on the variation of the proposed technique. Can compare with state of the art shortest path computation approaches
- Edge weights are assumed to be constant however in real-world edge weights are function of time
- Assumptions: Monotonic shortest path map for each vertex? Can the shortest path map be non-monotonic?

References

- Hanan Samet, Jagan Sankaranarayanan, Houman Alborzi: Scalable network distance browsing in spatial databases. SIGMOD Conference 2008: 43-54
- A presentation by Ugur Demiryurek in csci587 Fall’2010