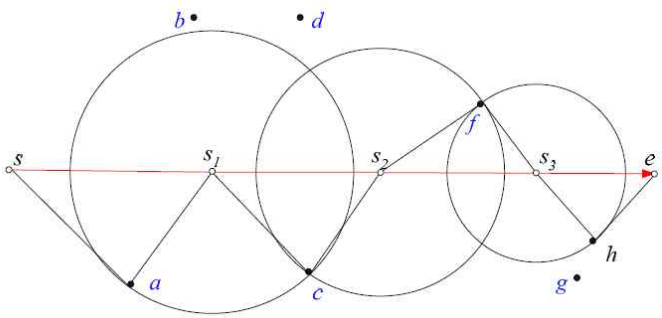
CSE6488: Mobile Computing Systems

Continuous Nearest Neighbor Query Processing using R-tree

Sungwon Jung
Data Engineering and Mobile Computing Lab.
Department of Computer Science and Engineering
Sogang University
Seoul, Korea
Email: jungsung@sogang.ac.kr

Problem: Continuous Nearest Neighbor

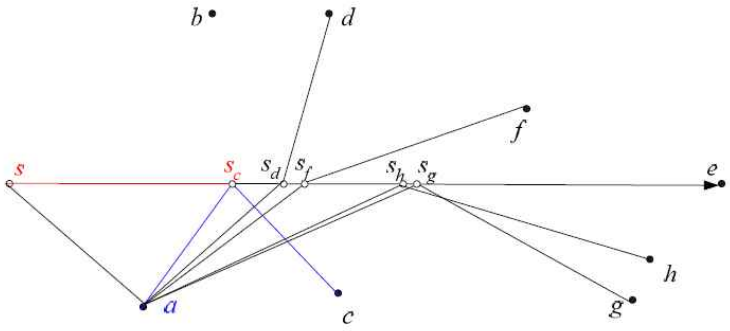


Data: A set of points
Query: A line segment q=[s, e]
Result: The nearest neighbor (NN) of *every* point on q.
Result representation: {s(.NN=a), s_1(.NN=c), s_2(.NN=f), s_3(.NN=h), e}

For the sake of simplicity we present Continuous 1-NN, while the solution generalizes to k-NN, and trajectories of multiple line segments (see paper).

2

Previous Approach-Time Parameterized Queries



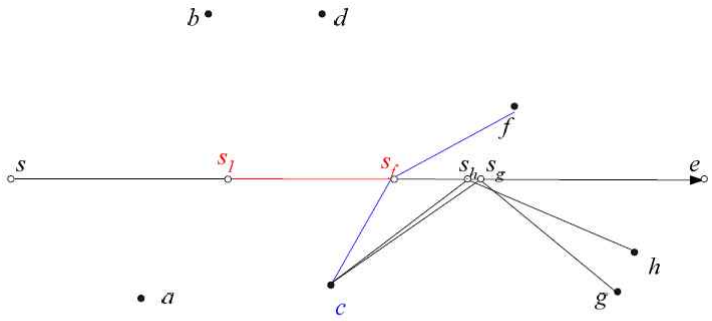
Step 1: Find the NN of the start point s, i.e., point a.

Step 2: Use the TP technique to find:

The first point on the line segment (s_c) where there is a change in the NN (i.e., point c) will become the next NN.

3

TP NN (cont'd)



From Step 2 we have decided the next NN change is point c at s_1

Step 3: Perform another TP NN to find:

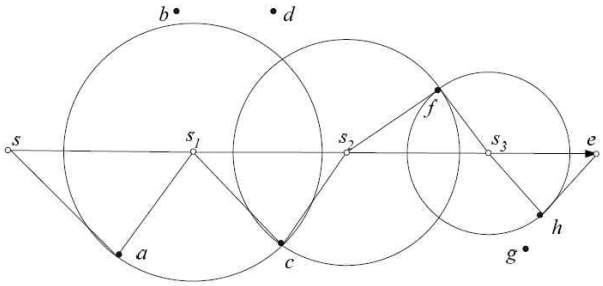
Starting from s_1, how far we need to travel for the current NN (i.e., c) to change.

Repeat this until we finish the entire segment.

Problem: # of TP queries = # of NN changes (i.e., output sensitive)

4

Our Goal



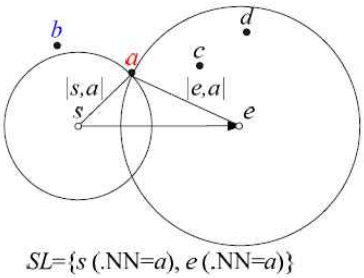
Find all *split points* s_1, s_2, s_3 (as well as the corresponding NN for each partition) *with a single traversal of the dataset.*

Term1: The set of split points (including s and e) constitute the *split list*.
Term2: The circle that centers at split point s_i with radius $dist(s_i, s_i.NN)$ is the *vicinity circle* of s_i.
Term3: We say a data point u *covers* a point s if u=s.NN. E.g., points a, c, f, h *cover* segments $[s, s_1]$, $[s_1, s_2]$, $[s_2, s_3]$, $[s_3, e]$.

5

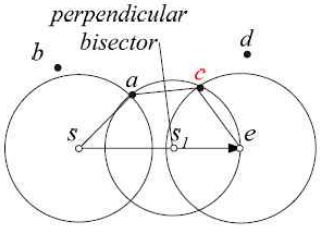
Lemma 1

Given a split list SL $\{s_0, s_1, \ldots, s_{|SL-1|}\}$, and a new data point p, then: p covers some point on query segment q if and only if p covers a split point.



$SL=\{s$ (.NN=a), e (.NN=a)$\}$

After processing a



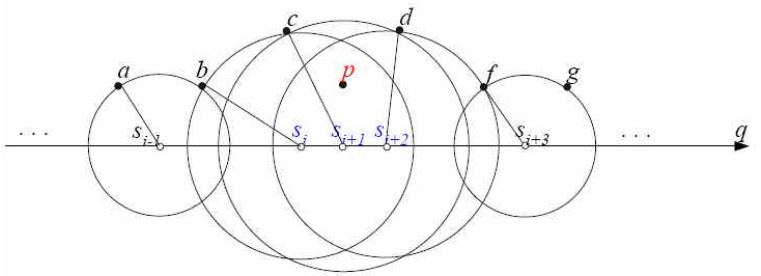
$SL=\{s$ (.NN=a), s_1 (.NN=c), e (.NN=c)$\}$

After processing c

6

Lemma 2 (Covering Continuity)

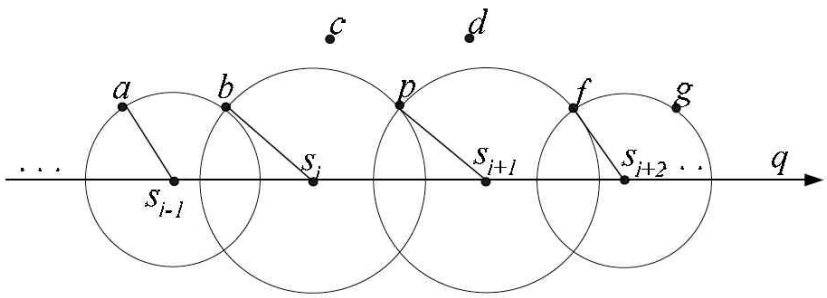
The split points covered by a point p are continuous.
Namely, if p covers split point s_i but not s_{i-1} (or s_{i+1}), then p cannot cover s_{i-j} (or s_{i+j}) for any value of $j>1$.



$SL=\{s_{i-1}$ (.NN=a), s_i (.NN=b), s_{i+1} (.NN=c), s_{i+2} (.NN=d), s_{i+3} (.NN=f)$\}$

7

Lemma 2 (Covering Continuity)



$SL=\{s_{i-1}$ (.NN=a), s_i (.NN=b), s_{i+1} (.NN=p), s_{i+2} (.NN=f)$\}$

After p is processed

8

Algorithm with R-trees Overview

Use branch-and-bound techniques to prune the search space.

When a leaf entry (i.e., a data point) p is encountered

SL is updated if p covers any split point (i.e., p is a *qualifying entry*) – By Lemma 1.

For an intermediate entry

We visit its subtree only if it may contain any qualifying data point – Use heuristics.

9

Heuristic 1

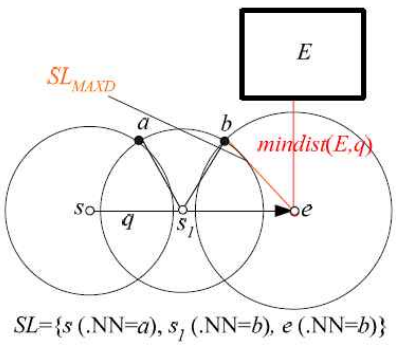
Given an intermediate entry E and query segment q, the subtree of E may contain qualifying points only if

$mindist(E,q) < SL_{MAXD}$,

where

$mindist(E,q)$ denotes the minimum distance between the MBR of E and q

SL_{MAXD} is the maximum distance between a split point and its NN.

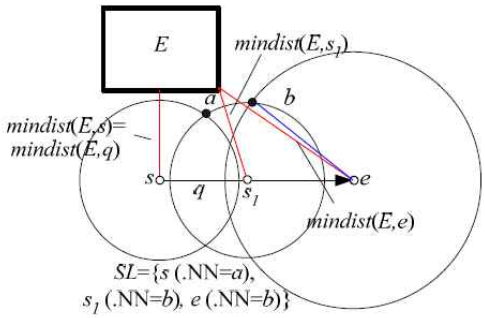


$SL=\{s$ (.NN=a), s_1 (.NN=b), e (.NN=b)$\}$

10

Heuristics 2

Given an intermediate entry E and query segment q, the subtree of E must be searched *if and only if*
there exists a split point $s_i \in$ SL such that $dist(s_i, s_i.NN) > mindist(s_i, E)$.

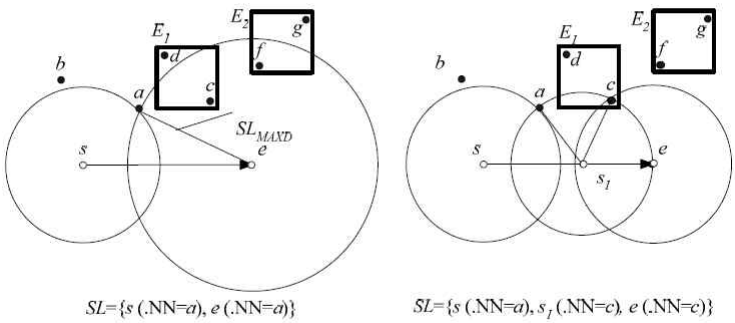


Heuristic 2 requires mindist computation between E and all split points. Hence it is applied only if E passes heuristic 1, which requires only one computation.

11

Heuristics 3 (Access Order)

Entries (satisfying heuristics 1 and 2) are accessed in increasing order of their minimum distances to the query segment q.



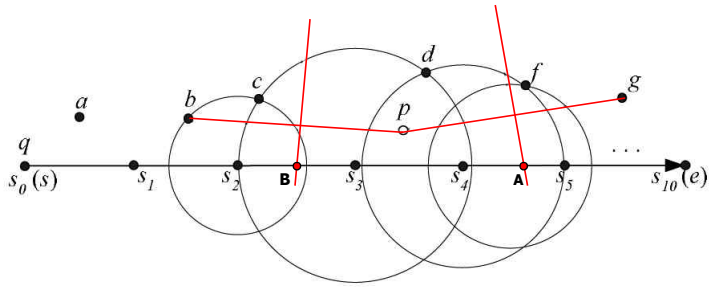
12

CNN Algorithms with R-trees

- When a leaf entry(i.e., a data point) p is encountered,
  1) Retrieves the set of split points $S_{COVERS}=\{s_i, s_{i+1}, \ldots, s_j\}$ covered by p.
  2) Updates SL accordingly if S_{COVERS} is not empty
- Use binary search
  - To avoid comparing p with all current NN for every split point.

Find the split points covered immediately before and after s3.

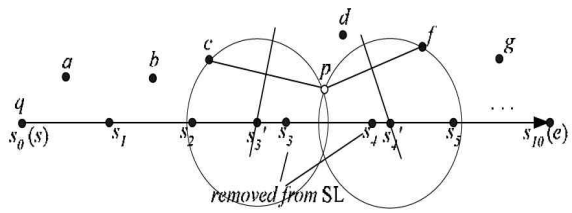
$S_{COVERS}=\{s_3, s_4\}$



12

CNN Algorithms with R-trees

- When a leaf entry(i.e., a data point) p is encountered,
  1) Retrieves the set of split points $S_{COVERS}=\{s_i, s_{i+1}, \ldots, s_j\}$.
  2) Updates SL



Algorithm Handle_Leaf_Entry
/*p: the leaf entry being handled, SL: the split list*/

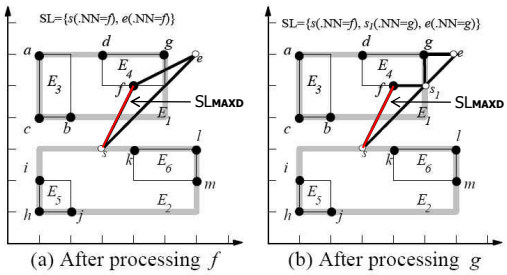
1. apply binary search to retrieve all split points covered by p: $S_{COVER}=\{s_i, s_{i+1}, \ldots, s_j\}$ → **Scover={s3, s4}**
2. let $u=s_{i-1}$.NN and $v=s_j$.NN → **u = s2, v = s4**
3. remove all split points in S_{COVER} from SL → **Remove s3, s4 from SL**
4. add a split point s_i' at the intersection of q and $\perp(u, p)$ with s_i'.NN=p, $dist(s_i', s_i'.NN)=|s_i', p|$ → **Add s3' at the intersection of q and ⊥(s2, p)**
5. add a split point s_{i+1}' at the intersection of q and $\perp(v, p)$ with s_{i+1}'.NN=p, $dist(s_{i+1}', s_{i+1}'.NN)=|s_{i+1}', p|$ → **Add s4' at the intersection of q and ⊥(s4, p)**

End Handle_Leaf_Entry

13

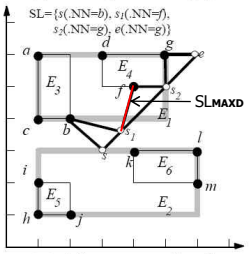
CNN Algorithms with R-trees

- Example of the CNN algorithm using depth-first traversal on the R-tree

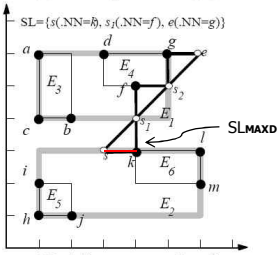


(a) After processing f

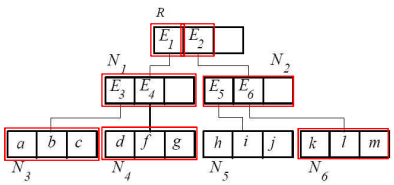
(b) After processing g



(c) After processing b



(d) After processing k



Result : {<k, [s,s_1]>, <f,[s_1,s_2]>, <g,[s_2,e]>}

14