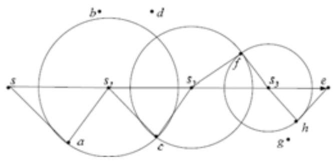


Continuous Nearest Neighbor Search

Yufei Tao, Dimitris Papadias, Qiongmao Shen
Hong Kong University of Science and Technology

Presented : Penny Bei Pan

Conferences

Short Name	Full Name
SIGMOD	Special Interest Group on Management Of Data
VLDB	Very Large Data Base
ICDE	International Conference on Data Engineering

2

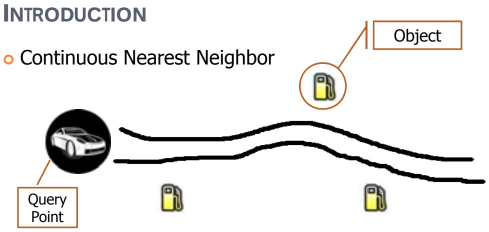
Overview

- Introduction
- Preliminary & Related Work
- Continuous k-Nearest Neighbor Query(CkNN)
  - Definition
  - Problem Characteristics
  - R-tree algorithm
  - Query analysis
  - Complex CNN extension
- Experiments
- Discussion and Conclusion

3

Introduction

- Continuous Nearest Neighbor

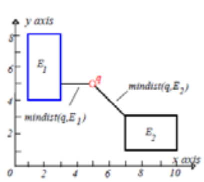


- Why called "continuous"?
  - Nearest neighbor of every points in the trajectory

4

Preliminary -- Point NN Queries

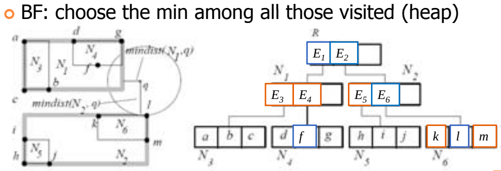
- Branch and bound algorithms use *mindist* between the query point *q* and an R-tree entry *E*, to prune the search space:
  - – *mindist(E, q)* = The minimum distance between *E* and *q*



5

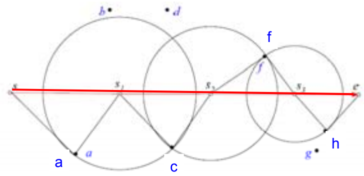
Preliminary -- Point NN Queries

- *Depth-first* (DF) and *Best-first* (BF) algorithms
  - E: R-tree entry
  - q: query point
- DF : choose the entrance with minimum min-dist
- BF: choose the min among all those visited (heap)



6

Preliminary -- Continuous Nearest Neighbor



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

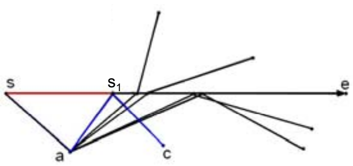
7

Related work – Sampling

- Try to convert the continuous-NN to point-NN
  - Every point on the line -> unlimited points
  - Sampling
- Drawback:
  - Sample Rate: low -> incorrect
  - Sample Rate: high -> overhead (still cannot guarantee accuracy)
- Time Parameterized queries
  - Output (R, T, C) : result, time period, changing point
  - Tao, Y., Papadias, D. Time Parameterized Queries in Spatio-Temporal Databases. ACM SIGMOD, 2002.

8

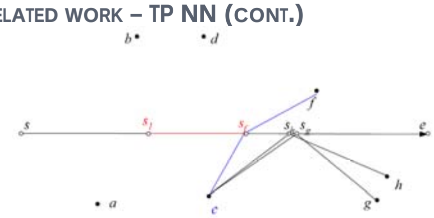
Related work – Time Parameterized NN



- 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_1*) where there is a change in the NN (i.e., point *c*) will become the next NN

9

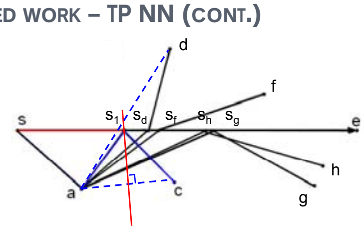
Related work – TP NN (cont.)



- Step 3: Perform another TP NN to find:
- Starting from s1, how far we need to travel for the current NN (i.e., *c*) to change to *f*.
- Repeat this until we finish the entire segment.

10

Related work – TP NN (cont.)

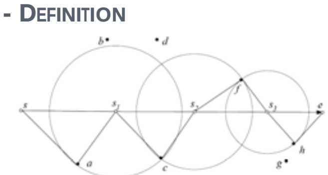


- Intuitively: perpendicular bisector & [s,e] segment
- Not only NN, but support k-NN
- Still overhead: n times

Yufei Tao , Dimitris Papadias ⇨ TP

11

CkNN - Definition

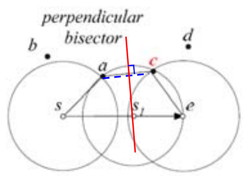


- Goal: Find all split points(as well as the corresponding NN for each partition) with a single traversal.
- Split list: The set of split points (including s and e).
- Vicinity circle: The circle that centers at split point s_i with radius dist(s_i, s_i.NN)
- We say a data point u covers a point s if u=s.NN. E.g., points a, c cover segments [s, s1], [s1, s2]

12

CKNN – Problem Characteristics

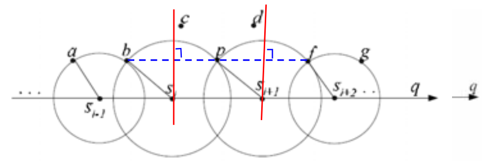
- Lemma 1: Given a split list SL $\{s_0, s_1, \ldots, s_{|SL-1|}\}$, and a new data point p, then: p covers some point on query segment q if and only if p covers a split point.



13

CKNN - Problem Characteristics

- 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.

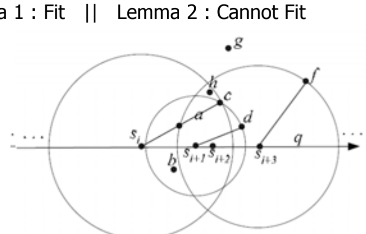


$SL=\{s_{i-1}$ (.NN=a), s_i (.NN=b), s_{i+1} (.NN=p), s_{i+2} (.NN=f)\}

14

CKNN - Problem Characteristics

- How about the k-NN?
- Lemma 1 : Fit || Lemma 2 : Cannot Fit
- Eg:
  - K=3



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

15

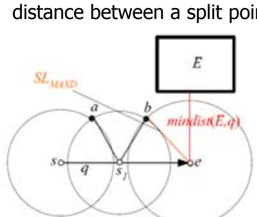
CKNN – R-tree Algorithm

- General key notes:
  - Use branch-and-bound techniques to prune the search space.
  - R-tree traverse principle:
    - 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.
  - Avoid accessing not qualified nodes

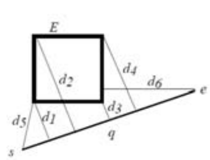
16

R-tree Algorithm – Heuristic 1

- Given an intermediate entry *E* and query segment *q*, the sub-tree of *E* may contain qualifying points only if mindist(E,q) < SL_{MAXD}, where SL_{MAXD} is the maximum distance between a split point and its NN.



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

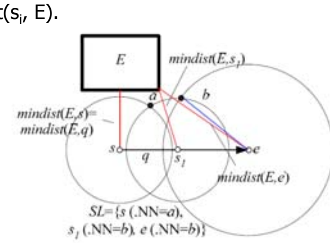


Compute Mindist(E,q)

17

R-tree Algorithm – Heuristic 2 (after 1)

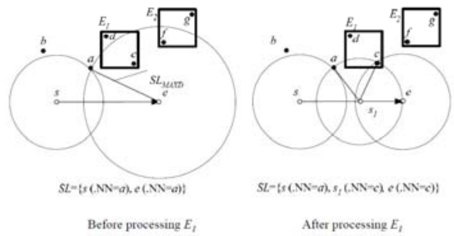
- Given an intermediate entry *E* and query segment *q*, the subtree of *E* must be searched if and only if there exists a split point $s_i \in SL$ such that dist(s_i, s_i.NN) > mindist(s_i, E).



18

R-tree algorithm – Heuristic 3 (Order)

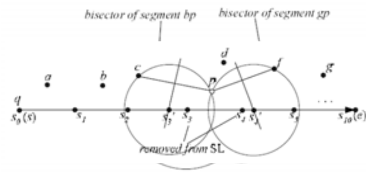
- Entries (satisfying heuristics 1 and 2) are accessed in increasing order of their minimum distances to the query segment *q.*



19

R-tree algorithm – Leaf entry

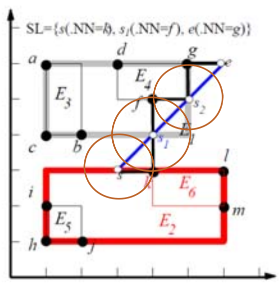
- Input: New entry *p*, SL $=\{s_1,\ldots s_{10}\}$
  - 1) retrieve the split points covered by p
  - 2) update SL
- Binary search: Start at s_5, then s_2…
- Using bisector to judge the direction



20

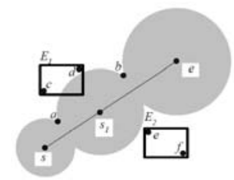
CkNN – R-tree algorithm (example)

- Depth First

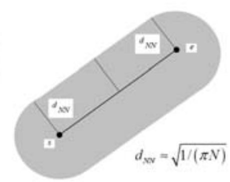


21

Analysis- Cost Model for Uniform Data



Actual Search region



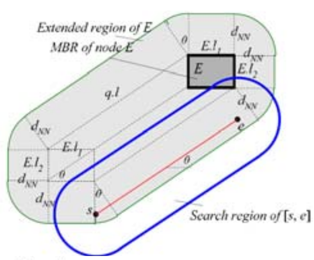
Approximate Search region

- An optimal algorithm on R-trees must access only those nodes whose MBRs intersect the actual search region (i.e., E1 but not E2).
- To facilitate the analysis we focus on a more regular (approximated) region

22

Analysis – Node Access Probability

- P_{ACCESS} is the probability the MBR *E* of a node Intersects the search region



$$P_{ACCESS}(E,q) = area(E_{EXT}) = \pi d_{NN}{}^2 + E.l_1 \cdot E.l_2 + 2d_{NN}(E.l_1 + E.l_2 + q.l) + 2q.l(E.l_1 \cdot |\cos\theta| + E.l_2 \cdot |\sin\theta|)$$

23

Analysis – Cost Model (Node Access)

$$NA(CNN) = \sum_{i=0}^{h-1} N_i \cdot P_{ACCESS}(E.l_i, q)$$
$$= \sum_{i=0}^{h-1} N_i \cdot \left[\pi d_{NN}{}^2 + E.l^2 + 2 \cdot d_{NN}(2 \cdot E.l + q.l) + 2 \cdot q.l \cdot E.l(|\cos\theta| + |\sin\theta|) \right]$$

- Dataset cardinality N
- R tree structure (Height: h)
- The query length: q.l
- The orientation angle θ

24

Analysis – Cost Model (cont.)

$$n_{NN} = N \cdot area(R_{SEARCH}) = N\left(\pi d_{NN}^2 + 2d_{NN} \cdot q.l\right)$$

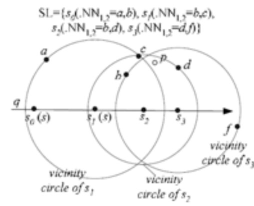
$$d_{NN} \approx \sqrt{1/(\pi N)}$$

- The number of distinct neighbors in the final result.
- CPU overhead comparison
  - TP: increase with n_{NN}
  - This paper: increase with dataset size N, query length l…

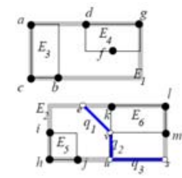
Other CNN query

- kCNN query (k=2)
  - Updating Vicinity circle

$$d_{k-NN} \approx \sqrt{k/(\pi N)}$$



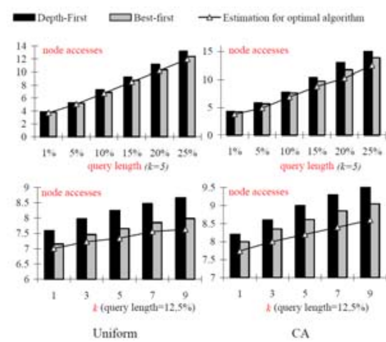
- Trajectory NN query (TNN)
  - q1 = [s,u]
  - q2 = [u,v]
  - q3 = [v,e]
  - Each segment has a SL
  - Treated one by one



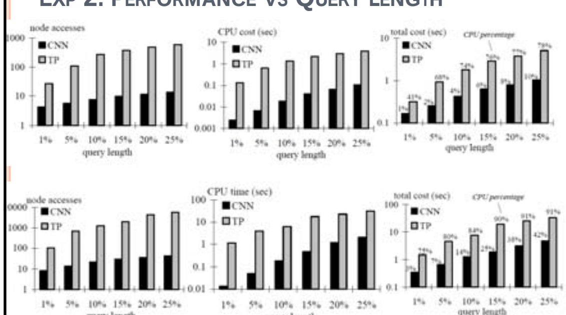
Experiments

- Datasets:
  - Uniform
  - Real street segments: CA (130K points), ST (2M points).
- Queries (each a segment):
  - Location and orientation randomly generated
  - Length is set as a parameter
- Performance is measured as the average of running 200 queries.
- Machine:
  - 1Ghz CPU, 256M memory
  - Page size=4K (R-tree node capacity=200)
- Compare CNN and TP (the only existing solution)

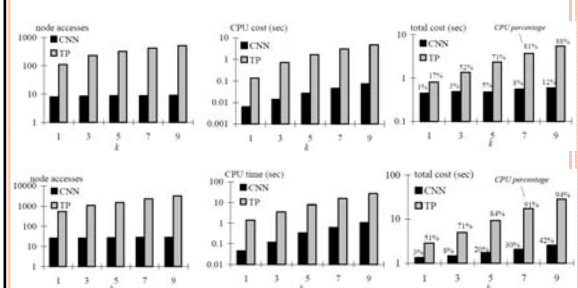
Exp 1: Cost Model Evaluation



Exp 2: Performance vs Query Length



Exp 3: Performance vs k



EXPERIMENTS – KEY NOTES

- In general, CNN outperform TP significantly
  - Single traversal
- For cost model:
  - BF better than DF (consistent with previous work)
  - The cost model is accurate
- Performance & query Length
  - Length increase, split points increase
  - CPU for TP: keep repeat retrieving the same objects
- Performance & k
  - For CNN: k has not much influenced on NA, but k influences CPU: higher number of split points

31

DISCUSSION AND CONCLUSION

- A fast algorithm for C-*kNN query.*
- Future work:
  - Rectangle data
  - Moving data points
  - Application to road networks (i.e., travel instead of Euclidean distance)

Thank you!

32