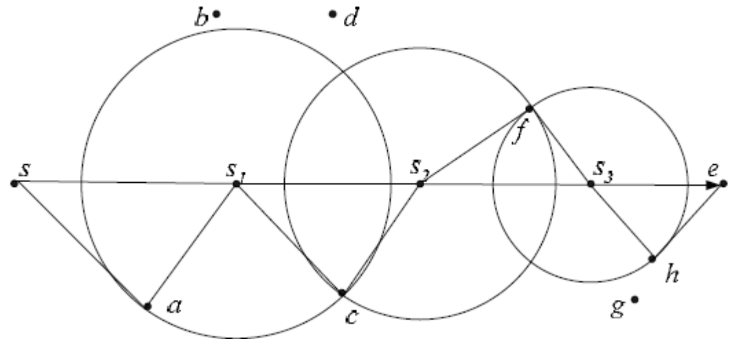
USC Viterbi
School of Engineering



CONTINUOUS NEAREST NEIGHBOR SEARCH

Instructor: Cyrus Shahabi

USC Viterbi
School of Engineering

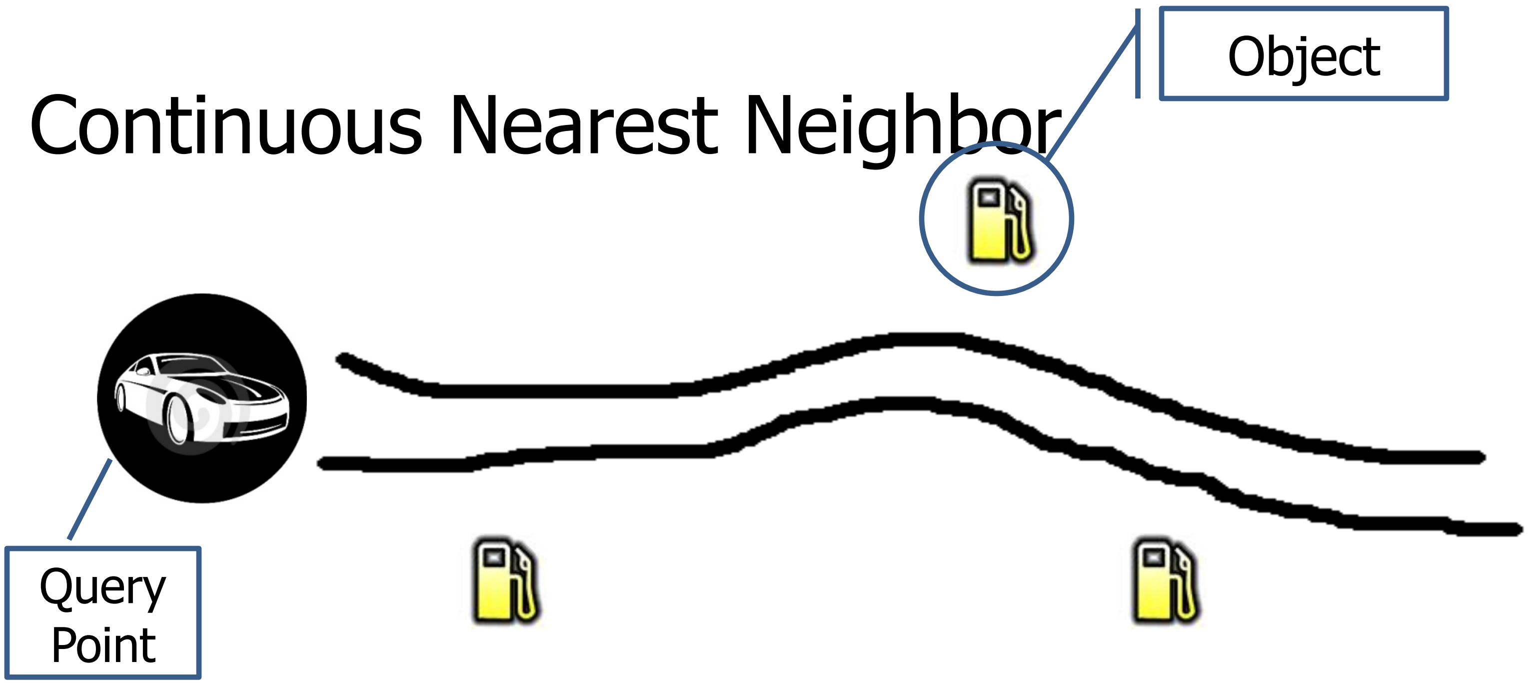
OVERVIEW

- 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

USC Viterbi School of Engineering

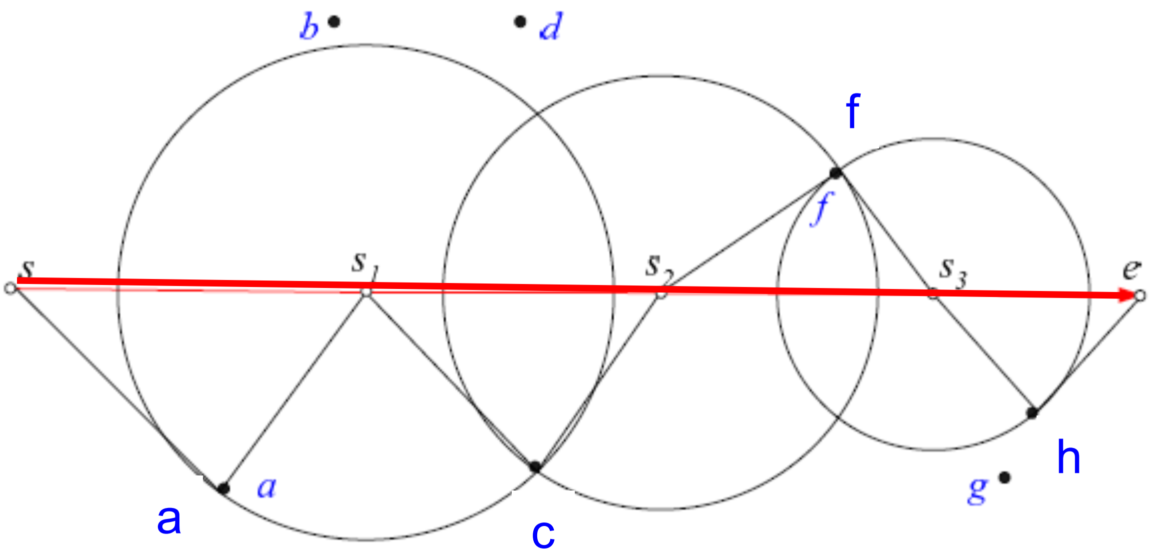
INTRODUCTION

- Continuous Nearest Neighbor



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

PRELIMINARY -- CONTINUOUS NEAREST NEIGHBOR



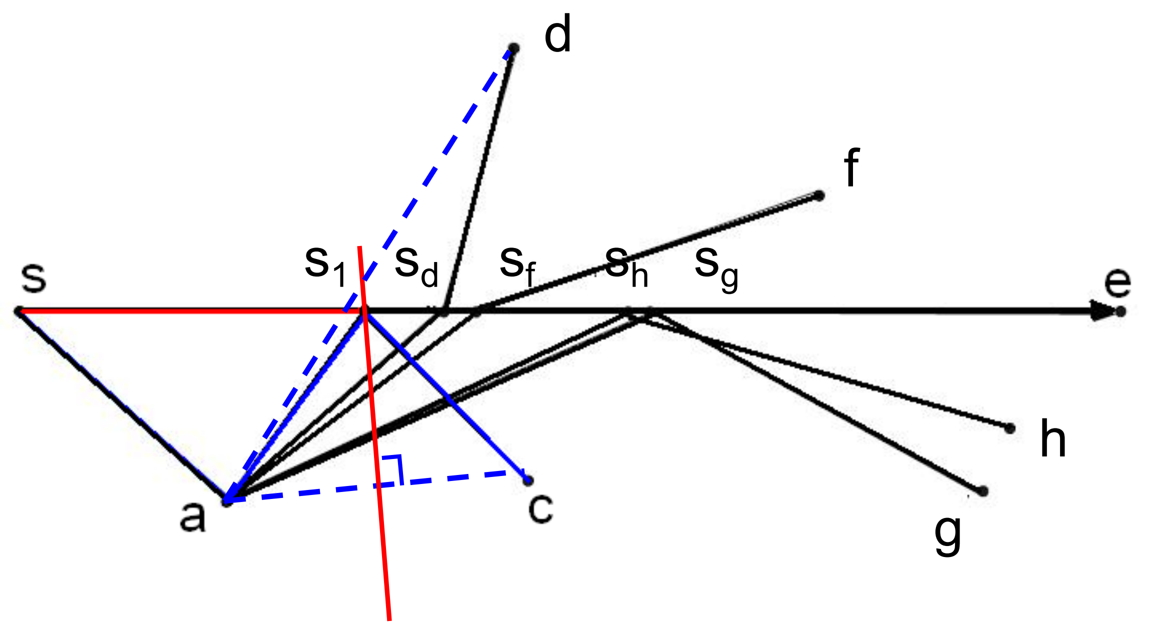
- Data: A set of points (P={*a,b,c,d,f,g,h*})
- Query: A line segment q=[s, e]
- Result: The nearest neighbor (NN) of every point on q.
- Result representation: {<a,[s,s_1]>, <c,[s_1,s_2]>, <f,[s_2,s_3]>, <h, [s_3,e]>}

USC Viterbi School of Engineering

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)

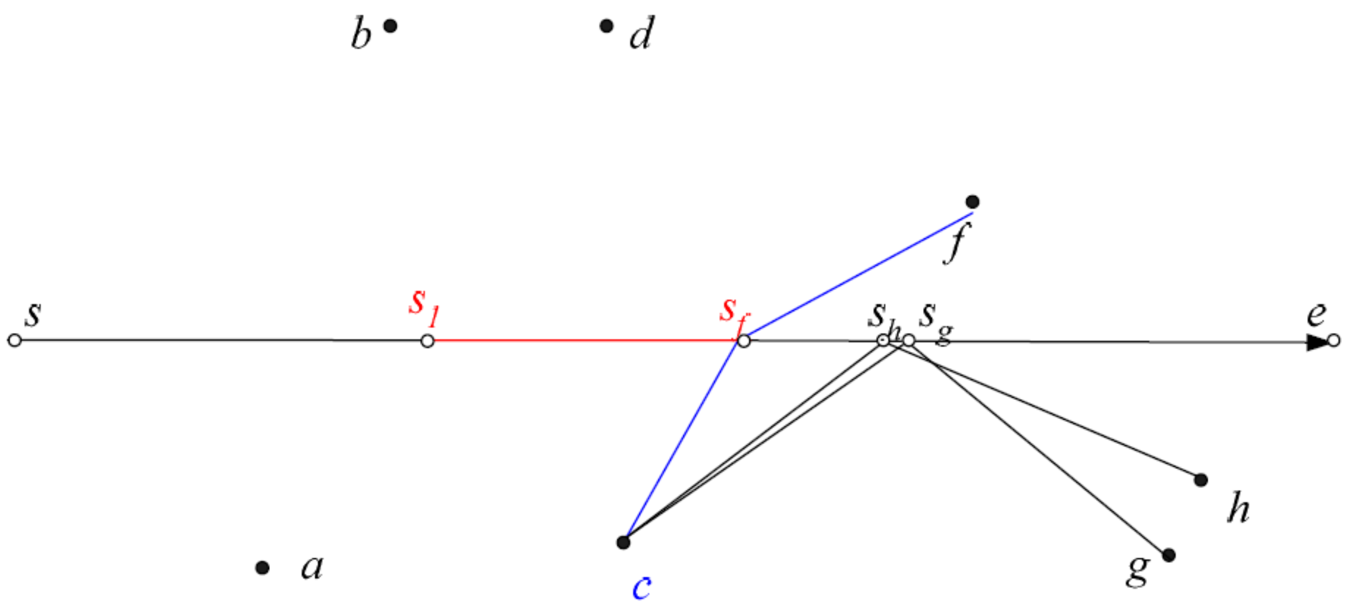
RELATED WORK – TP NN (CONT.)



- Step 1: Find the NN of the start point *s*, i.e., point *a*.
- Step 2: Use the TP technique: 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 – result: <*a*, [*s*,s_1), *c*>
- Can be thought as conventional NN query, where the goal is to find the point *x* with the minimum *dist*(*s*,*sx*)

USC Viterbi School of Engineering

RELATED WORK – TP NN (CONT.)

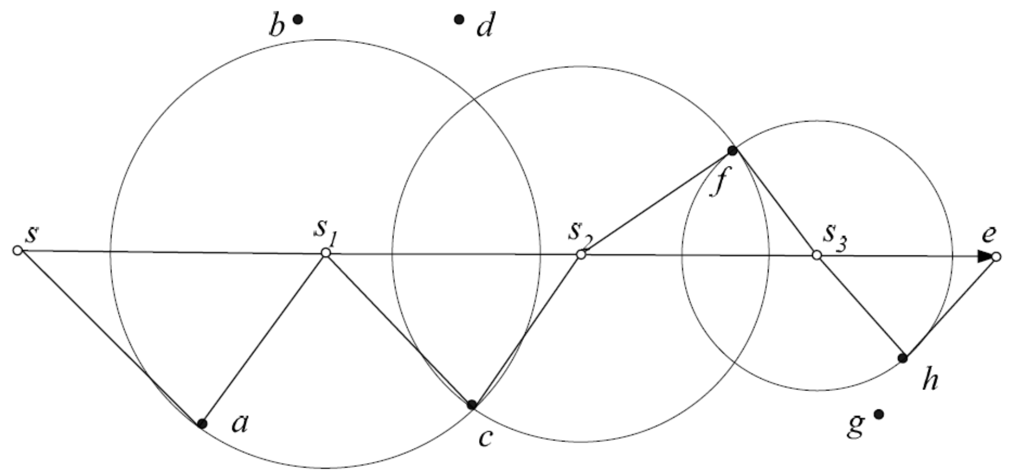


- Step 3: Perform another TP NN to find:
- Starting from s1, the smallest distance we need to travel for the current NN (i.e., *c*) to change
- Repeat this until we finish the entire segment.

RELATED WORK – TP NN (CONT.)

- Not only NN, but support k-NN
- Still overhead: n (= split points) times NN queries, multiple scans of database

CKNN - DEFINITION



- Goal: Find all split points (as well as the corresponding NN for each segment) with a single traversal.
- Split List (SL): The set of split points (including s and e).
- Each split point $s_i \in SL$ and all points in $[s_i, s_{i+1}]$ have the same NN, denoted as s_i.NN (e.g., s_1.NN is *c*, which is also the NN for all points in interval $[s_1, s_2]$)
- s_i.NN (e.g., *c*) *covers* point s_i (s_1) and interval $[s_i, s_{i+1}]$ ($[s_1, s_2]$).
- Vicinity Circle (VC): The circle that centers at split point s_i with radius dist(s_i, s_i.NN)

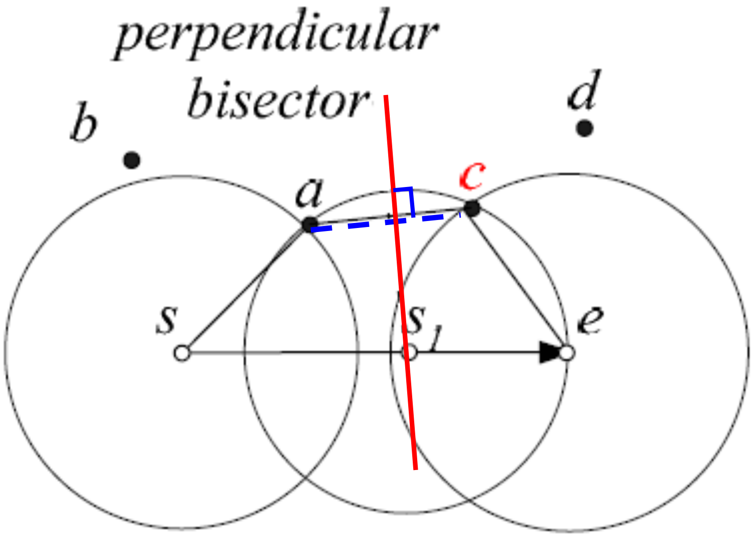
USC Viterbi School of Engineering

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.

Analyzing the first data point "a" (in alphabetical order)

Analyzing "b": not in VC of s and e, hence no point on [s,e] closer to b than a



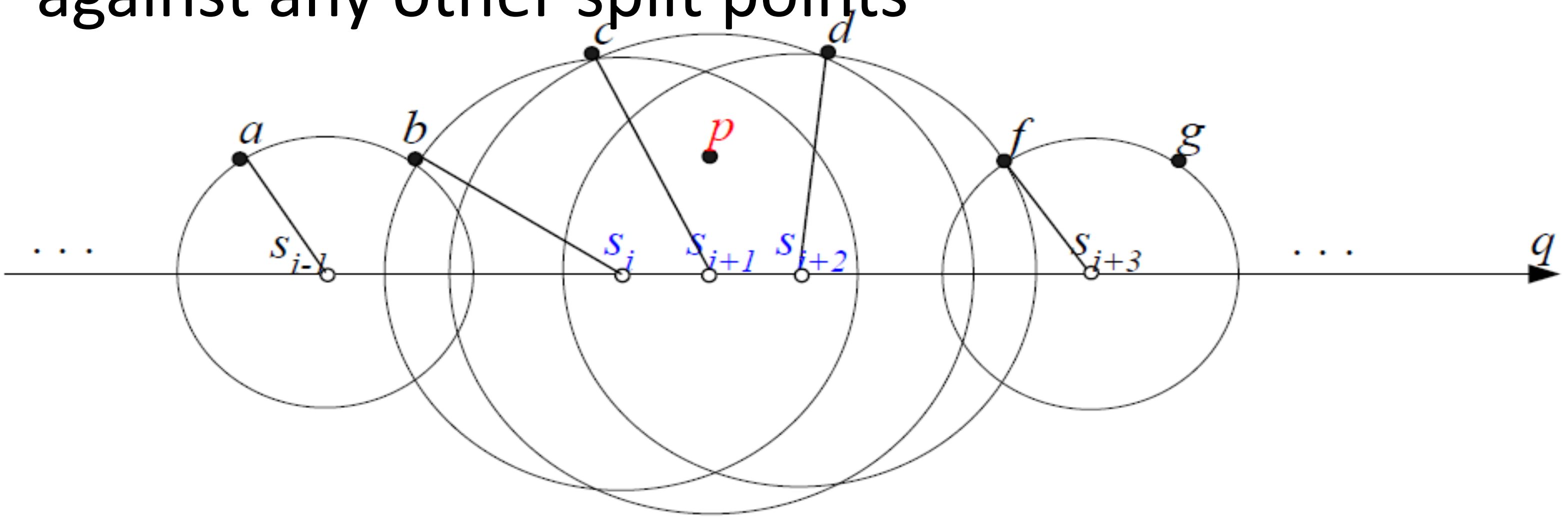
Analyzing "c": in VC of e, hence: Creating a new split point…

d not in any VC (note that it was in VC of e before adding c)

Result: {<*a*, [*s*,*s1*]>, <*c*, [*s1*,*e*]> }

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.
  - Below: p cover Si, Si+1 and Si+2 (*p* falls in their vicinity circles), but not *si-1*, *si+3*, so no need to check p against any other split points



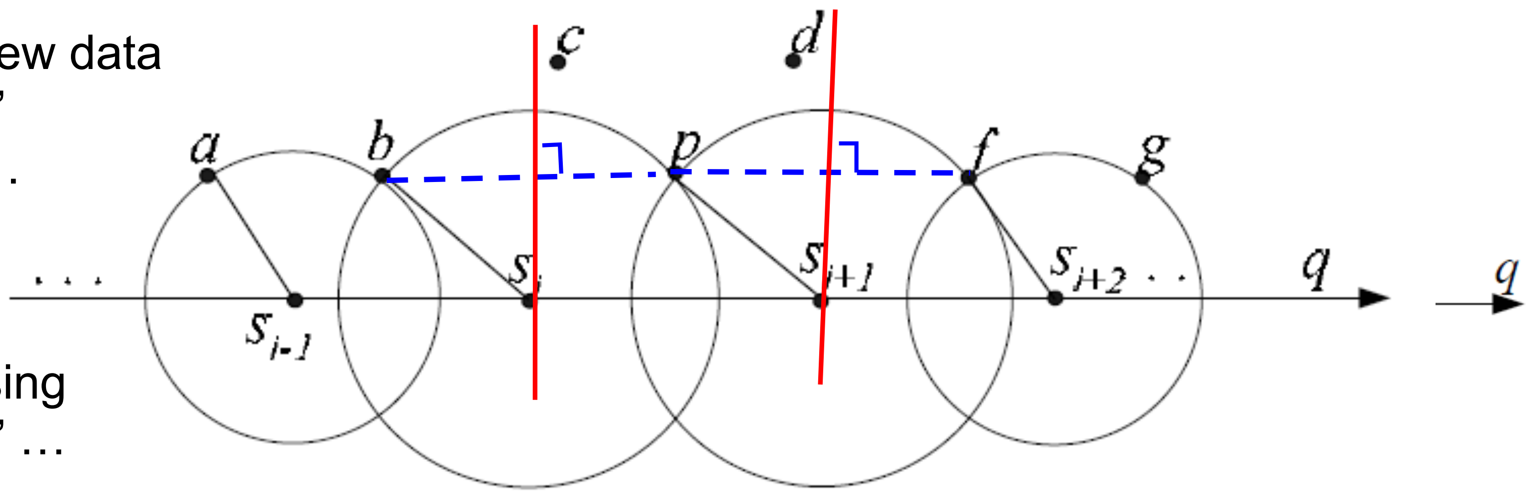
$SL=\{s_{i-1}$ (.NN=a), s_i (.NN=b), s_{i+1} (.NN=c), s_{i+2} (.NN=d), s_{i+3} (.NN=f)$\}$

CKNN - PROBLEM CHARACTERISTICS

- Finding new split points for b (after adding p):
  - b covers S_{i-1} and f covers S_{i+2}; so we need to find the space that NN changes from b to p and then to f

When new data point “p” arrives…

Processing point “p” …

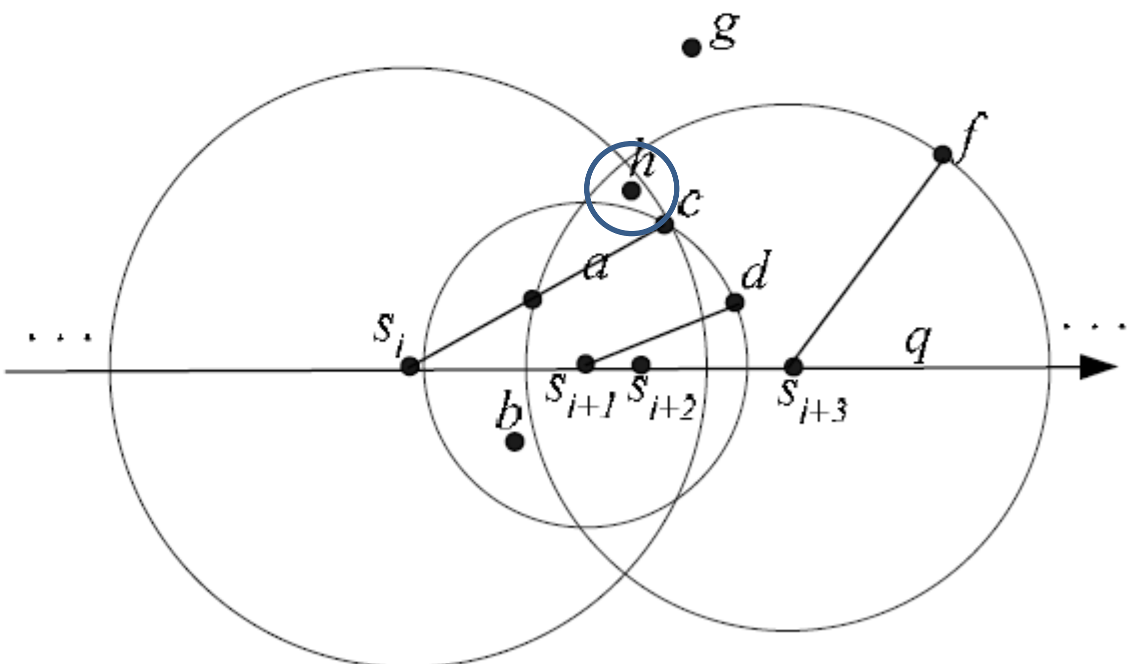


$SL=\{s_{i-1}$ (.NN=a), s_i (.NN=b), s_{i+1} (.NN=p), s_{i+2} (.NN=f)$\}$

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

CKNN - PROBLEM CHARACTERISTICS

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



h covers s_i, s_{i+3}
But not s_{i+1}, s_{i+2}

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

USC Viterbi School of Engineering

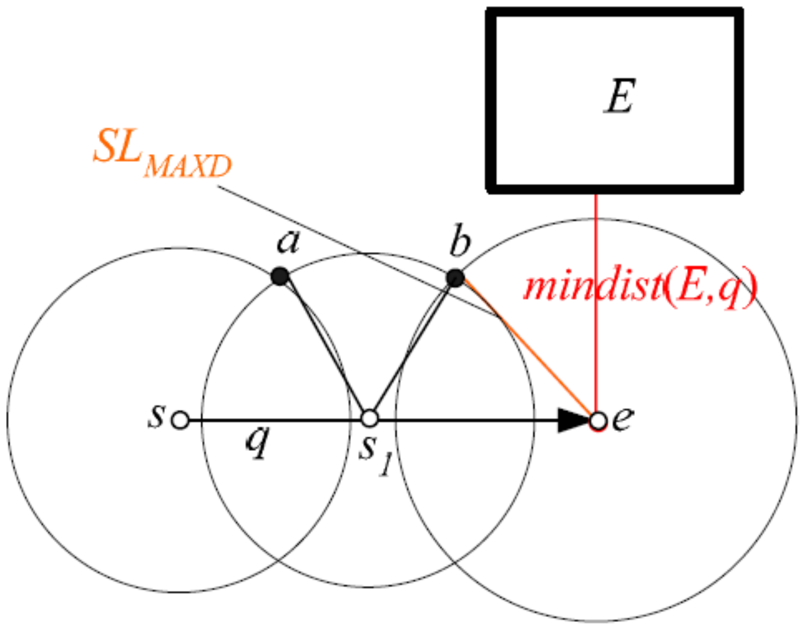
CKNN – R-TREE ALGORITHM

- 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 non qualified nodes

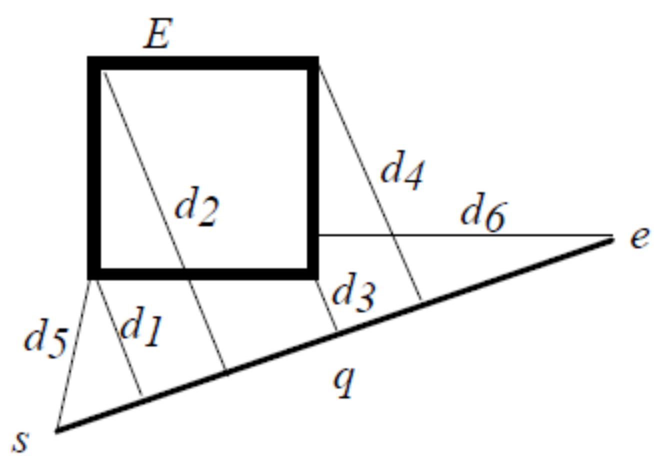
USC Viterbi School of Engineering

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.



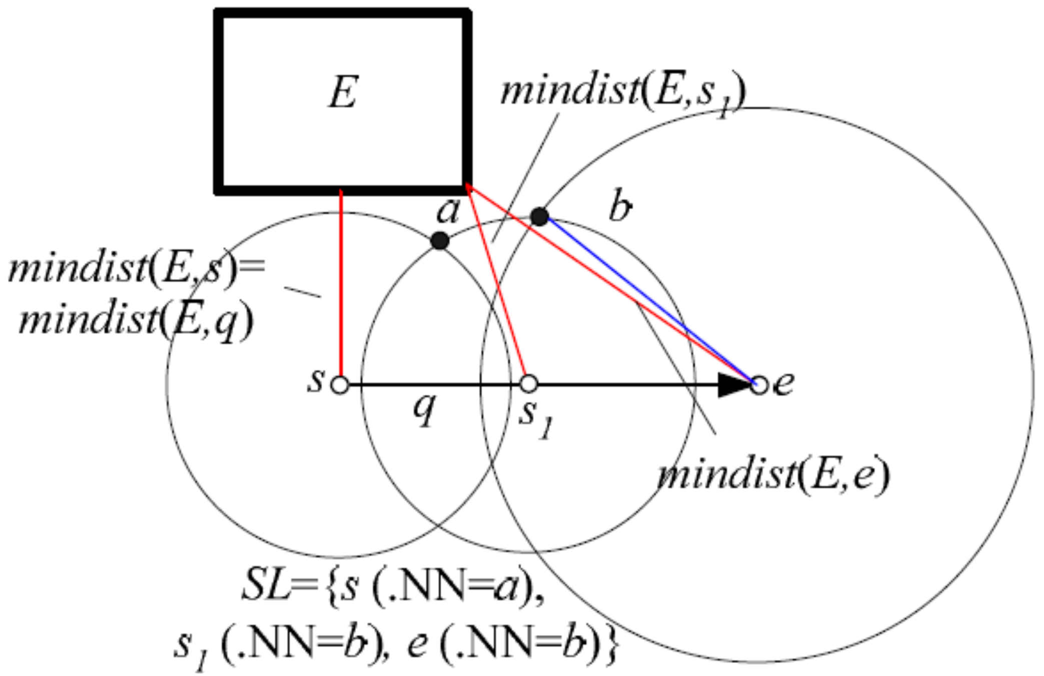
SL={s (.NN=a), s_1 (.NN=b), e (.NN=b)}



Compute Mindist(E,q)

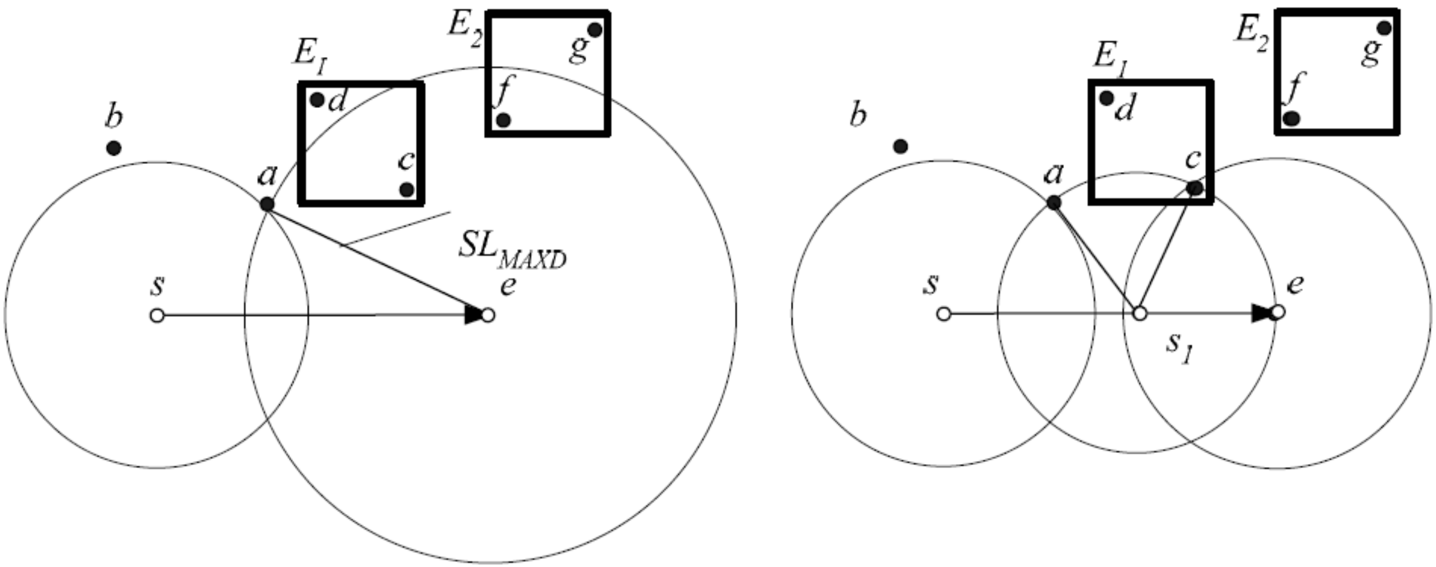
R-TREE ALGORITHM – HEURISTIC 2 (AFTER 1)

- 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).



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.*



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

Before processing E_1

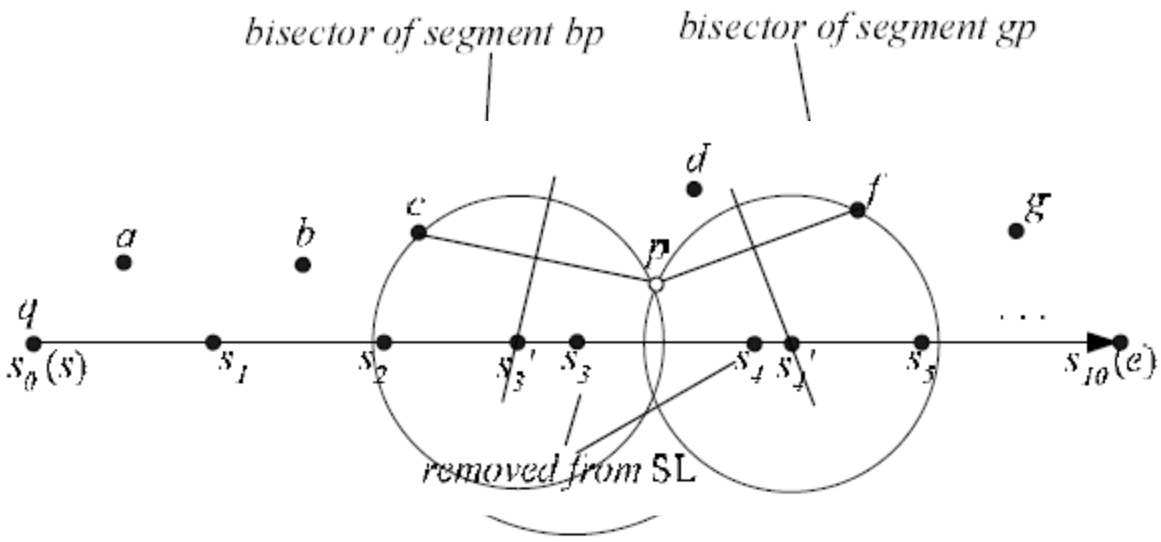
$SL = \{s\ (.NN{=}a),\ s_1\ (.NN{=}c),\ e\ (.NN{=}c)\}$

After processing E_1

USC Viterbi School of Engineering

R-TREE ALGORITHM – LEAF ENTRY

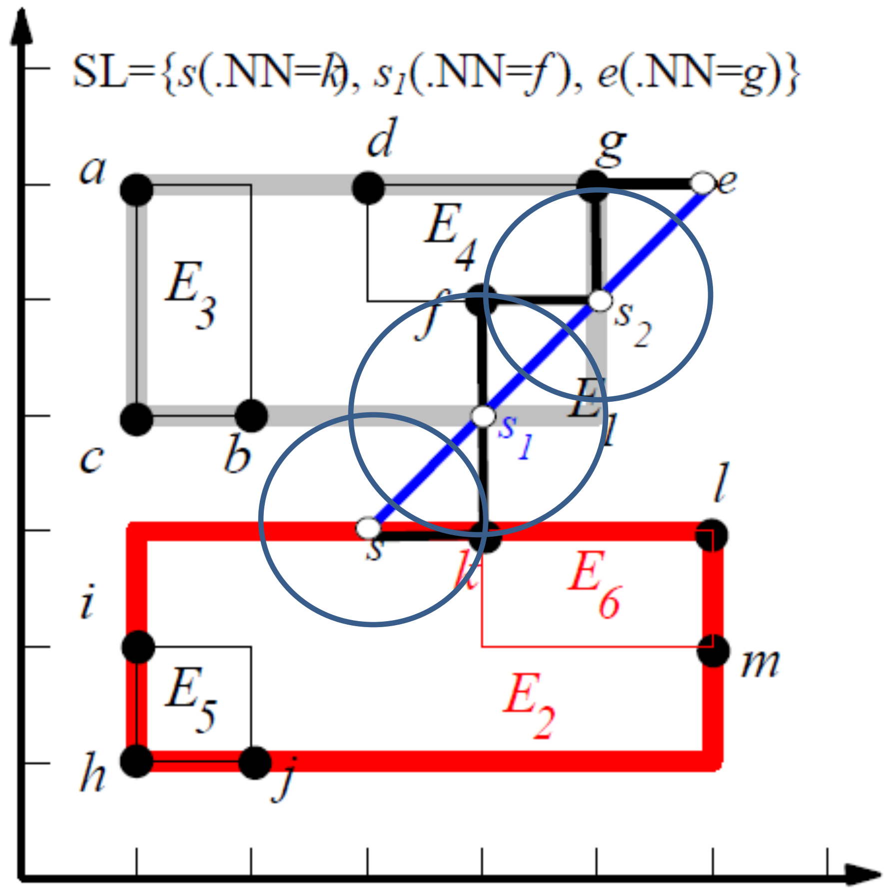
- Input: New entry *p*, SL ={s_1,…s_{10}}
  - 1) retrieve the split points covered by p
  - 2) update SL
- Binary search: 1) [s_0,s_{10}] -> s_5 2) [s_0,s_5] -> s_2
  - Using bisector to judge the direction



USC Viterbi School of Engineering

CKNN – R-TREE ALGORITHM (EXAMPLE)

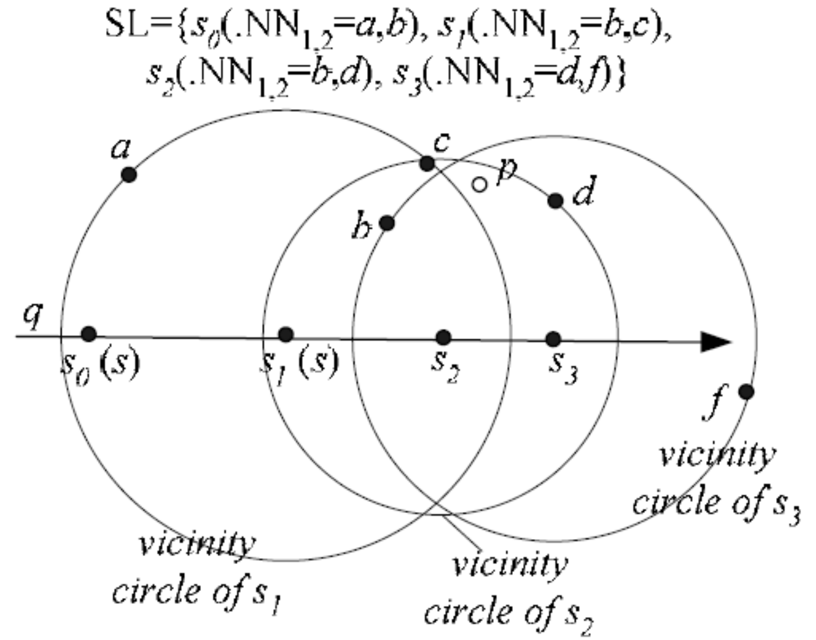
- Depth First (query segment: se)



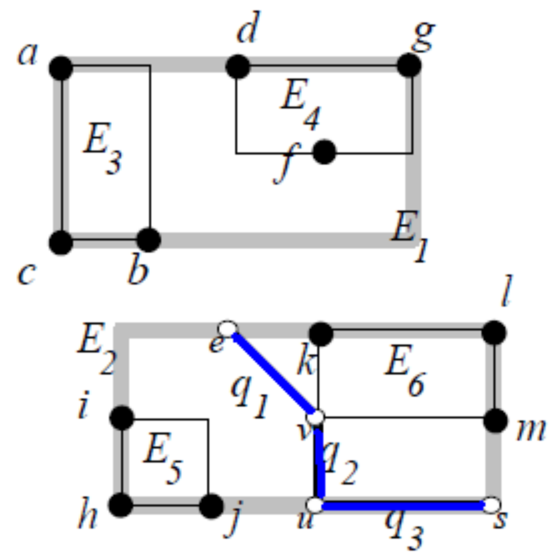
USC Viterbi

OTHER CNN QUERY

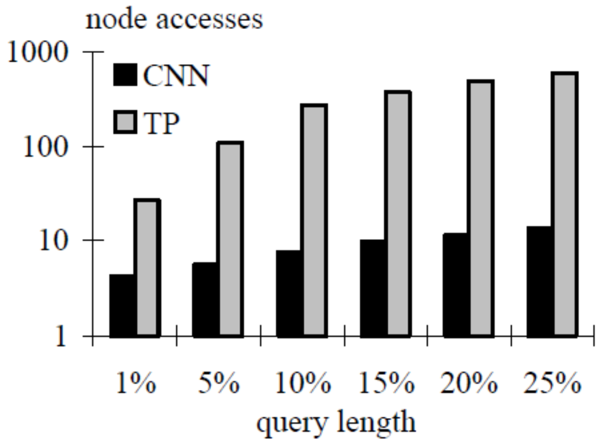
- kCNN query (k=2)

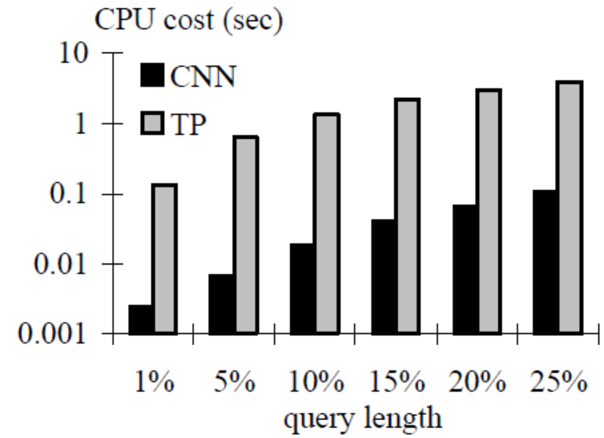


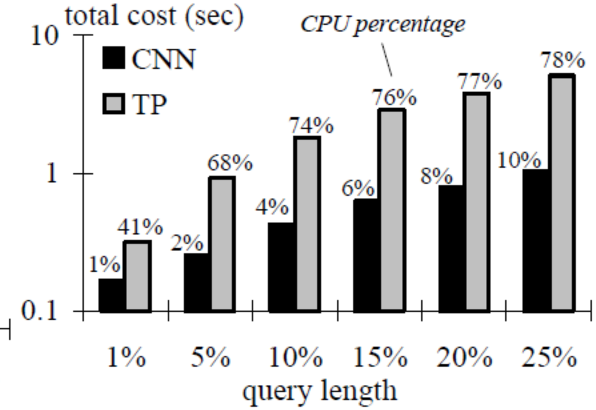
- Trajectory NN query (TNN)
  - q1 = [s,u]
  - q2 = [u,v]
  - q3 = [v,e]
  - Each segment has a SL
  - Treated one by one

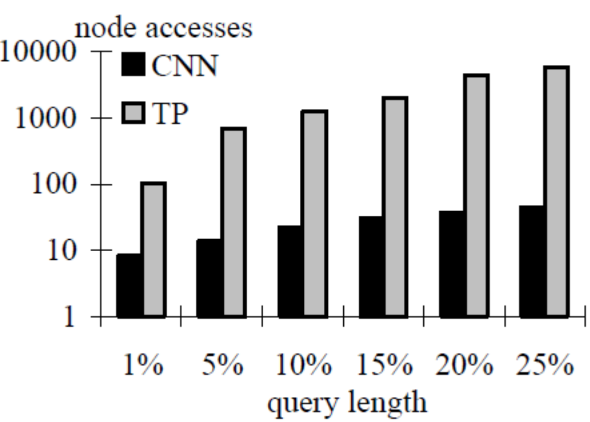


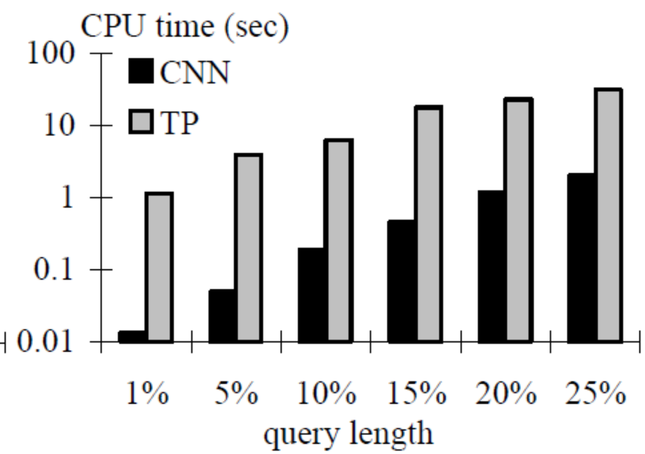
EXP: PERFORMANCE VS QUERY LENGTH

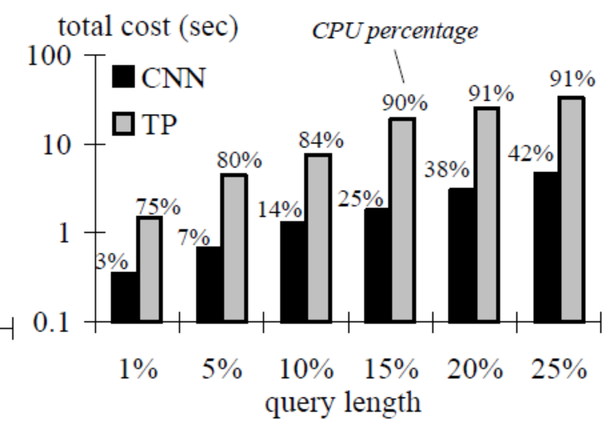












USC Viterbi
School of Engineering

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)

References

- Tao, Y.; Papadias, D. & Shen, Q. Continuous Nearest Neighbor Search. VLDB, 2002, 287-298.
- A presentation by Penny Pan in csci587 Fall’2010

USC Viterbi School of Engineering

Sample question