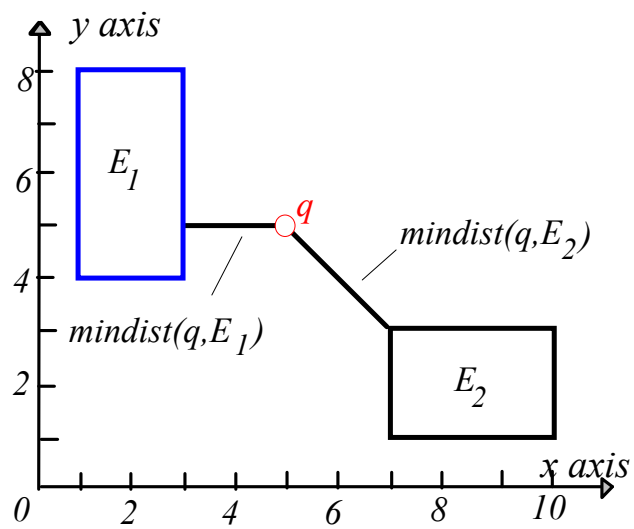
Continuous Nearest Neighbor Search

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

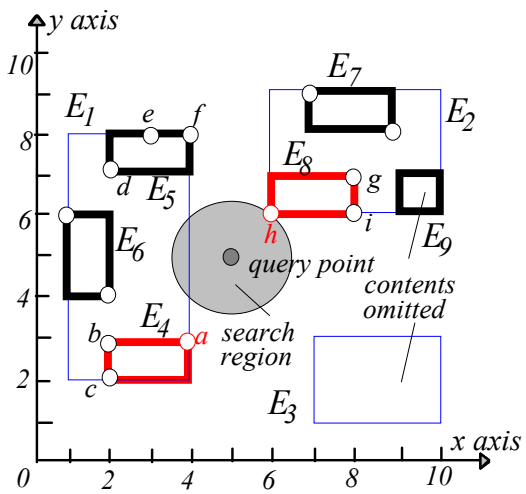
Point Nearest Neighbor (NN) Queries [Roussopoulos et al SIGMOD95, Hjaltason and Samet TODS 99]

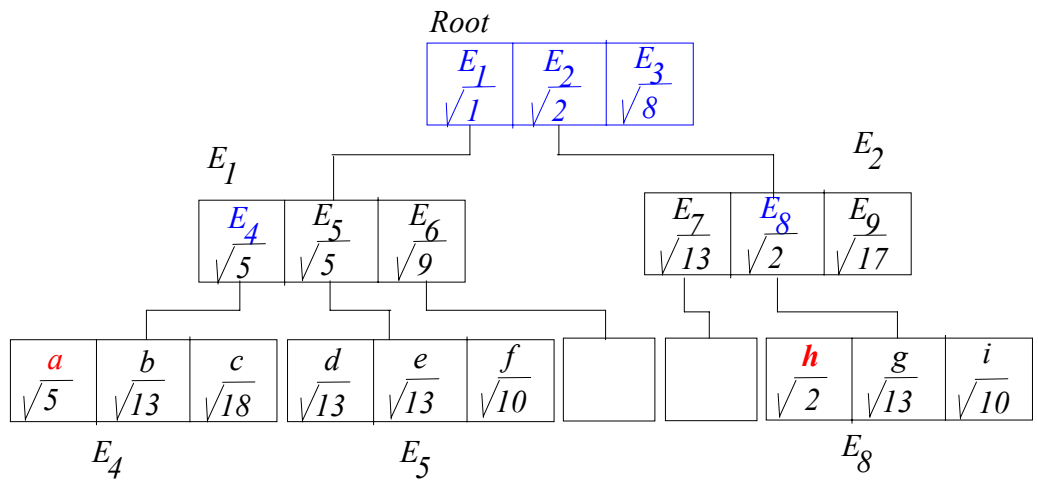
- *Branch and bound algorithms* use *mindist* between the query point q and an R-tree entry E, to prune the search space:
  - *mindist*(E, q) = The minimum distance between E and q



Nearest Neighbor Search (NN) with R-Trees

- *Depth-first* (DF) and *Best-first* (BF) algorihms:

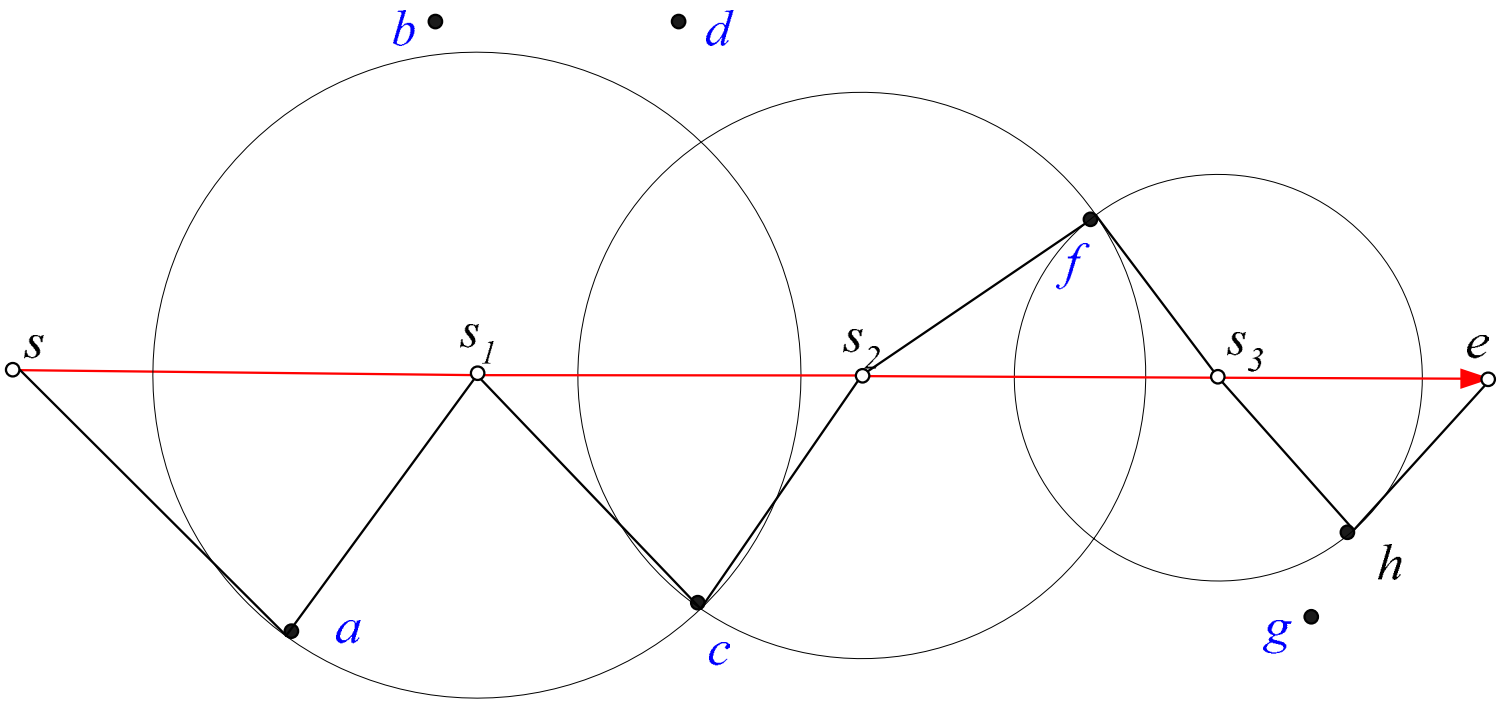




Action	*Heap*							*Result*
Visit Root	$E_1\sqrt{1}$	$E_2\sqrt{2}$	$E_3\sqrt{8}$					*{empty}*
follow E_1	$E_2\sqrt{2}$	$E_4\sqrt{5}$	$E_5\sqrt{5}$	$E_3\sqrt{8}$	$E_6\sqrt{9}$			*{empty}*
follow E_2	$E_8\sqrt{2}$	$E_4\sqrt{5}$	$E_5\sqrt{5}$	$E_3\sqrt{8}$	$E_6\sqrt{9}$	$E_7\sqrt{13}$	$E_9\sqrt{17}$	*{empty}*
follow E_8	$E_4\sqrt{5}$	$E_5\sqrt{5}$	$E_3\sqrt{8}$	$E_6\sqrt{9}$	$E_7\sqrt{13}$	$E_9\sqrt{17}$		$\{(h, \sqrt{2})\}$

Report h and terminate

Problem: Continuous Nearest Neighbor



Data: A set of points

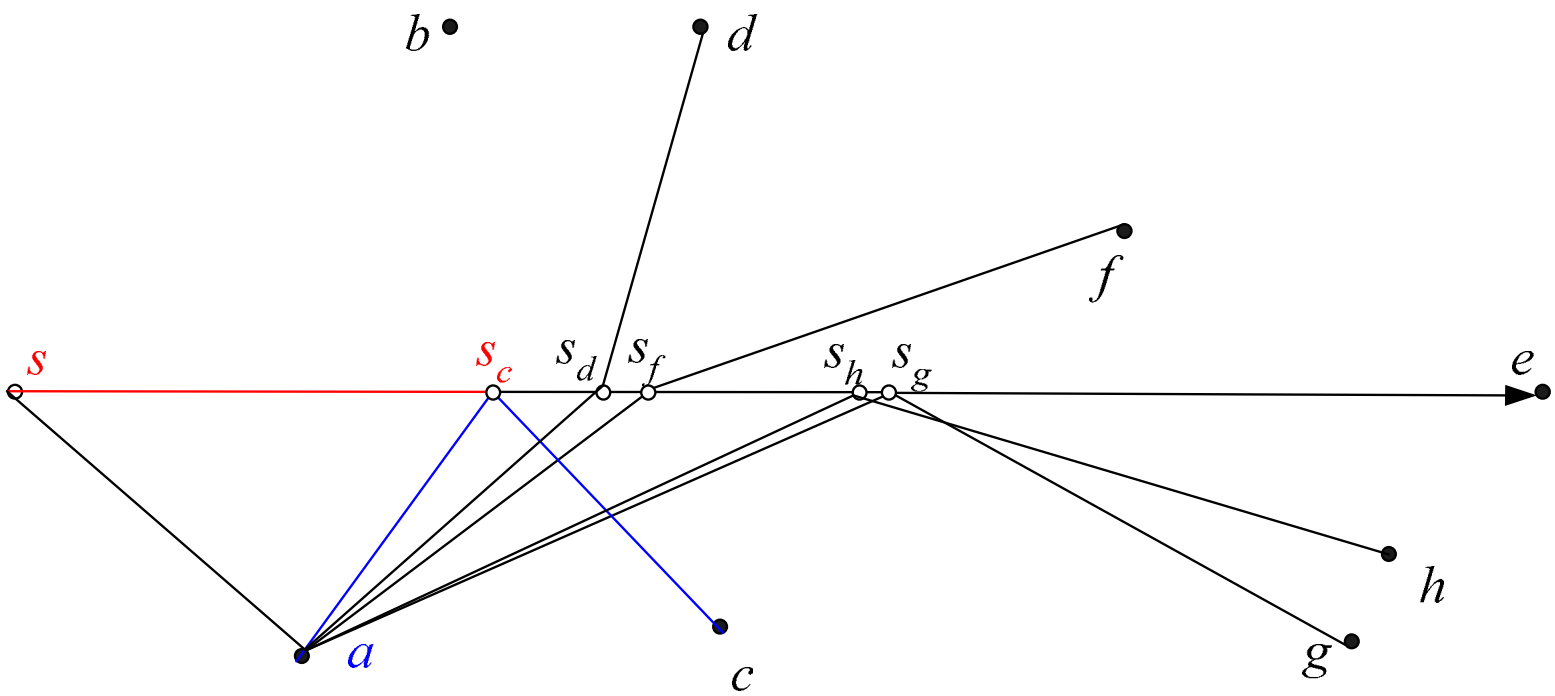
Query: A line segment q=[s, e]

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).

Previous Approach – Time Parameterized Queries (Tao and Papadias, SIGMOD 02)

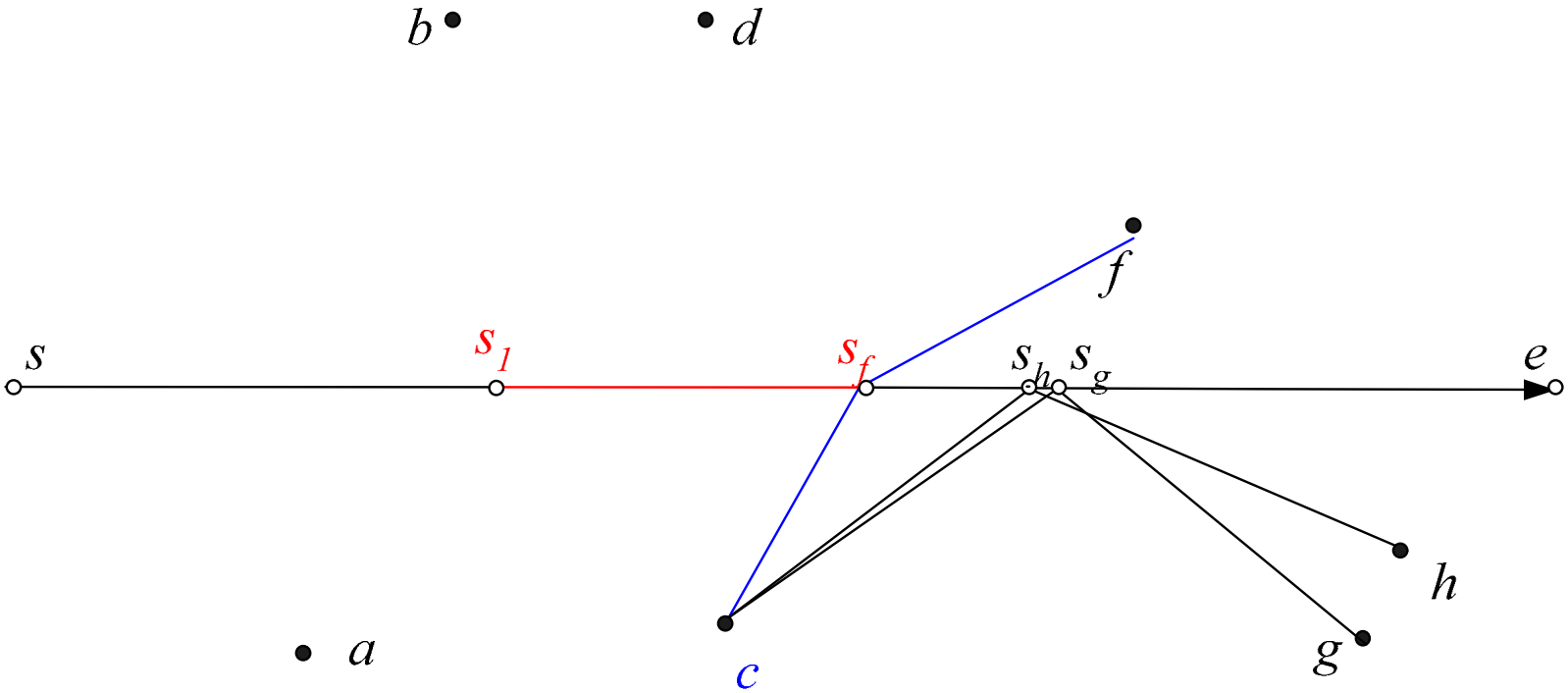


Step 1: Find the NN of the start point *s*, i.e., point *a*.

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.

TP NN (cont)



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

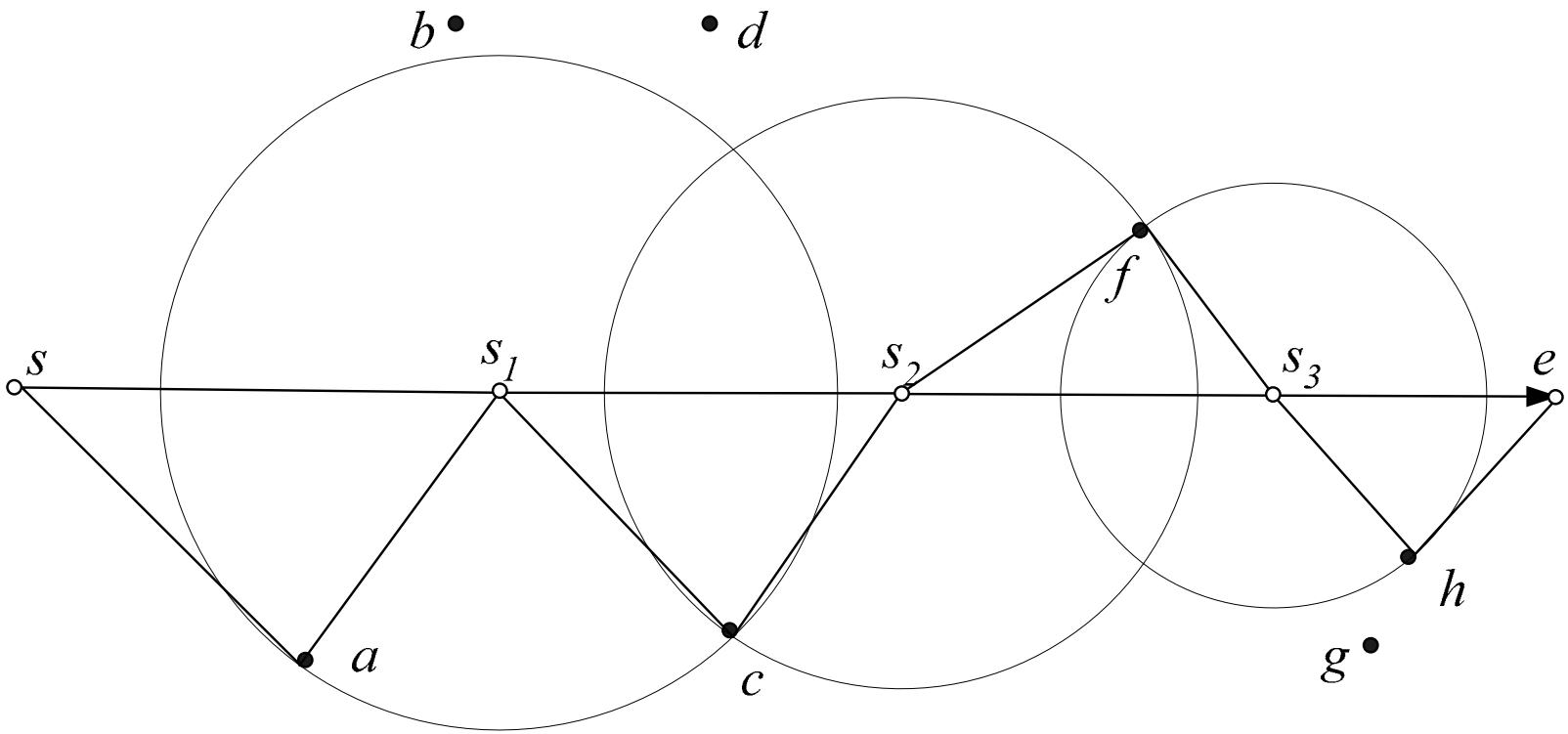
Step 3: Perform another TP NN to find:

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)

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*.

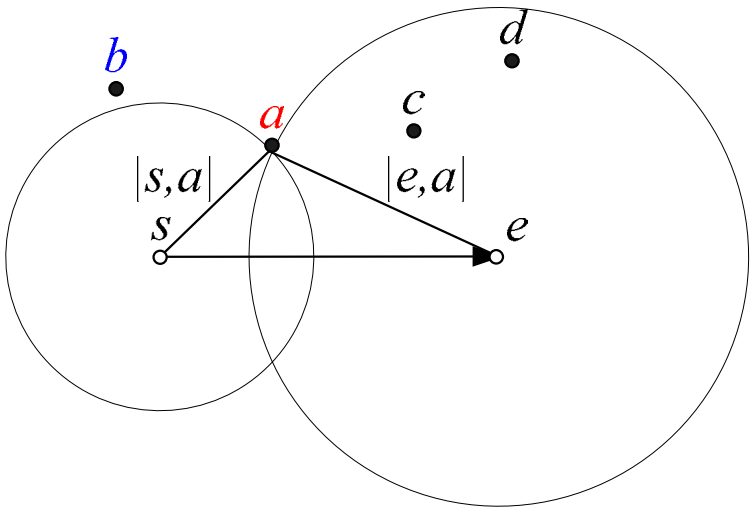
Term1: The set of split points (including s and e) constitute the *split list*.

Term2: The circle that centers at split point s_i with radius $dist(s_i, s_i.\text{NN})$ 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 $[s, s_1]$, $[s_1, s_2]$, $[s_2, s_3]$, $[s_3, e]$.

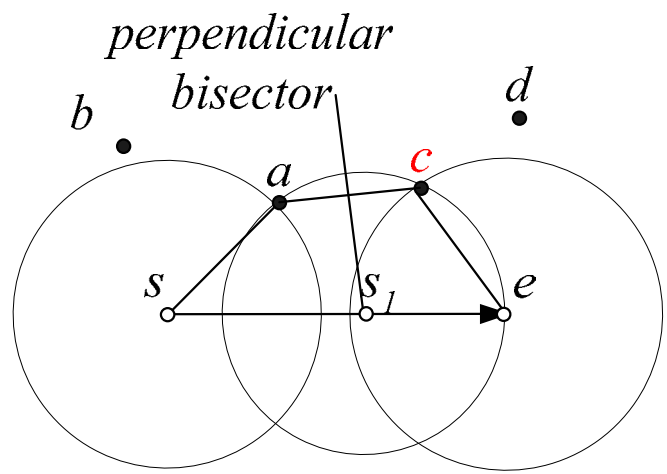
Lemma 1

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

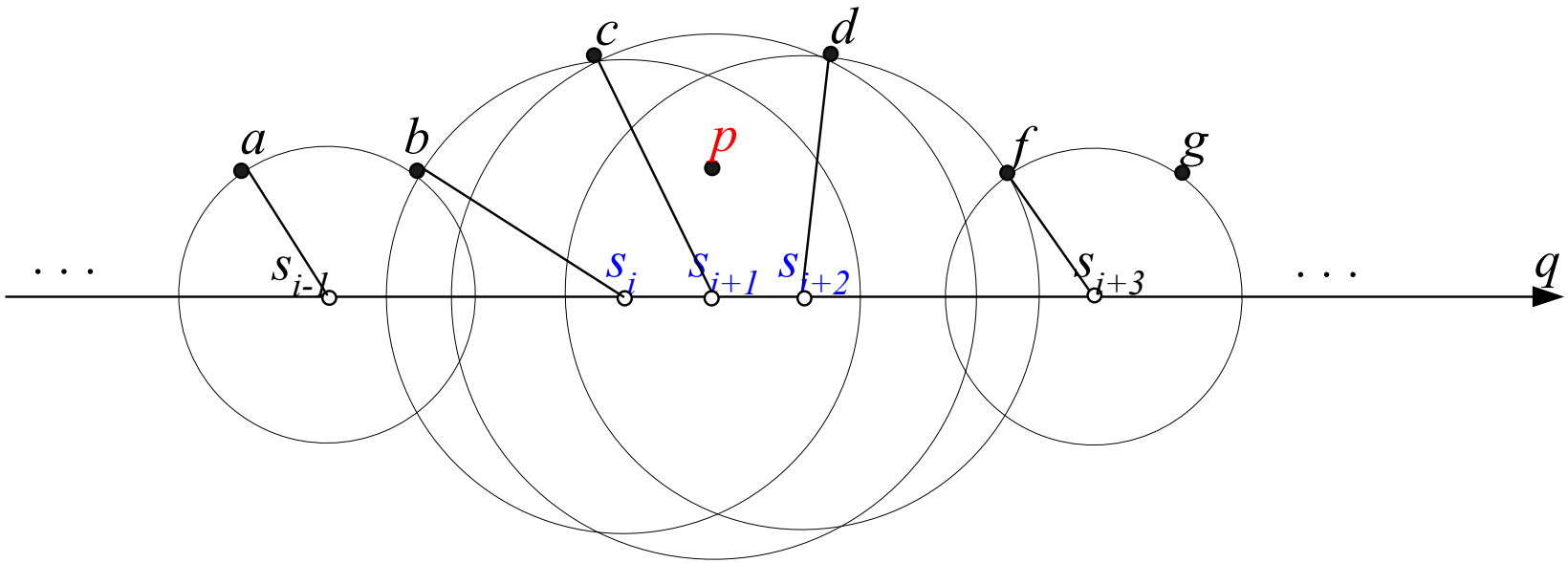


$SL=\{s$ (.NN=a), s_1 (.NN=c), e (.NN=c)$\}$

After processing c

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)$\}$

Algorithm with R-trees Overview

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

<u>When a leaf entry (i.e., a data point) *p* is encountered</u>

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

<u>For an intermediate entry</u>

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

Heuristic 1

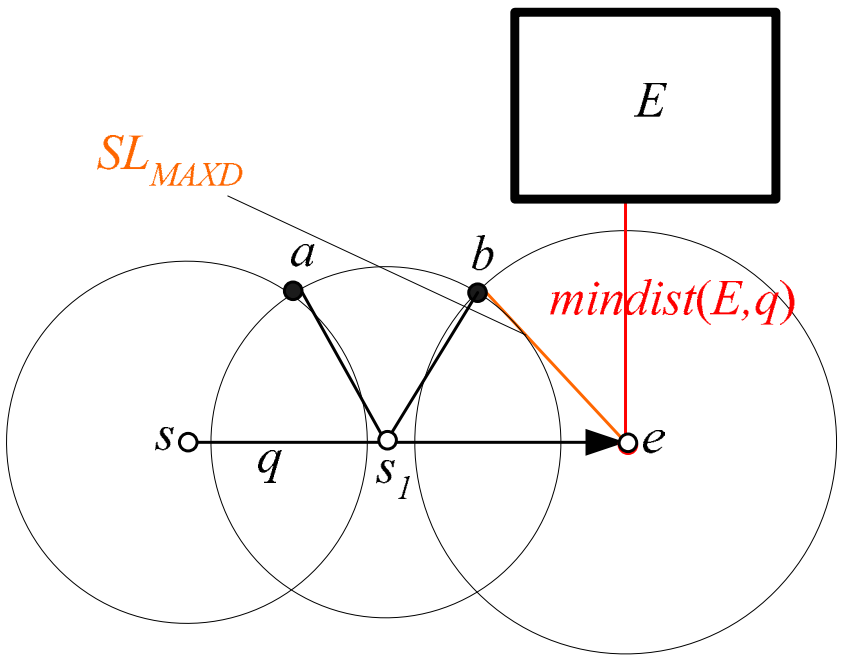
Given an intermediate entry E and query segment q, the subtree of E may contain qualifying points only if

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

where

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

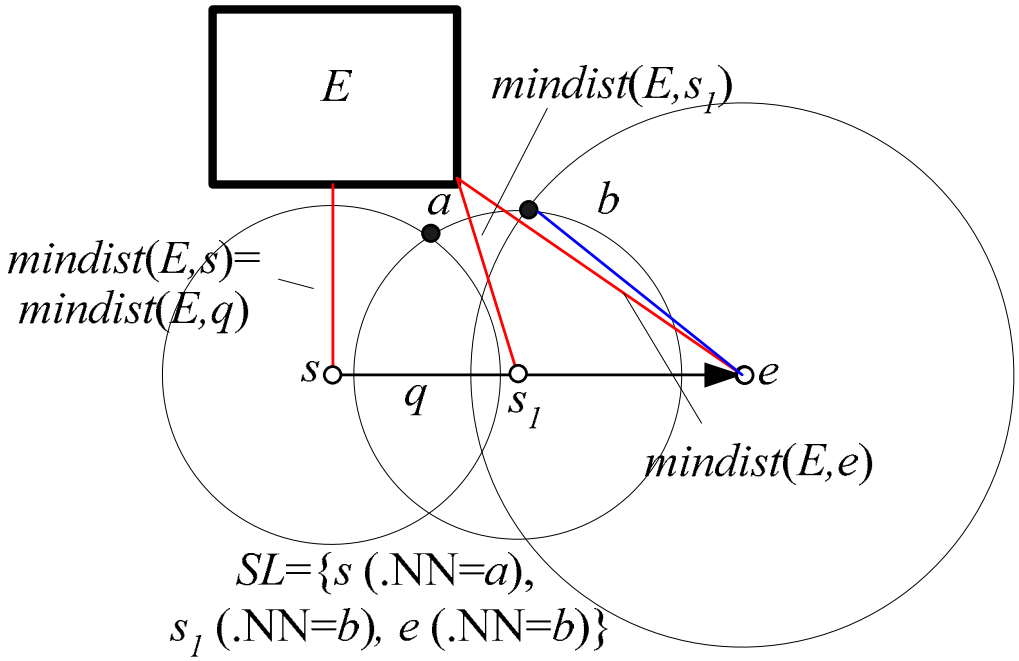
SL_{MAXD} is the maximum distance between a split point and its NN.



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

Heuristic 2

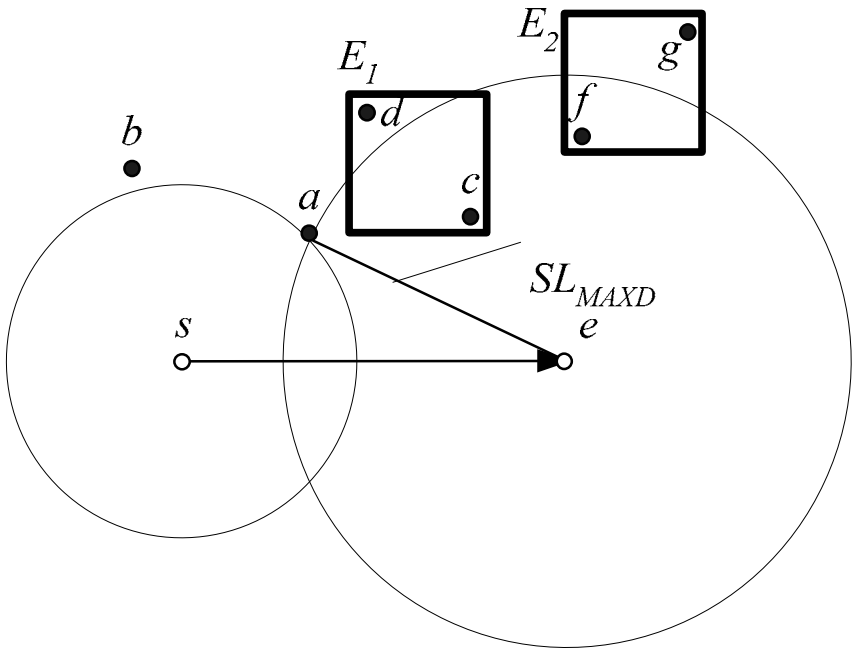
Given an intermediate entry E and query segment q, the subtree of E must be searched *if and only if*
there exists a split point $s_i \in$ SL such that $dist(s_i, s_i.\text{NN}) > mindist(s_i, E)$.



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.

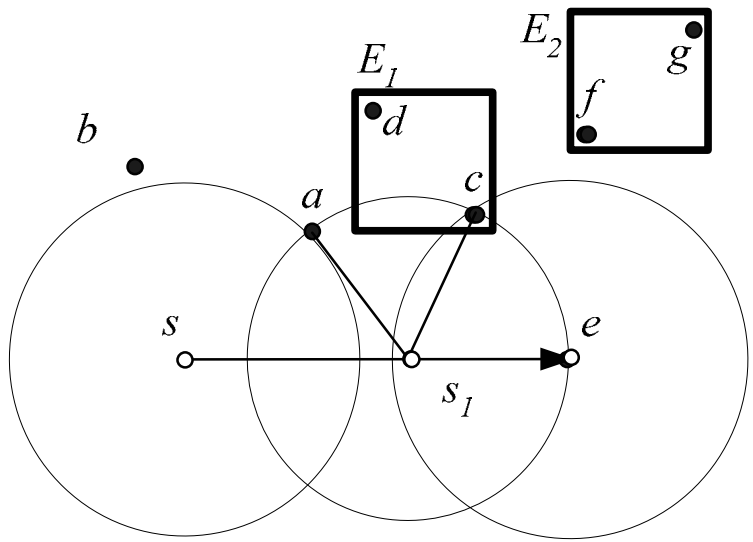
Heuristic 3 (Access Order)

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)}

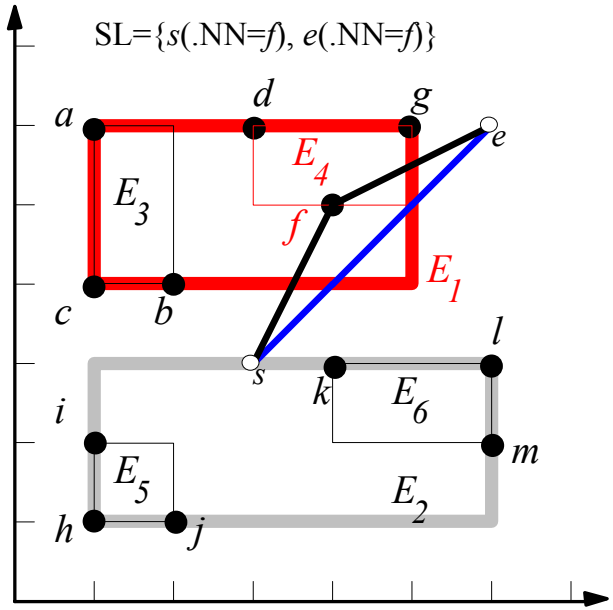
Before processing E_1

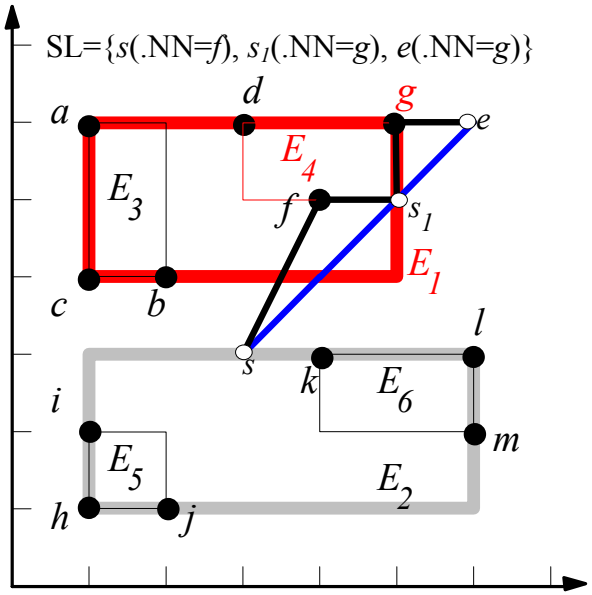


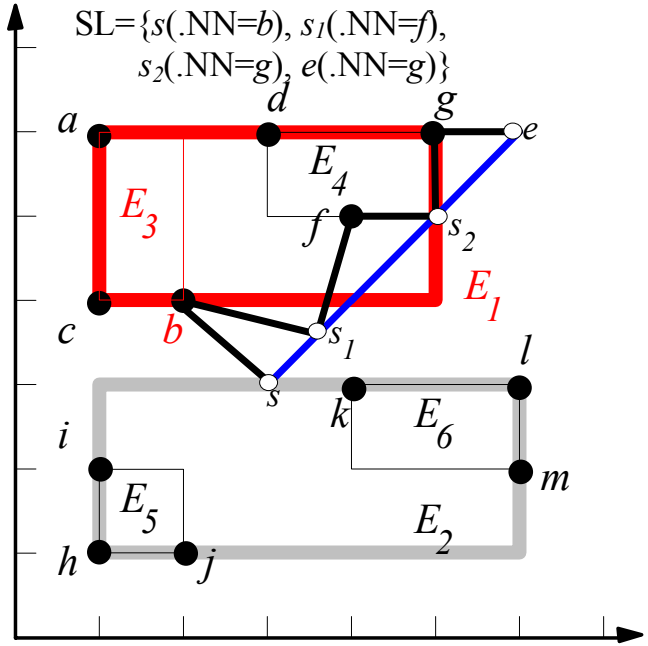
SL={s (.NN=a), s_1 (.NN=c), e (.NN=c)}

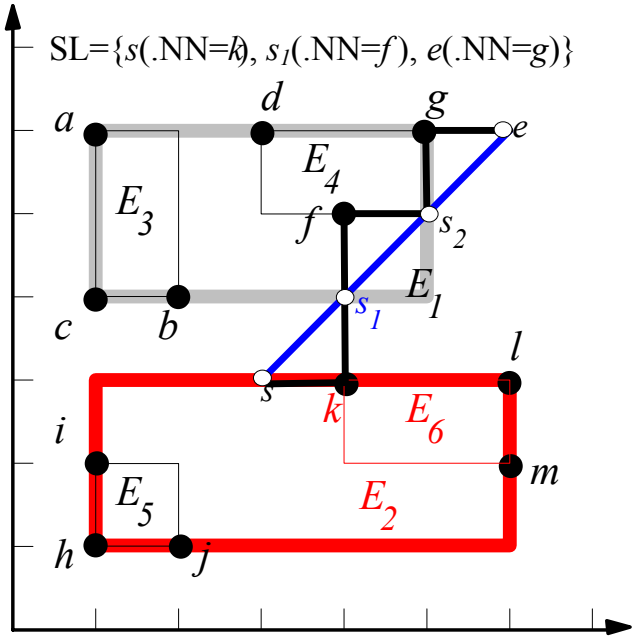
After processing E_1

An Example (Depth First Approach)

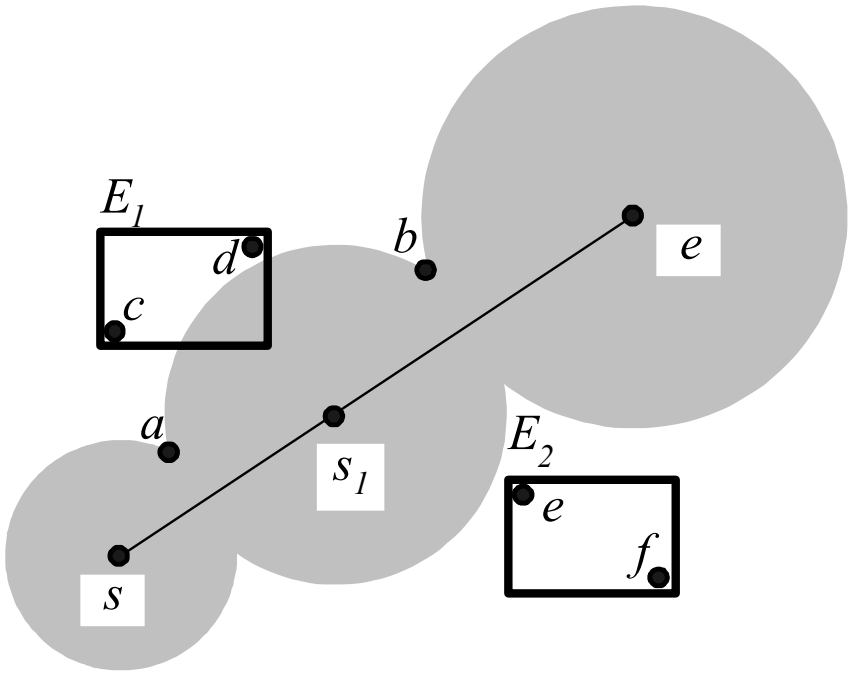




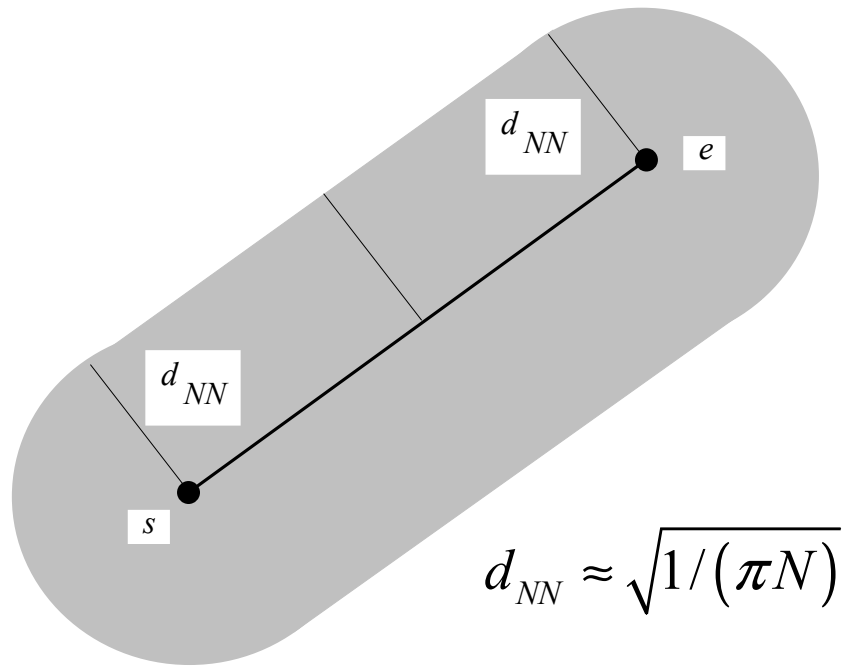




Cost Model for Uniform Data (real data are handled with histograms)



Actual search region

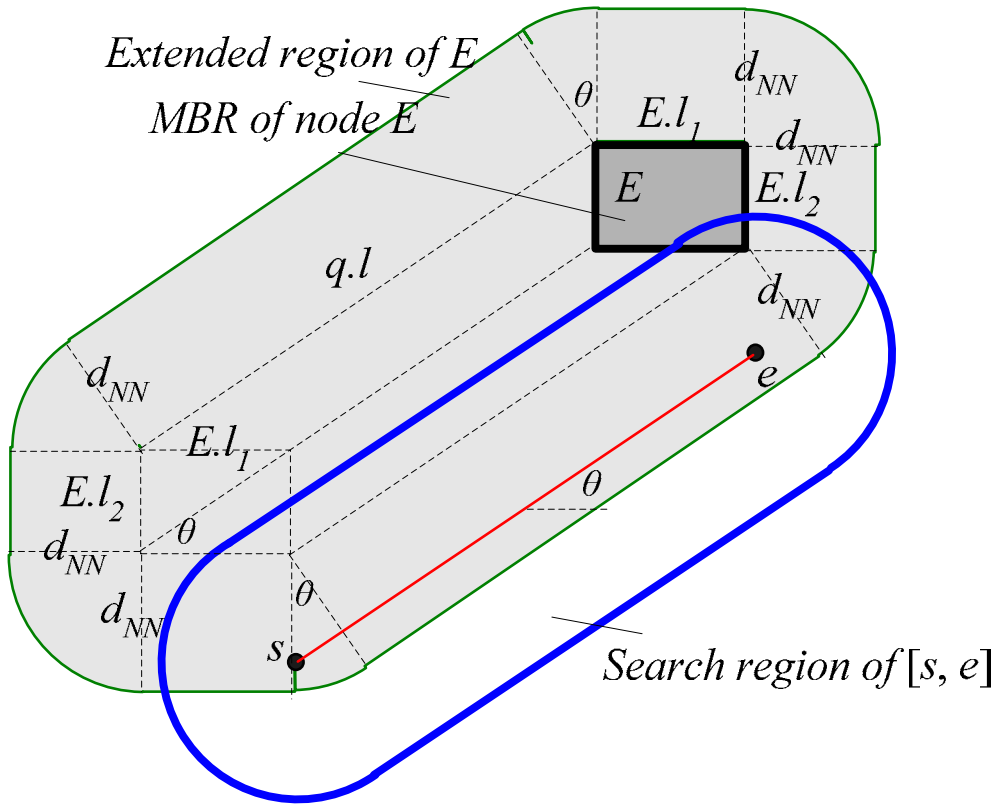


Approximated search region

An optimal algorithm on R-trees must access only those nodes whose MBRs intersect the actual search region (i.e., E_1 but not E_2).

To facilitate the analysis we focus on a more regular (approximated) region

Node Access Probability



$$P_{ACCESS}(E,q) = area(E_{EXT}) =$$

$$\pi d_{NN}{}^2 + E.l_1 \cdot E.l_2 + 2d_{NN}\left(E.l_1 + E.l_2 + q.l\right)$$

$$+2q.l\left(E.l_1 \cdot |\cos\theta| + E.l_2 \cdot |\sin\theta|\right)$$

Cost Model

$$NA(CNN) = \sum_{i=0}^{h-1} N_i \cdot P_{ACCESS}\left(E.l_i, q\right)$$

$$= \sum_{i=0}^{h-1} N_i \cdot \begin{bmatrix} \pi d_{NN}{}^2 + E.l^2 + 2 \cdot d_{NN}\left(2 \cdot E.l + q.l\right) \\ +2 \cdot q.l \cdot E.l\left(|\cos\theta| + |\sin\theta|\right) \end{bmatrix}$$

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

Various models have been proposed for $E.l$ and N_i in the context of R-tree analysis.

Our algorithm is I/O-bounded. Hence the above model (producing number of node accesses) reflects the performance.

The performance of non-uniform data can be easily captured with histograms.

Experimental Settings

Datasets:

Uniform

Real: 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.

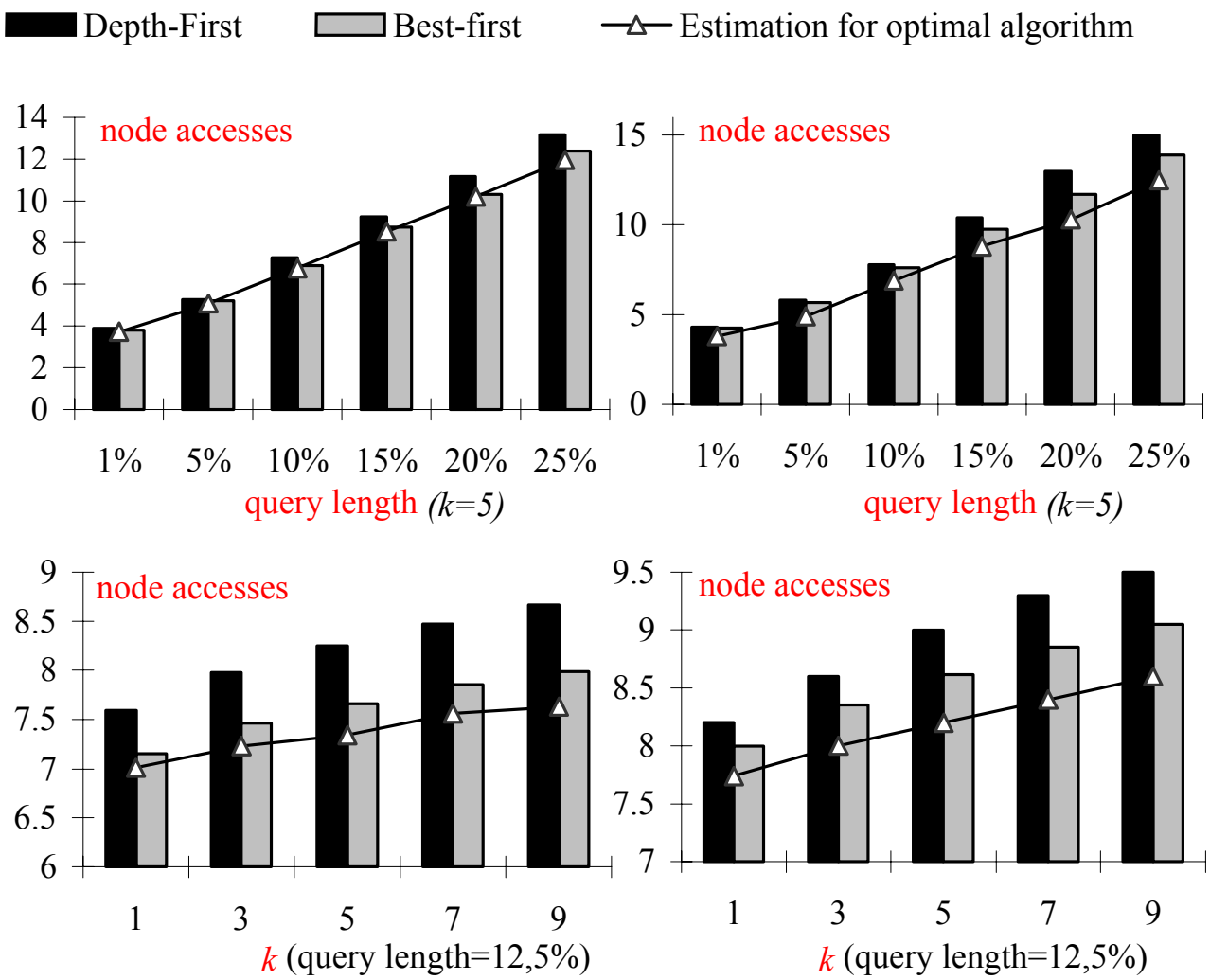
Machine:

1Ghz CPU, 256M memory

Page size=4K (R-tree node capacity=200)

Compare CNN and TP (the only existing solution)

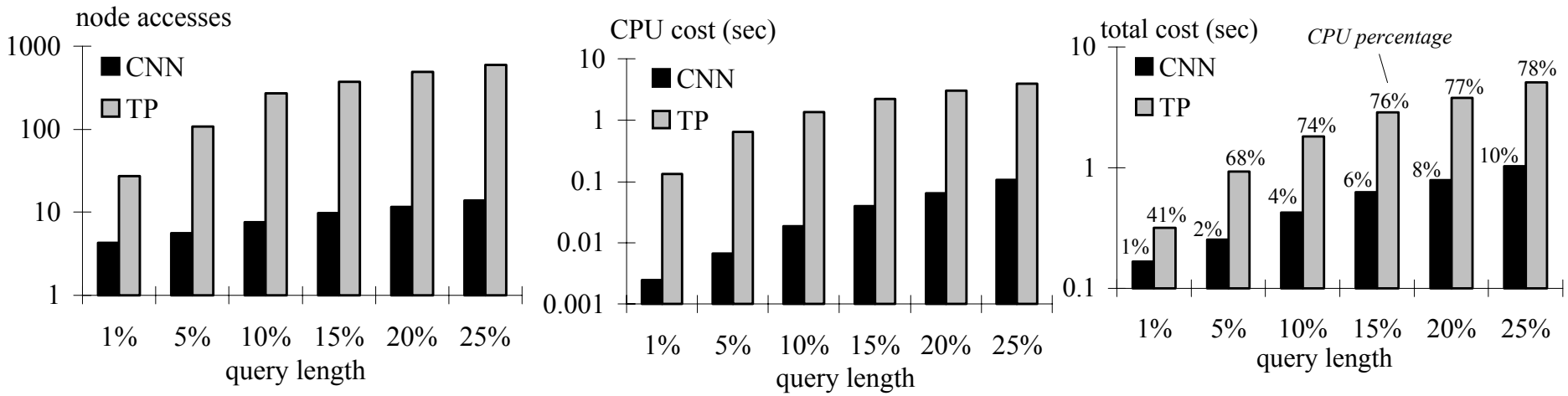
Exp 1: Cost Model Evaluation



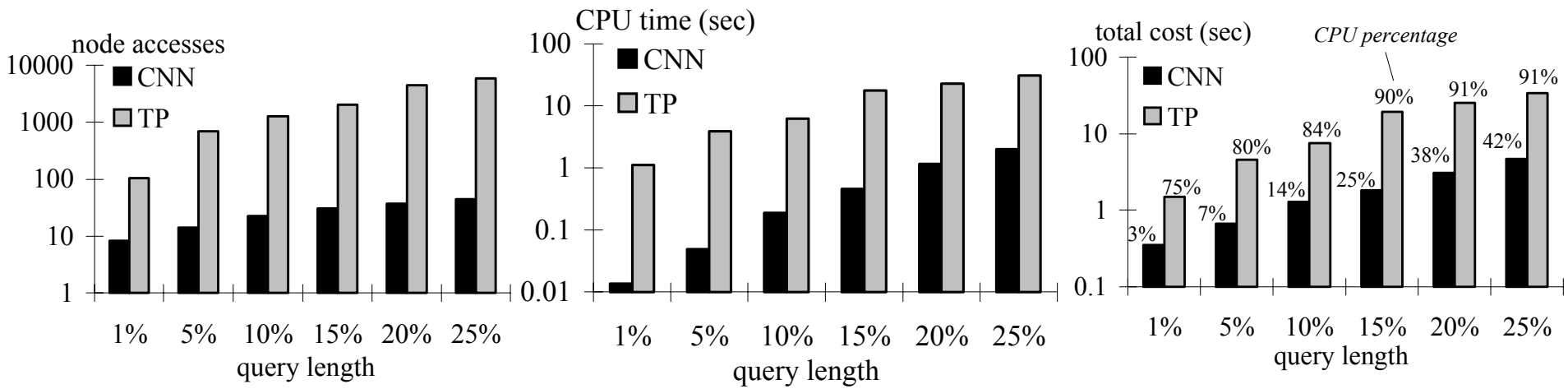
Uniform

CA

Exp 2: Performance vs Query Length

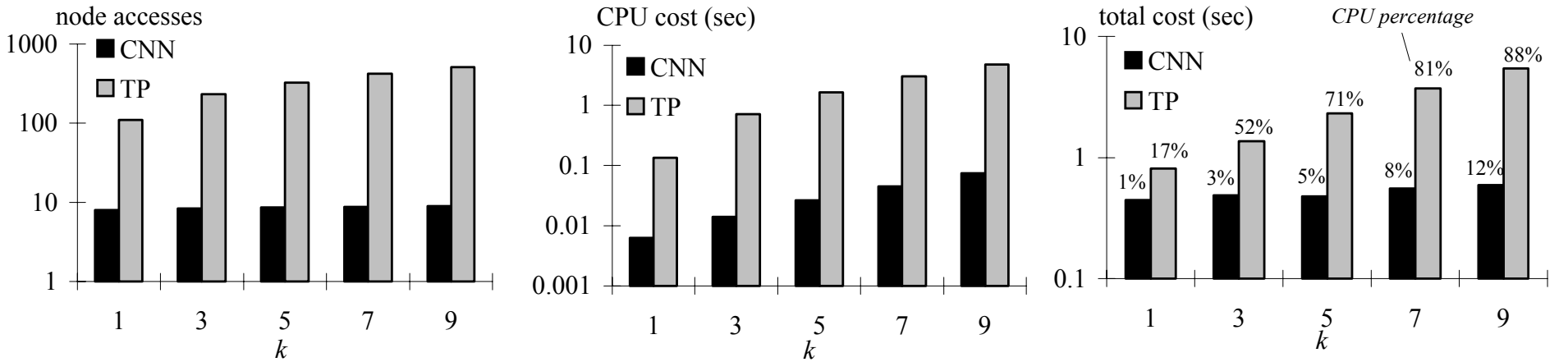


CA

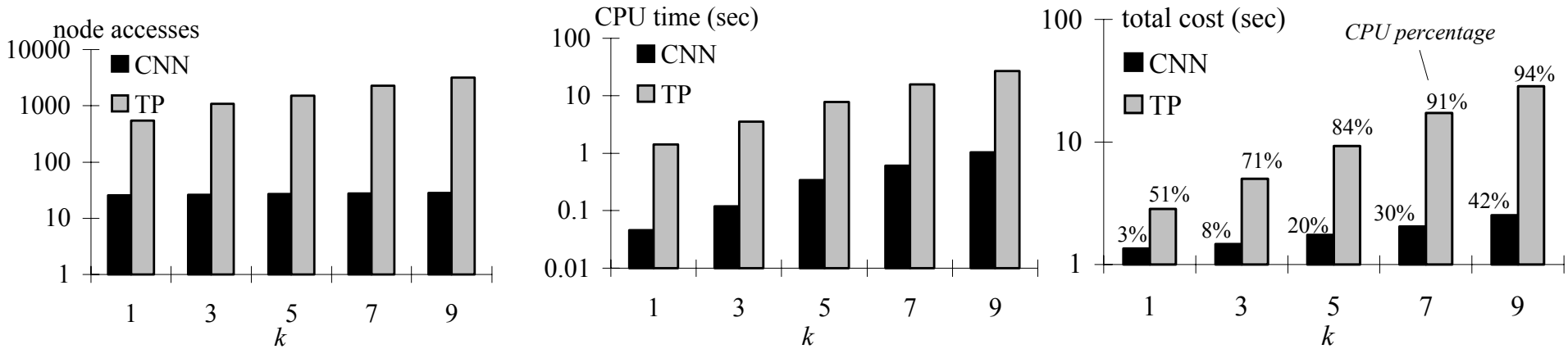


ST

Exp 3: Performance vs *k* (number of neighbors to be retrieved for each point)



CA



ST

Conclusion

A fast algorithm for C-*k*NN query.

Future work:

Rectangle data

Moving data points

Application to road networks (i.e., travel instead of Euclidean distance)