

– MRkNNCoP-Baum

[Achtert, Böhm, Kröger, Kunath, Pryakhin, Renz. ACM Int. Conf. Management of Data (SIGMOD), 2006]

- Vergleich bisheriger Verfahren

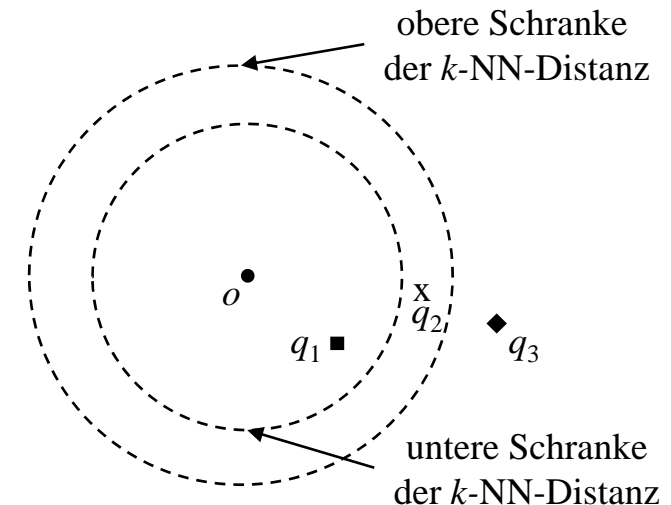
- RNN-Tree/RdNN-Tree: Vorberechnung der NN-Distanz
 - » Wert für k ist fix und vorher bekannt
 - » Update-Problematik
 - » Dafür: erweiterbar auf metrische Daten (z.B. M-Tree)
- Geometrische Suche
 - » Nur für Vektordaten
 - » Teurer Verfeinerungsschritt, schlechtere Selektivität
 - » Dafür: beliebiges k , keine Update-Problematik

- Idee:

- Benutze die gute Selektivität der vorberechneten NN-Distanzen
- Berechne für mehrere (am besten alle) Werte für k die k -NN-Distanzen vor
- Problem: Speicherung aller Distanzen zu aufwendig
 - » Pro Objekt alle k -NN-Distanzen => Index wäre sehr hoch => hohe Kosten
- Lösung: Approximiere die k -NN-Distanzen
 - » Approximation sollte obere Schranke (UB) der k -NN-Distanz sein => true drops, wir können Objekte (Seiten) frühzeitig ausschließen
 - » Zusätzliche Approximation als untere Schranke (LB) => true hits

- Bewertung von Objekt o (analog: Seiten) mit UB- und LB-Approximationen

- $\text{dist}(o, q) \leq \text{LB}_{k\text{-NN-Dist}}(o)$
 $\Rightarrow o$ true hit, d.h. $o \in \text{RNN}(q, k)$
 » Beispiel: $q = q_1$
- $\text{dist}(o, q) \geq \text{UB}_{k\text{-NN-Dist}}(o)$
 $\Rightarrow o$ true drop, d.h. $o \notin \text{RNN}(q, k)$
 » Beispiel: $q = q_3$
- $\text{UB}_{k\text{-NN-Dist}}(o) \leq \text{dist}(o, q) \leq \text{LB}_{k\text{-NN-Dist}}(o)$
 $\Rightarrow o$ Kandidat
 » Beispiel: $q = q_2$



- Gegeben: für jedes Objekt o eine Sequenz der k -NN-Distanzen, $\langle 1\text{-NN-Dist}(o), 2\text{-NN-Dist}(o), \dots, k_{\max}\text{-NN-Dist}(o) \rangle$ für ein hinreichend großes k_{\max}
- Frage: wie kann ich UB- und LB-Approximation dieser k -NN-Distanzen berechnen und kompakt speichern?

- Lösung aus der Theorie der Selbstähnlichkeit

- Potenzgesetz gilt für Verhältnis zwischen
 - » dem Radius einer Hyperkugel
 - » der Anzahl an Objekten innerhalb der Hyperkugel

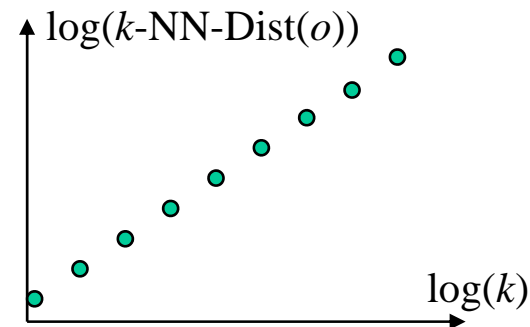
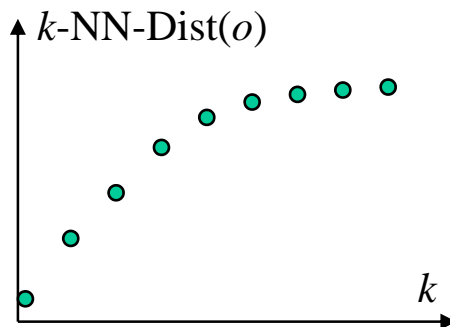
$$encl(\varepsilon) \propto \varepsilon^{d_f}$$

wobei $encl(\varepsilon) = \#$ Objekte innerhalb der Kugel
 $d_f =$ „Fraktale Dimension“

- Übertragung auf k -NN-Sphäre:

- » $\varepsilon = k$ -NN-Distanz
- » $encl(\varepsilon) = k$

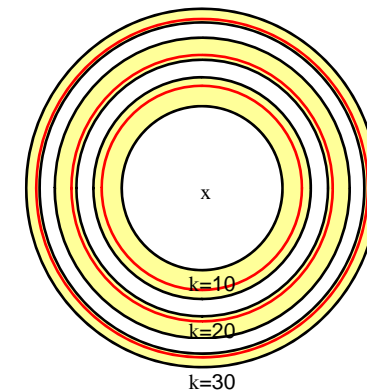
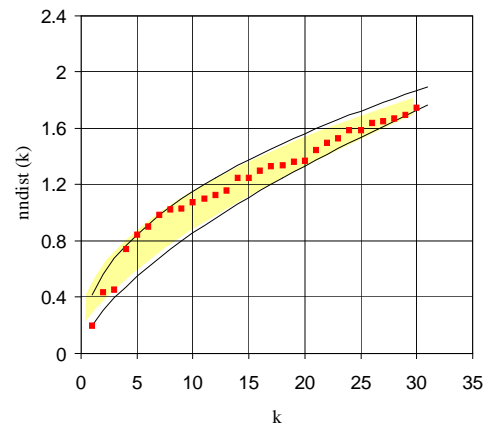
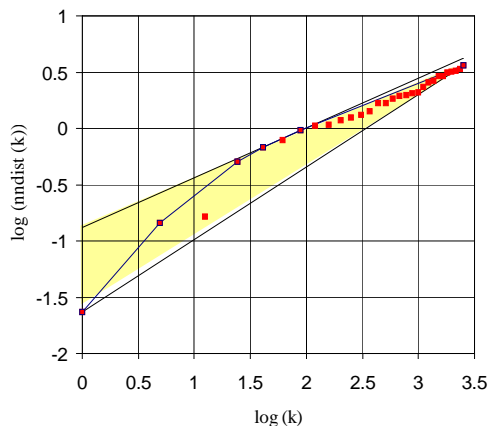
- Im log-log-Raum: $\log(k - \text{NN} - \text{Dist}(o)) \propto \frac{\log(k)}{d_f}$



- In der Realität verhalten sich die Distanzen nicht wie perfekte Linien im log-log-Raum
- Trotzdem: im log-log-Raum können die k -NN-Distanzen mit einer Linie approximiert werden
- Das ist erheblich billiger als alle k -NN-Distanzen zu speichern, oder andere Funktionen höherer Ordnung zu verwenden, um die Distanzen im normalen k / k -NN-Dist – Raum zu approximieren
- LB- und UB-Approximationen
 - UB-Approximation ist eine Linie im log-log-Space, sodass

$$\forall k \leq k_{\max} : k - \text{NN} - \text{Dist}(o) \leq \text{UB}_{k-\text{NN-Dist}}(o)$$
 - LB-Approximation ist eine Linie im log-log-Space, sodass

$$\forall k \leq k_{\max} : k - \text{NN} - \text{Dist}(o) \geq \text{LB}_{k-\text{NN-Dist}}(o)$$



- Jedem Objekt wird zugeordnet
 - Eine LB-Approximation der k -NN-Distanzen
 - Eine UB-Approximation der k -NN-Distanzen
- Jeder Seite im Index wird zugeordnet
 - Eine UB-Approximation der LB-Approximationen der Kindseiten
 - LB-Approximation wird nicht gespeichert, da zu wenig selektiv
- Vorteil:
 - Beliebige k
 - Für allgemein metrische Daten (M-Tree) oder Vektordaten (z.B. X-Tree)
 - Durch UB- und LB-Approximationen höhere Filterselektivität als Geometrisches Verfahren => weniger Kandidaten die verfeinert werden müssen
- Nachteil
 - Updateproblematik
 - k_{\max} muss bekannt sein (ABER i.d.R. kein Problem)
 - Teure Verfeinerung nötig (ABER i.d.R. deutlich weniger Kandidaten)

- **Filter-Algorithmus** für allgemein metrische Daten (M-tree)

Knoten Node = (RoutingObj, CovRadius)

$MINDIST(q, Node) = \max\{\text{dist}(q, Node.RoutingObj) - CovRadius, 0\}$

MRkNNCoP-Tree-Search(DB, q) // DB als MRkNNCoP-Tree organisiert

result = \emptyset ;

candidates = \emptyset ;

queue = **LIST OF** (dist:Real, obj:Object) **ORDERED BY** dist **ASCENDING**;

queue = [(0.0, DB.root)];

WHILE NOT queue.isEmpty() **DO**

 p = queue.first().Object;

IF p.isDataPage() **THEN**

FOR $i=0$ **TO** p.size() **DO**

IF $\text{dist}(q, p.\text{getObject}(i)) \leq LB_{k\text{-NN-Dist}}(p.\text{getObject}(i))$ **THEN**

 result := result \cup getObject(i);

ELSE IF $\text{dist}(q, p.\text{getObject}(i)) \leq UB_{k\text{-NN-Dist}}(p.\text{getObject}(i))$ **THEN**

 candidates := candidates \cup getObject(i);

ELSE // p ist Directoryseite

FOR $i=0$ **TO** p.size() **DO**

IF $MINDIST(q, p.\text{getRegion}(i)) \leq UB_{k\text{-NN-Dist}}(p.\text{getRegion}(i))$ **THEN**

 queue.insert(($MINDIST(q, p.\text{getRegion}(i))$, p.childPage(i)));

2.5.3 Mutual Pruning Strategien

Idee:

- Ausschluss von Seiten/Objekten mittels Distanzabschätzungen (ohne Vorberechnung, zur Laufzeit der Anfrage ermittelt)
- Für mindestens k Objekte ist die Distanz zu Objekt o mindestens so groß wie die Distanz zwischen Anfrageobjekt q und o
=> Objekt $o \notin RkNN(q)$
- Distanzabschätzungen über Indexseitenregionen

Vorteile:

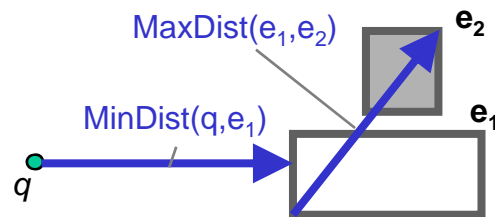
- Keine Vorberechnung notwendig
→ keine Update-Problematik mehr
- Parameter k zur Anfragezeit frei wählbar

– Min/Max-Distanz-Ansatz

[Achtert, Kröger, Kriegel, Renz, Züfle, EDBT, 2009]

- Voraussetzung: Indexseite speichert Anzahl der Objekte $|e|$ in Seite e

Ausschluß von Seite e
durch Seite e' ($k \geq 1$):



$\text{MaxDist}(e_1, e_2) < \text{MinDist}(q, e_1)$
 $\Rightarrow e_1$ kann keine $\text{RkNN}(q)$ -Ergebnisse
enthalten wenn $|e_2| \geq k$

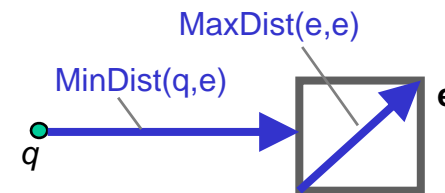
$\text{MinDist}(e_1, e_2)$: Distanz zwischen den beiden voneinander am nächsten liegenden Punkten $p_1 \in e_1$ und $p_2 \in e_2$

$\text{MaxDist}(e_1, e_2)$: Distanz zwischen den beiden voneinander entferntesten Punkten $p_1 \in e_1$ und $p_2 \in e_2$

- Vorteile:

- » Keine Vorberechnung mehr (keine Update-Problematik)
- » Parameter k zur Anfragezeit frei wählbar
- » Pruningkonzept auf allg. metrische Daten anwendbar

Seite e schließt sich selbst aus ($k \geq 1$):

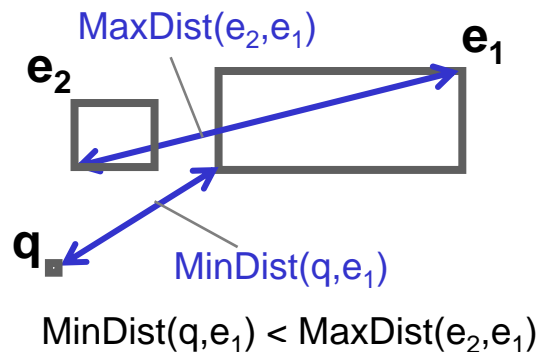


$\text{MaxDist}(e, e) < \text{MinDist}(q, e)$
 $\Rightarrow e$ kann keine $\text{RkNN}(q)$ -Ergebnisse
enthalten wenn $|e|-1 \geq k$

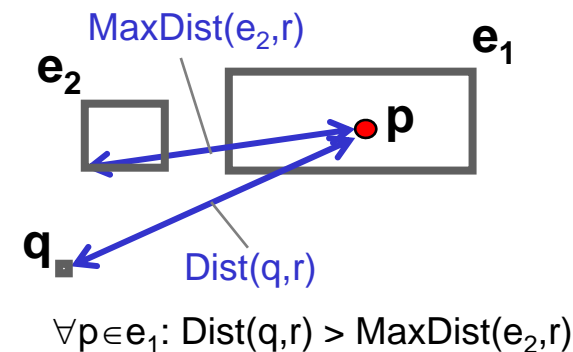
– Problem:

- » Abhängigkeiten zwischen Distanzen unberücksichtigt

Min/Max-Dist Pruning:



„Optimales“ Pruning:



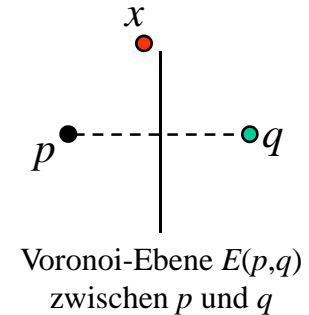
- » Die Distanz $\text{MinDist}(q, e_1)$ schätzt die Distanz $\text{dist}(q, p)$ zwischen q und einem Objekt p in Seitenregion e_1 (untere Distanzabschätzung)
 - » Die Distanz $\text{MaxDist}(e_2, e_1)$ schätzt die Distanz $\text{dist}(o, p)$ zwischen einem Objekt o in Seitenregion e_2 und einem Objekt p in Seitenregion e_1 (obere Distanzabschätzung)
 - » Beide Distanzen $\text{dist}(q, p)$ und $\text{dist}(o, p)$ hängen von der Lage des Objektes p in Region e_1 ab $\Rightarrow \text{dist}(q, p)$ abhängig von $\text{dist}(o, p)$
 - » Diese (Abhängigkeits-) Information geht bei der Verwendung von $\text{MaxDist}(e_2, e_1)$ und $\text{MinDist}(q, e_1)$ verloren
- **geringeres Pruningpotential**

– Geometrische RNN-Suche (Filter/Verfeinerung)

[Tao, Papadias, Lian. Int. Conf. Very Large Databases (VLDB), 2004]

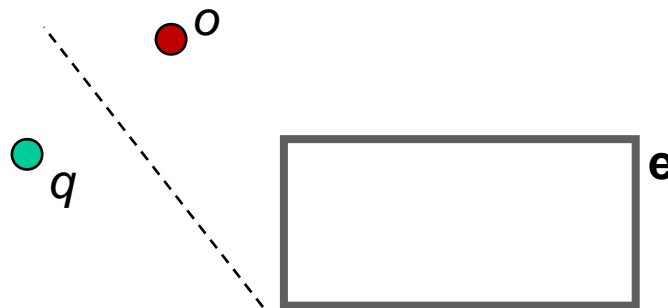
– Idee

- » Gegeben: Voronoi-Ebene zwischen q und beliebigen Punkt p
- » Liegt ein Punkt x auf der Seite von p dieser Voronoi-Ebene, kann q nicht NN von x sein und damit $x \notin RNN(q)$
- » Voronoi-Ebene $E(p,q)$: für alle Punkte $e \in E$ gilt: $dist(q,e) = dist(p,e)$



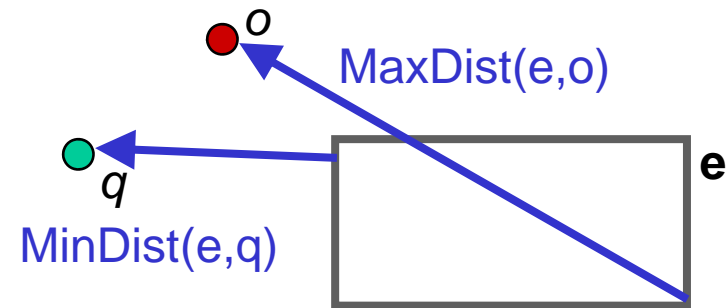
– „Geometrisches“ Pruning ist optimal bzgl. der Pruning-Stärke

geometrisches Pruning:



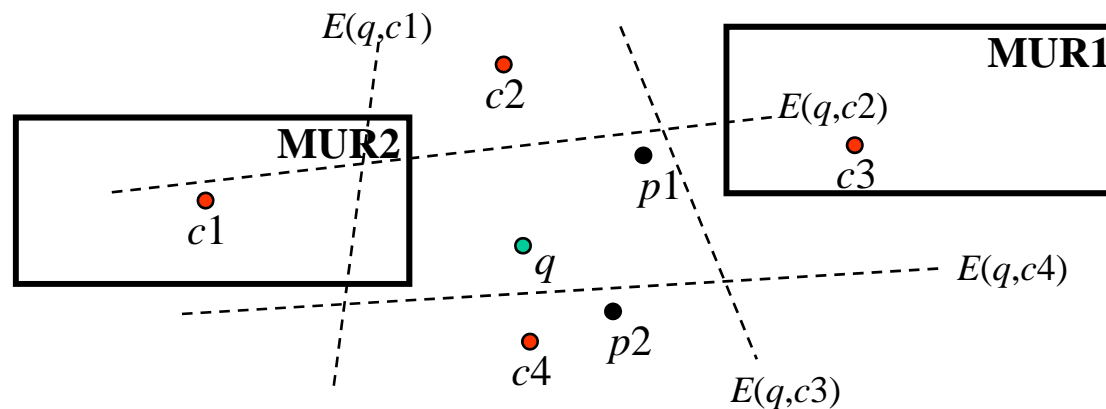
e hinter der Hyperebene
=> **prune e**

Min/Max-dist Pruning:



$MaxDist(e,o) > MinDist(e,q)$
=> **prune e nicht!**

- Algorithmus: Filter-Schritt (Skizze)
 - Berechne ein NN-Ranking der DB
 - Solange noch Objekte im Ranking sind:
 - » Rufe getNext() auf
 - » Wenn aktueller Punkt p nicht „hinter“ einer Voronoi-Ebene liegt, konstruiere neue Voronoi-Ebene $E(p,q)$; p wird zur Kandidatenmenge hinzugefügt
 - » Punkte/Directoryseiten, die „hinter“ einer der Voronoi-Ebenen liegen (außer der eigenen), können aus dem Ranking/Kandidatenmenge gelöscht werden
 - Punkte, die die Ebenen bestimmen, müssen verfeinert werden, d.h. für diese Punkte muss jeweils eine NN-Anfrage berechnet werden
- Beispiel



Bisherige Kandidaten:

$\{c1, c2, c3, c4\}$

**Inhalt des Rankings
(ungeordnet):**

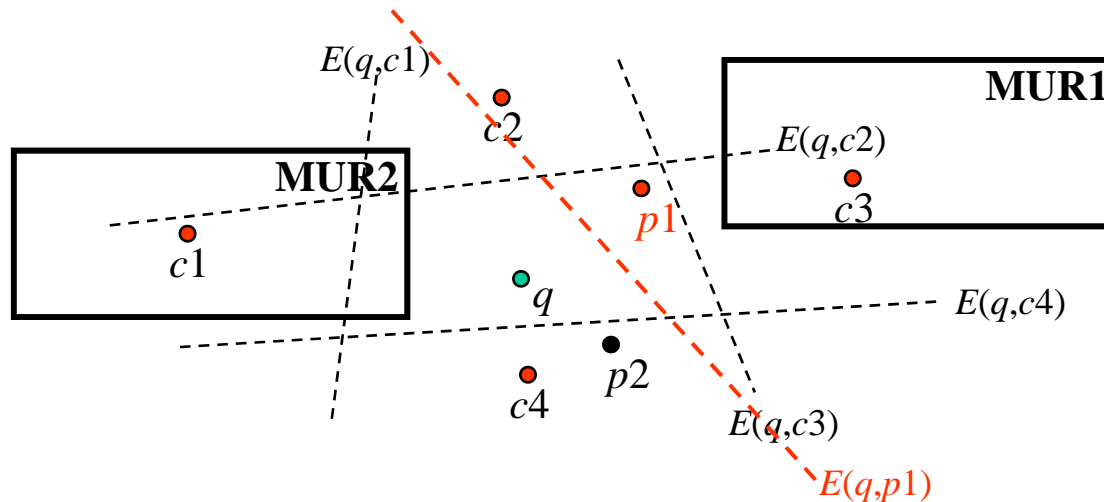
MUR1: nicht verfeinern

MUR2: verfeinern

$p1$: verfeinern

$p2$: nicht verfeinern

- Verfeinerung von $p1$
 - » Streiche $c2$ und $c3$ aus Kandidatenliste (liegt nun hinter $E(q,p1)$)
 - » MUR2 muss weiterhin verfeinert werden



Bisherige Kandidaten:

$\{c1, c4, p1\}$

**Inhalt des Rankings
(ungeordnet):**

MUR2: verfeinern

- Vorteil
 - » k kann beliebig sein (Ausschlusskriterium: Objekt/Seite muss hinter k Ebenen liegen)
 - » Keine vorberechneten Distanzen, daher keine Update-Problematik und bessere Speicherkomplexität
- Nachteil
 - » Nur für Vektordaten
 - » Teurer Verfeinerungsschritt (eine NN-Anfrage pro Kandidat)
 - » Teilweise komplexe Ebenenverwaltung

- Trimmen

- Partielles abschneiden (trimmen) von Seitenregionen (Rechtecken) bzgl. einer Pruningebene



- Anpassung der $MINDIST(q,e)$ nach dem Trimmen
 - führt eventuell zur Erhöhung der $MINDIST(q,e)$
 - erhöht die Chance, dass Seitenregion e früher ausgefiltert (geprunt) werden kann.

- Algorithmus (Filter):

Algorithm TPL-filter(q) /* q is the query point */

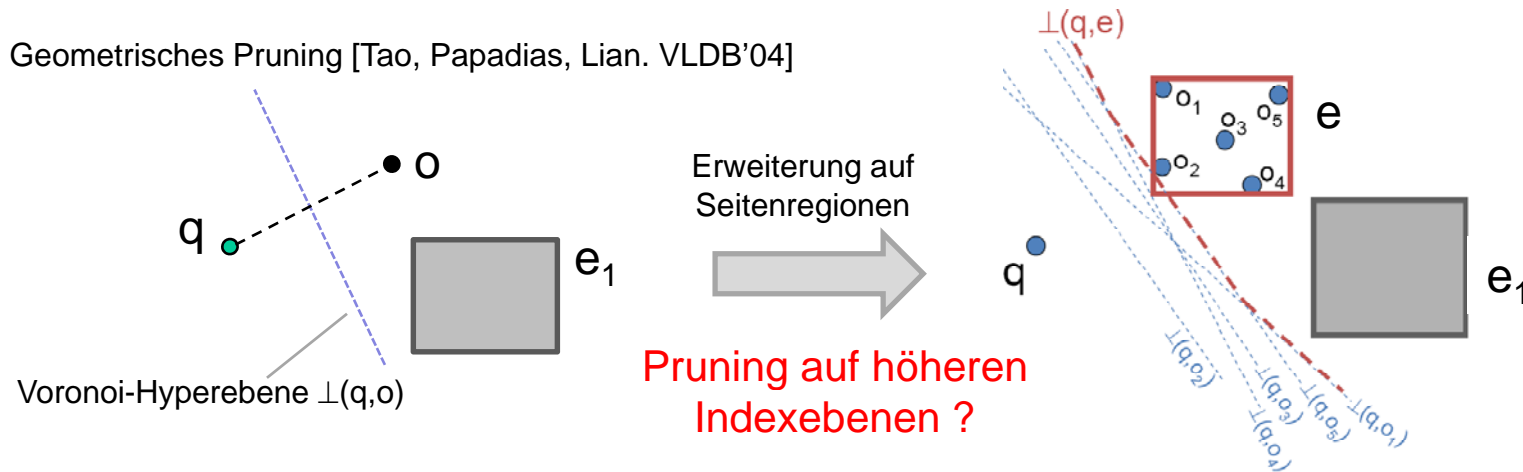
1. initialize a min-heap H accepting entries of the form (e, key)
2. initialize sets $S_{cnd}=\emptyset, S_{rfn}=\emptyset$
3. insert (R-tree root, 0) to H
4. while H is not empty
5. $(e, key)=\text{de-heap } H$
6. if (**trim**(q, S_{cnd}, e) $=\infty$) then $S_{rfn}=S_{rfn}\cup\{e\}$
7. else // entry may be or contain a candidate
8. if e is data point p
9. $S_{cnd}=S_{cnd}\cup\{p\}$
10. else if e points to a leaf node N
11. for each point p in N (sorted on $\text{dist}(p,q)$)
12. if (**trim**(q, S_{cnd}, p) $\neq\infty$) then insert $(p, \text{dist}(p,q))$ in H
13. else $S_{rfn}=S_{rfn}\cup\{p\}$
14. else // e points to an intermediate node N
15. for each entry N_i in N
16. $\text{mindist}(N_i^{\text{resM}}, q)=\text{trim}(q, S_{cnd}, N_i)$
17. if ($\text{mindist}(N_i^{\text{resM}}, q)=\infty$) then $S_{rfn}=S_{rfn}\cup\{N_i\}$
18. else insert $(N_i, \text{mindist}(N_i^{\text{resM}}, q))$ in H

End TPL-filter

falls e hinter einer bestehenden Pruningebene

Quelle: [Tao, Papadias, Lian. Int. Conf. Very Large Databases (VLDB), 2004]

- Weitere Eigenschaft der Geometrischen RNN-Suche
 - Ausschluß (Pruning) von Punkten/Seiten nur aufgrund anderer Punkte
 - Ausschlußkriterium könnte auch vollständig auf Directory-Ebene angewandt werden, aber wie?
- Erweiterung des geometrischen Pruning-Konzepts auf Basis von Voronoi-Hyperebenen auf Seitenregionen



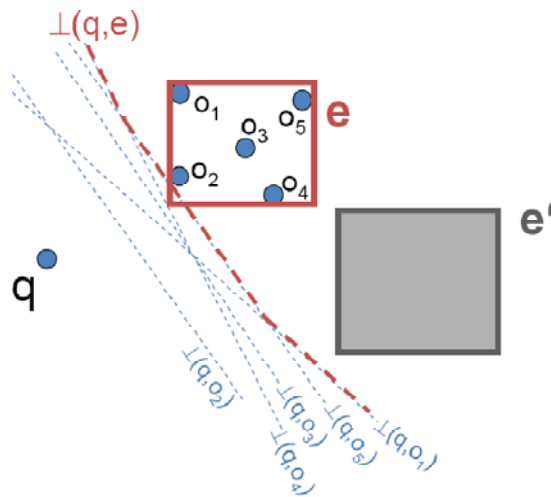
- Frage: Effiziente Repräsentation von Voronoi-Hyperebenen zwischen einem Punkt und einer Seitenregion ?
- Idee: Konservative Approximation der Hyperebenen
Eigenschaft: $\forall p \in \perp(q,e): \text{dist}(q,p) = \text{MaxDist}(e,p)$

– Erweiterung der Geometrischen RNN-Suche (gilt nur für Vektordaten!!!) [Kriegel, Kröger, Renz, Züfle: SSDBM, 2009]

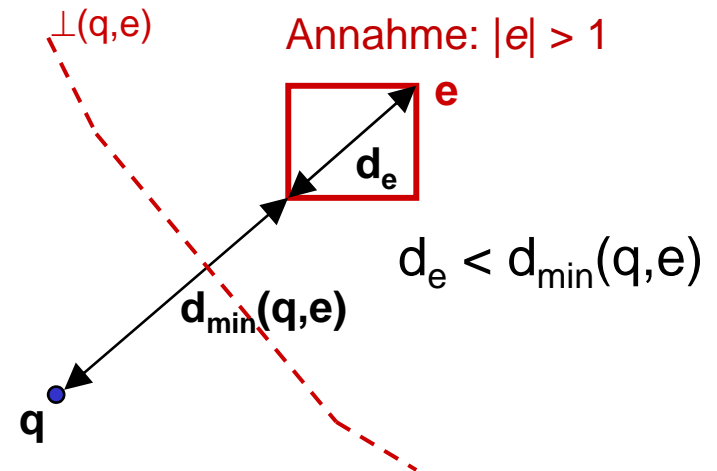
- Ziel: Pruning auf höheren Indexebenen
- Definition von Hyper-Ebenen zwischen Anfragepunkt und Seitenregion e
- Bilde konservative Approximation \mathcal{H} aller Hyperebenen bzgl. aller Punkte innerhalb der Seitenregion e
- Anzahl der konservativ approximierten Hyperebenen kann zum Ausschluß von Seitenregionen (Self/Mutual Pruning) verwendet werden (insbesondere auch für RkNN-Suche mit $k > 1$)

Seite e schließt Seite e' aus (k=1):

Seite e schließt sich selbst aus (k=1):

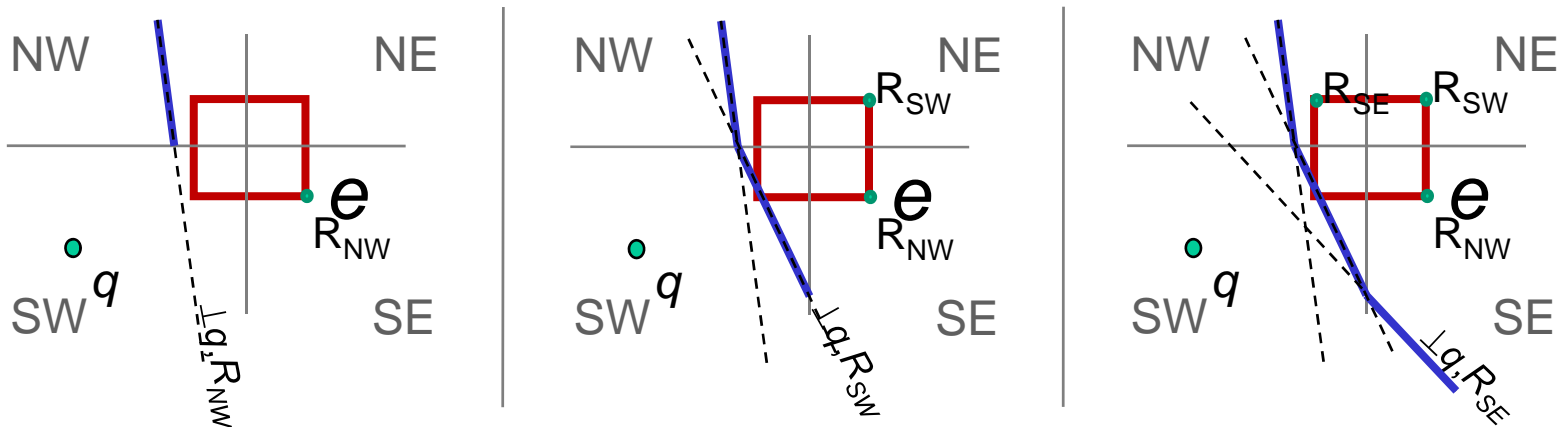


$\forall o \in e': o \notin RNN(q)$, da $\exists o_i \in e: e'$ hinter Hyperebene $\perp(q, o_i)$



- Berechnung der konservativen Approximation aller Hyperebenen zwischen Anfragepunkt q und Seitenregion e
 - Aufspaltung des Datenraums in 2^d Partitionen orientiert am Mittelpunkt der Seitenregion E (z.B. 4 Partitionen NW, NE, SE und SW in 2D Raum)
 - Für jede Partition P : Wähle Referenzpunkt $R \in e$, sodaß gilt:

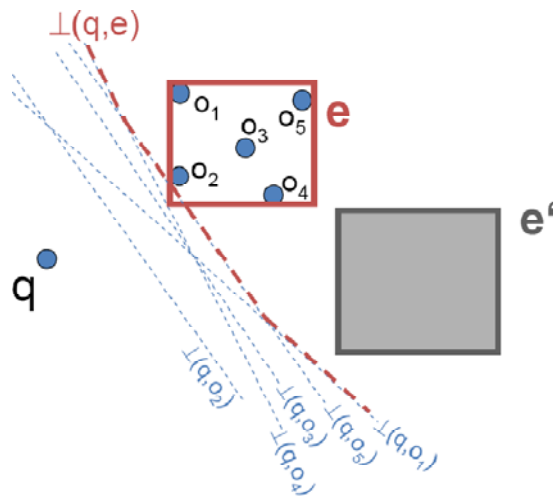
$$\forall p \in P, \forall e \in E: d(p, e) \leq d(p, R)$$
 - Bemerkung: Referenzpunkt eindeutig durch die gewählte Partitionierung
 - Für jede Partition P bilde Hyperebene zwischen q und Referenzpunkt zu P



- Die konservative Hyperebenen-Approximation wird aus $2^d - 1$ Hyperebenen gebildet

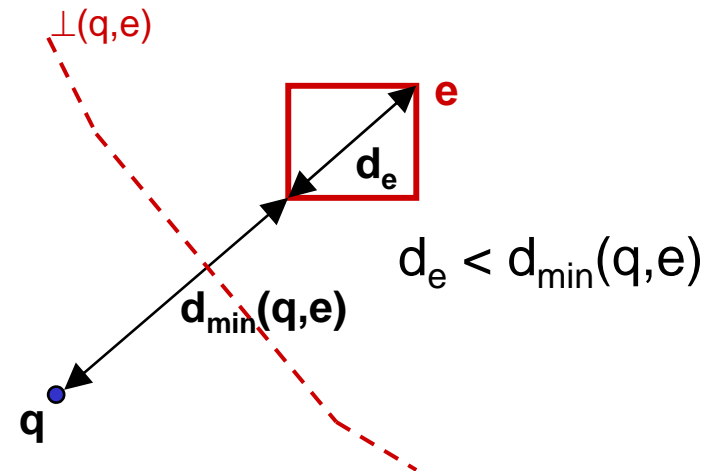
- Erweiterung der Geometrischen RkNN-Suche für $k > 1$
 - Verwende aR-Baum anstatt R-Baum, d.h. R-Baum mit zusätzlicher Angabe der Anzahl der Punkt-Objekte $|e|$ die innerhalb einer Seitenregion e organisiert werden.
 - Menge der Punkt-Objekte auf die sich eine konservative Approximation \mathcal{H} bezieht ist zur Anfragezeit bekannt.
 - => Anzahl N der Objekte, die näher an einem Objekt o bzw. einer Seitenregion e' sind als der Anfragepunkt q lässt sich einfach ermitteln
 - Prune Seitenregion e' (bzw. e' bei self pruning) falls $N > k$ (bzw. $N-1 > k$)

Seite e schließt Seite e' aus ($k > 1$):



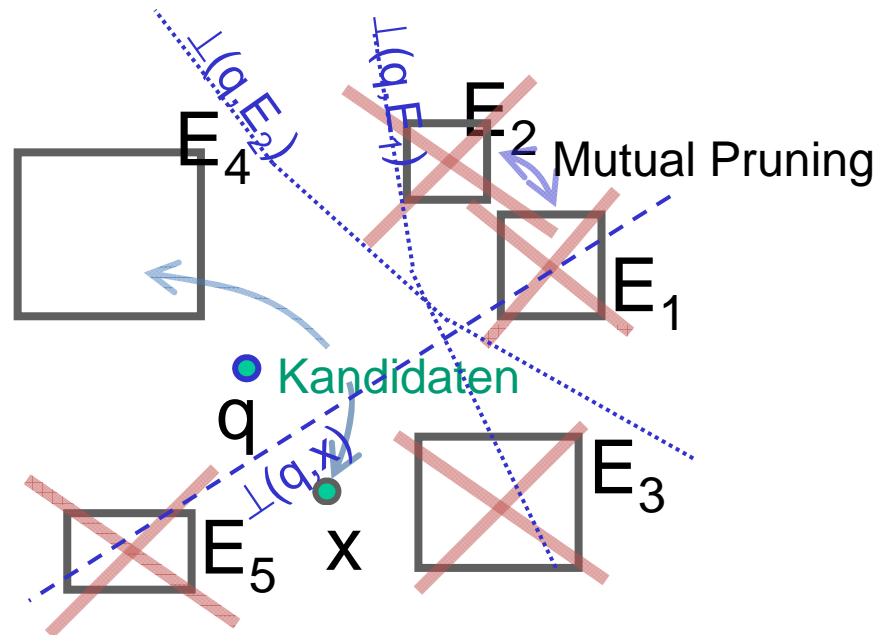
$\forall o \in e': o \notin RkNN(q)$, where $k \leq |e|$

Seite e schließt sich selbst aus ($k > 1$):



$\forall o \in e: o \notin RkNN(q)$, where $k \leq |e|-1$

- Beispiel:



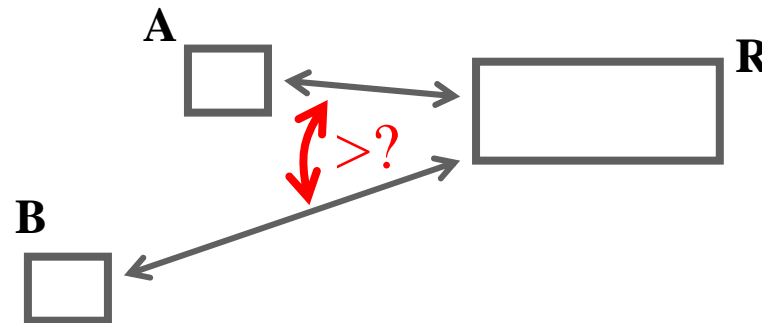
- Bemerkung zum „Optimalen“ Pruning:
 - Die Suche mittels Mutual-Pruning ohne Vorberechnung ist weniger selektiv als mit vorberechneten NN-Distanzen
 - ==> schlechteres Pruning-Verhalten als bei (reinen) Self-Pruning-Methoden

- Eigenschaften der Geometrischen RkNN-Suche
 - Vorteile:
 - » Vollständige Flexibilität bzgl. k
 - » Keine Zusätzliche Kosten für Änderungen im Index (Update-Kosten)
 - » Pruning-Filter selektiver als bei Min/Max-Dist-Ansatz
 - Nachteile:
 - » 2^d Hyperebenen pro Seitenregion müssen materialisiert werden => Overhead der Ebenenverwaltung (schlechte Performanz in höher-dimensionalen Räumen)
 - » Test ob Seitenregion geprunt werden kann ist teuer (2^d Ebenen-Pruning-Tests)
- Fazit:
 - Min/Max-Dist-Ansatz hat geringeres Pruningpotential als Geometrisches Pruning
 - Geometrisches Pruning eignet sich nur für niedrig-dimensionale Räume
- Gewünscht:
 - Verfahren mit „optimalen“ Pruningpotential, das auch für hoch-dimensionale Räume effizient funktioniert
 - Welche Beziehung gilt zwischen dem Min/Max-Dist-Basierten Ansatz und dem Geometrischen Pruning Ansatz?

– Generelle Problemstellung:

Gegeben:

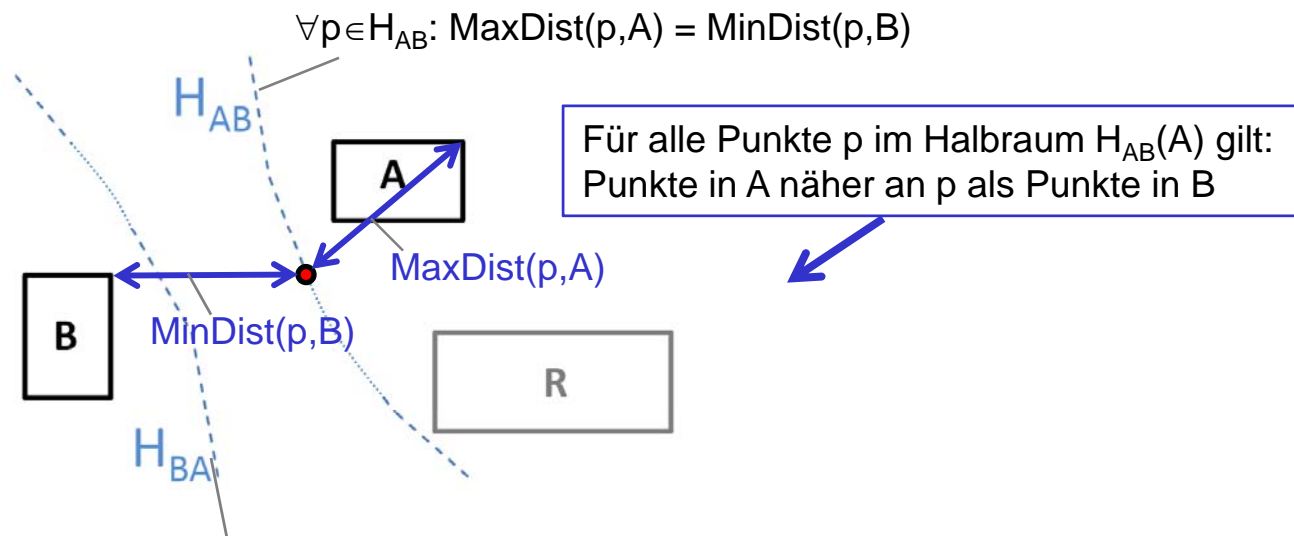
Drei Punktobjekt-Approximationen A, B und R gegeben als achsenparallele Rechtecke



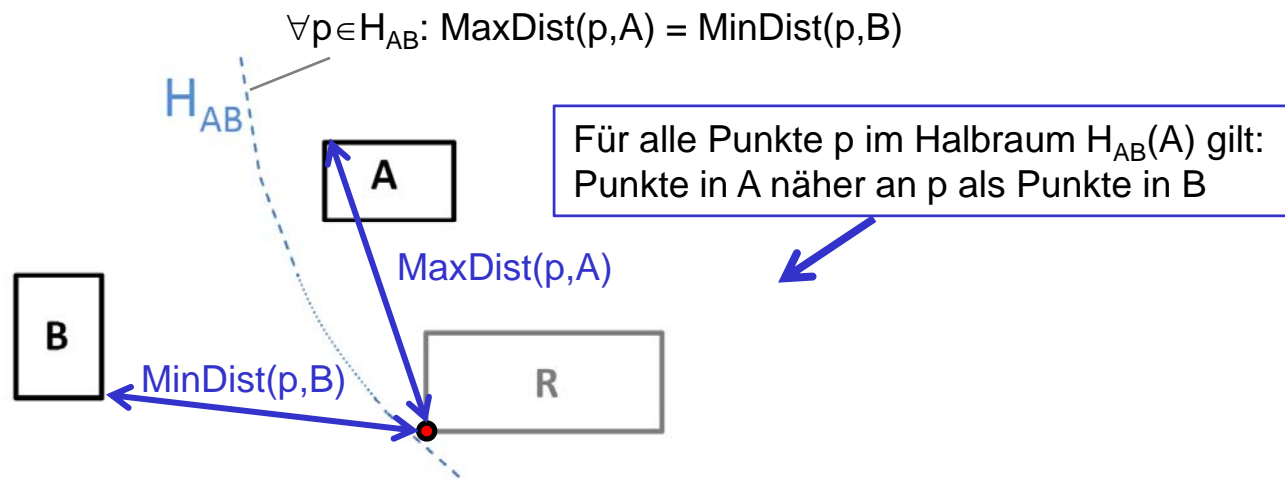
Frage:

Wie kann man effizient bestimmen ob die Objekte in R näher an den Objekten in A oder näher an den Objekten in B liegen

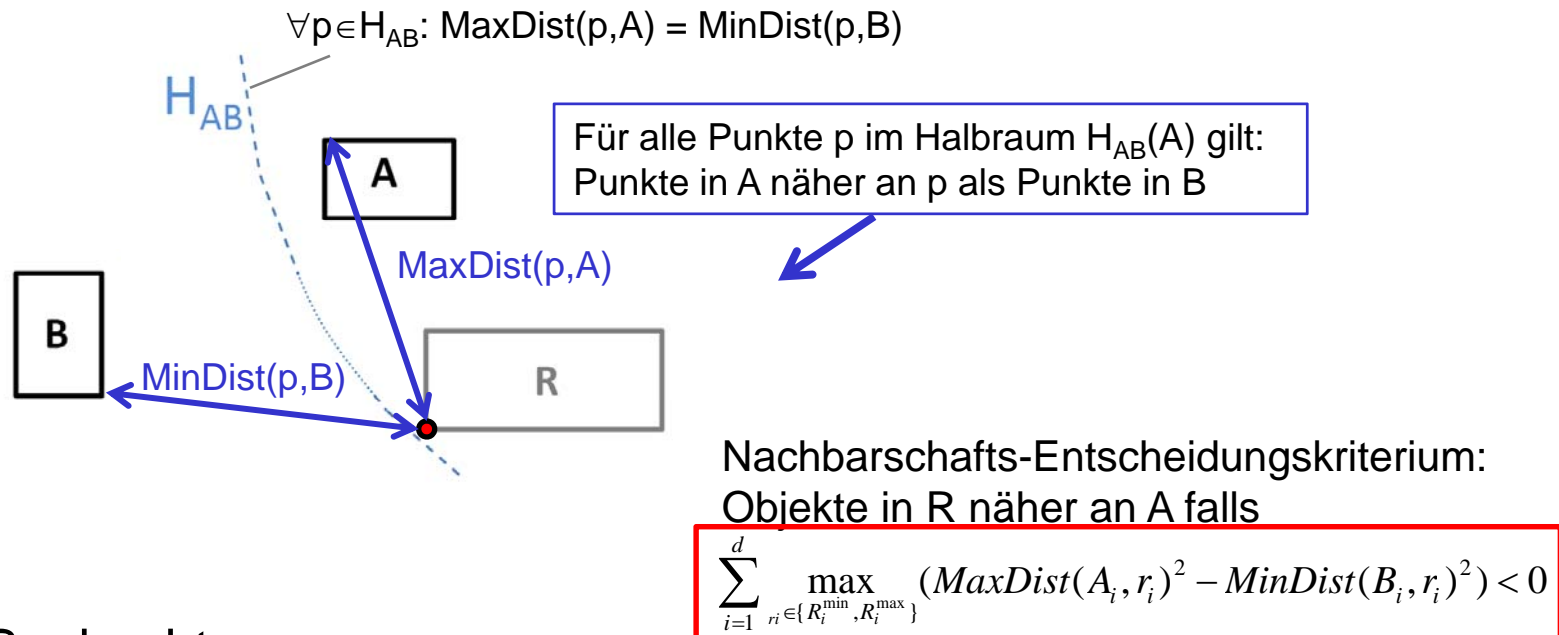
→ Nachbarschafts-Entscheidungskriterium [SIGMOD 10]



- Beobachtung:
 - Halbraum $H_{AB}(A)$ in der A liegt ist konvex
 - alle Eckpunkte von R liegen in $H_{AB}(A) \Rightarrow R$ liegt vollständig in $H_{AB}(A)$



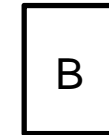
- Beobachtung:
 - Halbraum $H_{AB}(A)$ in der A liegt ist konvex
 - alle Eckpunkte von R liegen in $H_{AB}(A) \Rightarrow R$ liegt vollständig in $H_{AB}(A)$
- Idee:
 - Finde Eckpunkt p von R bei dem $\text{MaxDist}(p,A) - \text{MinDist}(p,B)$ den größten Wert hat
 - Falls dieser Wert kleiner 0 (d.h. p in $H_{AB}(A)$)
 - Alle Ecken von R in $H_{AB}(A) \Rightarrow$ Region R vollständig in $H_{AB}(A)$



- Beobachtung:
 - Halbraum $H_{AB}(A)$ in der A liegt ist konvex
 - alle Eckpunkte von R liegen in $H_{AB}(A) \Rightarrow R$ liegt vollständig in $H_{AB}(A)$
- Idee:
 - Finde Eckpunkt p von R bei dem $\text{MaxDist}(p,A) - \text{MinDist}(p,B)$ den größten Wert hat
 - Falls dieser Wert kleiner 0 (d.h. p in $H_{AB}(A)$)
 - Alle Ecken von R in $H_{AB}(A) \Rightarrow$ Region R vollständig in $H_{AB}(A)$

• Herleitung

- Idee: Umformung der Distanzrelation:



A näher an R als B?



$$\forall a \in A, b \in B, r \in R : dist(a, r) < dist(b, r)$$

$$\Leftrightarrow \forall r \in R : \text{MaxDist}(A, r) < \text{MinDist}(B, r) \leftarrow \text{konservative Distanzabschätzung zu A und B}$$

$$\Leftrightarrow \forall r \in R : \sqrt[p]{\sum_{i=1}^d \text{MaxDist}(A_i, r_i)^p} < \sqrt[p]{\sum_{i=1}^d \text{MinDist}(B_i, r_i)^p}$$

$$\Leftrightarrow \forall r \in R : \sum_{i=1}^d \text{MaxDist}(A_i, r_i)^p - \sum_{i=1}^d \text{MinDist}(B_i, r_i)^p < 0$$

$$\Leftrightarrow \max_{r \in R} \left(\sum_{i=1}^d (\text{MaxDist}(A_i, r_i)^p - \text{MinDist}(B_i, r_i)^p) \right) \leq 0$$

Reduzierung des Problems auf die einzelnen Dimensionen

$$\Leftrightarrow \sum_{i=1}^d \max_{r_i \in R_i} (\text{MaxDist}(A_i, r_i)^p - \text{MinDist}(B_i, r_i)^p) < 0$$

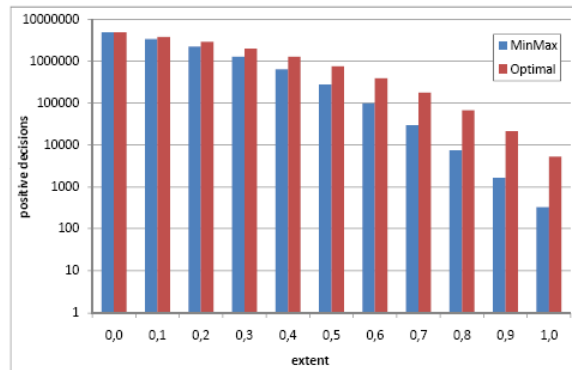
Reduzierung des Problems auf Extremwerte in R_i (pro Dimension nur 2 Werte)

$$\Leftrightarrow \sum_{i=1}^d \max_{r_i \in \{R_i^{\min}, R_i^{\max}\}} (\text{MaxDist}(A_i, r_i)^p - \text{MinDist}(B_i, r_i)^p) \leq 0$$

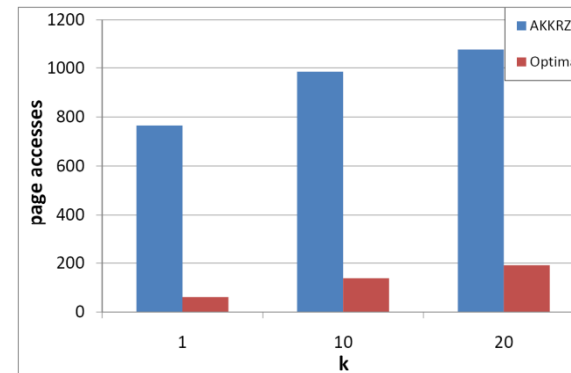
- Vorteil: Berechnungskomplexität $O(d)$ (d = Dimensionalität)

=> geeignet auch für höherdimensionale Räume

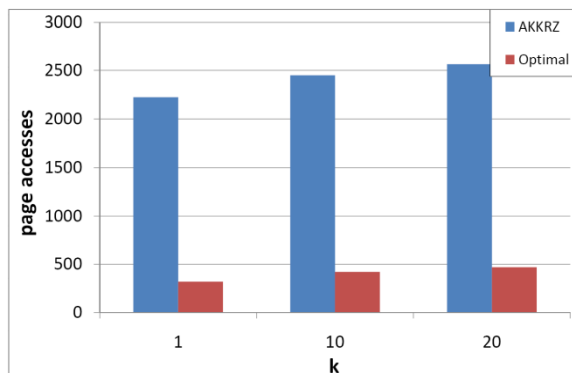
- Experimenteller Vergleich: Min/Max-Dist vs. Optimales Pruning:



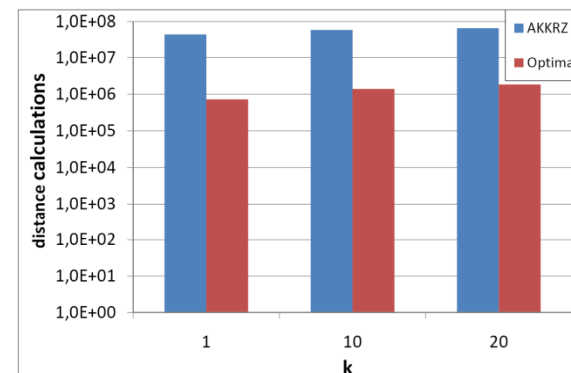
a) Anzahl der erkannten Ereignisse: Objekte in R näher an A als an B Datensatz (synthetisch): 10 Mil. Triple (A,B,R) von Rechtecken (gleichverteilt im Raum $[0,1]^2$)



b) Anzahl der Seitenzugriffe für RkNN-Anfragen mit Index Datensatz (synthetisch): 100K Punktobjekte (5D, gleichverteilt)



c) Anzahl der Seitenzugriffe für RkNN-Anfragen mit Index (I/O-Kosten) Datensatz (real): 581K Punktobjekte (10D)



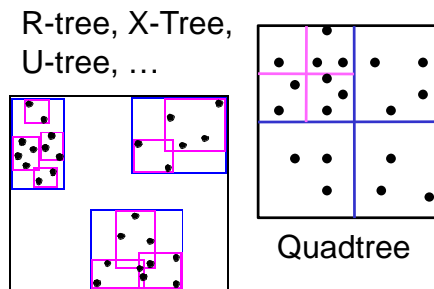
d) Anzahl der Distanzberechnungen für RkNN-Anfragen mit Index (CPU-Kosten), Datensatz (real): 581K Punktobjekte (10D)

– Weiterer Anwendungsbereiche des Nachbarschaftskriteriums:

Im Prinzip lässt sich das Nachbarschaftskriterium überall dort einsetzen wo Objekte durch achsenparallele Rechtecke approximiert sind und Nachbarschaftsbeziehungen relevant sind

Beispiele:

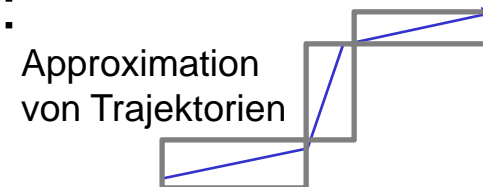
Spatial Index:



Anfragen:

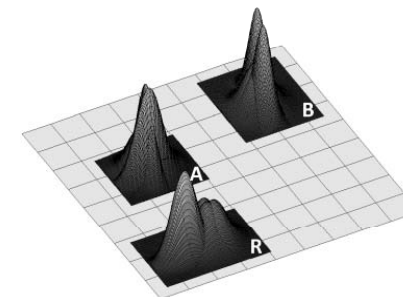
- kNN, RkNN
- continuous kNN (RkNN)
- probabilistic kNN (RkNN)
- inverse ranking
- group Nearest Neighbor
- clustering
- usw.

Spatio-Temporal Data:



Unsichere Daten:

Approximation von unsicheren Objekten



– Zusammenfassung

	Verfahren	Vorteile	Nachteile
self-pruning	RNN-Tree	Sehr gute Performanz, da keine Verfeinerung nötig	k fix; nur für Vektordaten; Updateproblematik; wenig selektiv bei normalen NN-Queries
	RdNN-Tree	Sehr gute Performanz, da keine Verfeinerung nötig; auf allgemein metrische Daten erweiterbar	k fix; Updateproblematik
	MRkNNCoP-Tree	variables k (mit Einschränkung); für allgemein metrische Daten	Verfeinerung nötig, Updateproblematik
mutual-pruning	Min/Max-Dist-basierte RkNN Suche	variables k ; geringe Kosten, für metrische Daten, erlaubt Pruning auf Directory-Ebene	schlechtere Filterselektivität
	Geometrische (Voronoi-basierte) RkNN Suche	variables k ;	Nur für Vektordaten; Verfeinerung nötig
	Erweiterte geometrische RkNN Suche	variables k ; erlaubt Pruning auf Directory-Ebene; optimale Filterselektivität	Nur für Vektordaten; teure Verwaltung der Hyperebenen
	optimal-pruning-basierte RkNN Suche	variables k , geringe Kosten erlaubt Pruning auf Directory-Ebene; optimale Filterselektivität;	Nur für Vektordaten;