Baumverfahren

Ausgangspunkt:

punktförmige hochdimensionale Feature-Objekte

- B-Baum eindimensional

Abbildung eines mehrdimensionalen Raums auf eine Dimension im Allgemeinen nicht distanzerhaltend möglich

- Simplex mit n+1 Punkten im n-dimensionalen Raum)

Fazit: mehrdimensionale Indexverfahren nötig

Baumverfahren (2)

Grundidee hierarchischer Indexierungsverfahren:

- Beschreiben von Punktmengen durch geometrische, umschreibende Regionen (Cluster)
- bei der Suche Test und evtl. Ausschluss von Clustern
- Cluster können sich gegenseitig enthalten
  → Halbordnung und daher
  Hierarchie (Hasse-Diagramm) oder Baum

Unterscheidungskriterien von Baumstrukturen

Merkmal	Unterscheidung
Cluster-Bildung	global zerlegend (space partioning) lokal gruppierend (data partitioning)
Cluster-Überlappung	überlappend disjunkt
Balance	balanciert unbalanciert
Objektspeicherung	Blätter und Knoten Blätter
Geometrie	Hyperkugel Hyperquader Hyperellipsoid ...

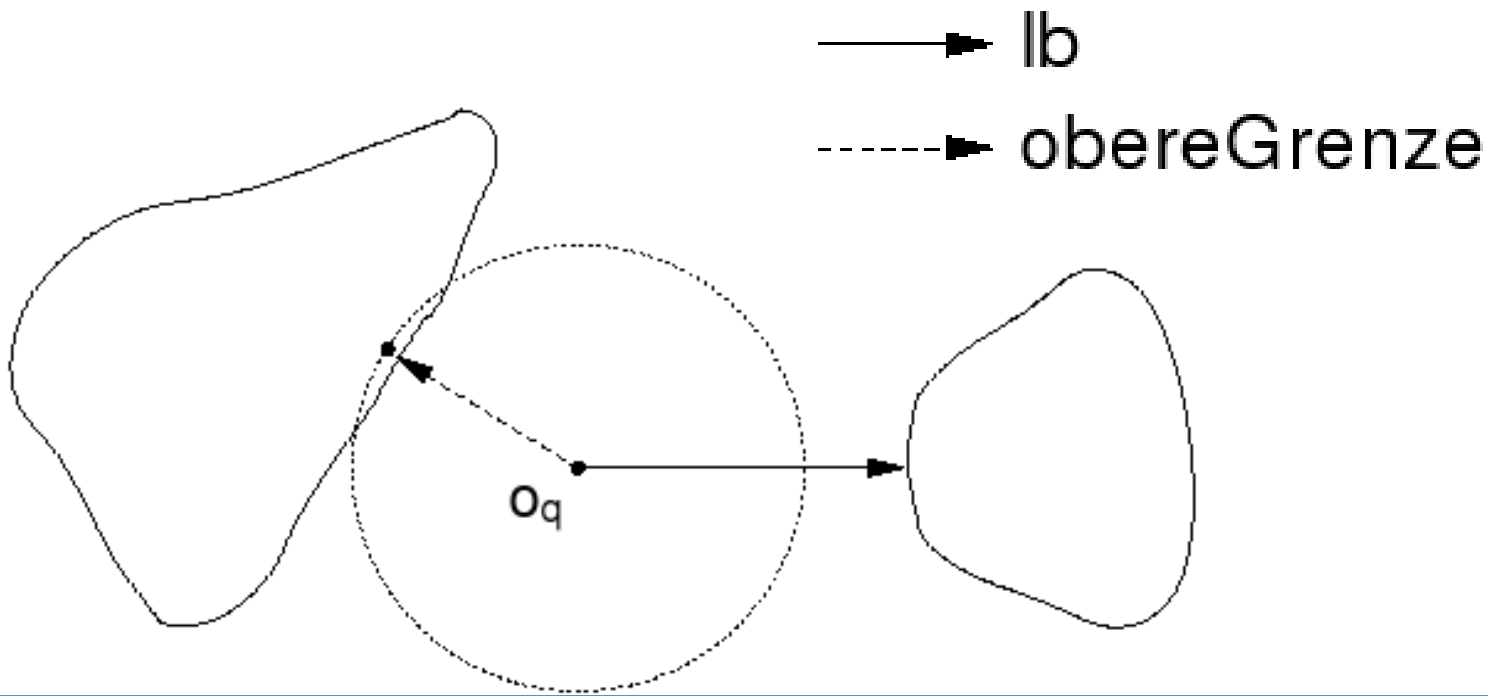
Algorithmen zur Berechnung des nächsten Niveaus

Anfragepunkt

Existenz zweier Distanzfunktionen

- Distanz zwischen zwei Punkten
- minimale Distanz zwischen q und potentiellem Clusterpunkt eines Clusters

- geht von Objektspeicherung in Blättern aus
- realisiert Tiefensuche
- verwendet dynamisch angepasste Distanz `obereGrenze` zu NN-Kandidat



Branch-and-Bound-Algorithmus (2)


```
[1]  real obereGrenze = ∞
[2]  punkt naechsterNachbar = nil
[3]
[4]  procedure BranchAndBound(punkt q,knoten T)
[5]
[6]    sortiere Subknoten von T aufsteigend nach lb-Distanz zu q
[7]    for each Subknoten k von T do
[8]      if k ist Blatt then
[9]        for each Punkt p in k do
[10]         distanz = d(p,q) //Distanz zwischen p und q
[11]         if distanz < obereGrenze then do
[12]           obereGrenze = distanz
[13]           naechsterNachbar = p // NN-Kandidat
[14]         end if
[15]       end for
[16]     else do
[17]       lb=lb(q,k) //kleinstmögliche Distanz von q zu k
[18]       if lb > obereGrenze then
[19]           schließe k von allen weiteren Betrachtungen aus
[20]       else BranchAndBound(q,k)
[21]     end else
[22]   end for
[23] end procedure
```

Roussopoulos/Kelly/Vincent 1995

spezieller Branch-and-Bound-Algorithmus zur Nächste-Nachbarsuche

Feature-Objekte als hochdimensionale Objekte

für Baum mit lokal gruppierenden Hyperquadern und Feature-Objekten in den Blättern

Hyperquader als MBR (engl. minimum bounding rectangle)

- jede Hyperfläche berührt mind. ein Feature-Objekt von außen
- MBR wird durch zwei Punkte (s, t) (innerste und äußerste Ecke) identifiziert

MIN-Distanz: minimal mögliche Distanz zwischen Anfragepunkt q und MBR (s,t)

mit $$MINDIST(q,(s,t)) = \sum_{i=1}^{n} |q[i] - r[i]|^2$$

$$r[i] = \begin{cases} s[i] & \text{wenn } q[i] < s[i] \\ t[i] & \text{wenn } q[i] > t[i] \\ q[i] & \text{sonst.} \end{cases}$$

MINMAX-Distanz: maximal möglicher Abstand zum nächsten Nachbarn in (s,t)

Ausnutzen der Minimaleigenschaft eines MBR

$$MINMAXDIST(q,(s,t)) =$$

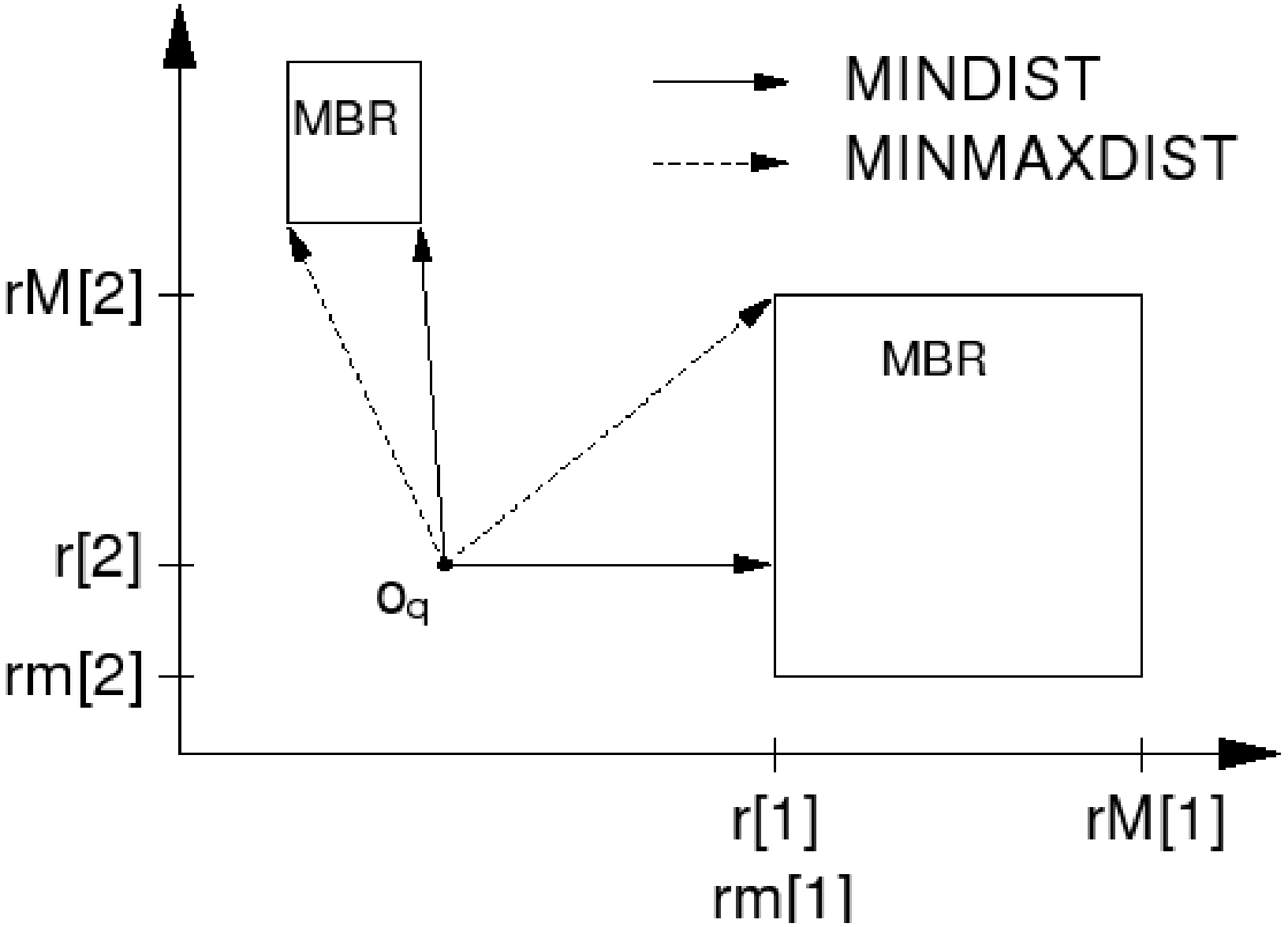
$$\min_{1\leq k\leq n}\left(|q[k]-rm[k]|^2+\sum_{\substack{i\neq k\\1\leq i\leq n}}|q[i]-rM[i]|^2\right)$$

mit:

$$rm[k]=\begin{cases}s[k] & \texttt{wenn } q[k]\leq\frac{(s[k]+t[k])}{2}\\ t[k] & \texttt{sonst}\end{cases}\quad\texttt{und}$$

$$rM[i]=\begin{cases}s[i] & \texttt{wenn } q[i]\geq\frac{(s[i]+t[i])}{2}\\ t[i] & \texttt{sonst}\end{cases}$$

MIN- und MINMAX-Distanzen graphisch



Reduzierung des Suchaufwands

Sortierung der Kindsknoten anhand
- MIN-Distanz (optimistisch) oder
- MINMAX-Distanz (pessimistisch)

3 Strategien zur Suchaufwandreduzierung
(`obereGrenze` ist Distanz zum NN-Kandidaten)

1. $MINDIST(q,(s,t)) > MINMAXDIST(q,(s',t')):$
   MBR (s,t) braucht nicht aufgesucht zu werden

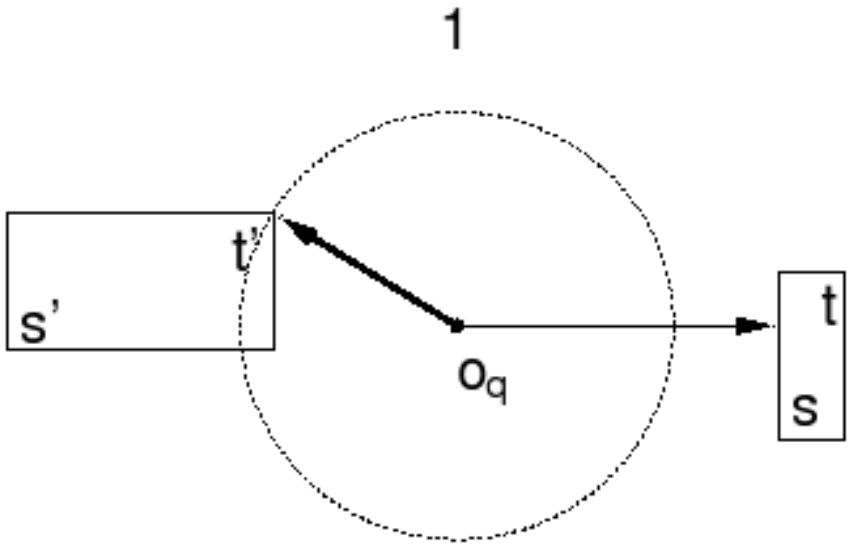
2. $obereGrenze > MINMAXDIST(q,(s,t)):$
   $\rightarrow \; obereGrenze = MINMAXDIST(q,(s,t))$

3. $obereGrenze < MINDIST(q,(s,t)):$
   MBR (s,t) braucht nicht aufgesucht zu werden

3 Strategien zur Suchaufwandreduzierung
(`obereGrenze` ist Distanz zum NN-Kandidaten)

1. $MINDIST(q,(s,t)) > MINMAXDIST(q,(s',t')):$
   MBR (s,t) braucht nicht aufgesucht zu werden
2. $obereGrenze > MINMAXDIST(q,(s,t)):$
   $\rightarrow obereGrenze = MINMAXDIST(q,(s,t))$
3. $obereGrenze < MINDIST(q,(s,t)):$
   MBR (s,t) braucht nicht aufgesucht zu werden

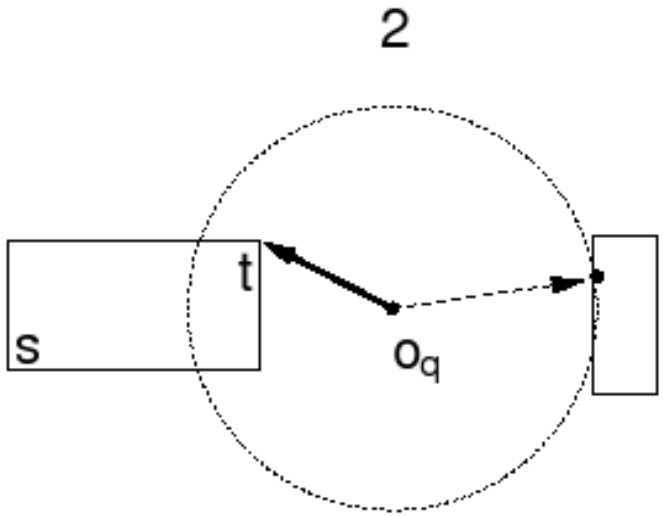
obereGrenze
MINMAXDIST
MINDIST



3 Strategien zur Suchaufwandreduzierung
(`obereGrenze` ist Distanz zum NN-Kandidaten)

1. $MINDIST(q,(s,t)) > MINMAXDIST(q,(s',t')):$
   MBR (s,t) braucht nicht aufgesucht zu werden
2. $obereGrenze > MINMAXDIST(q,(s,t)):$
   $\rightarrow obereGrenze = MINMAXDIST(q,(s,t))$
3. $obereGrenze < MINDIST(q,(s,t)):$
   MBR (s,t) braucht nicht aufgesucht zu werden

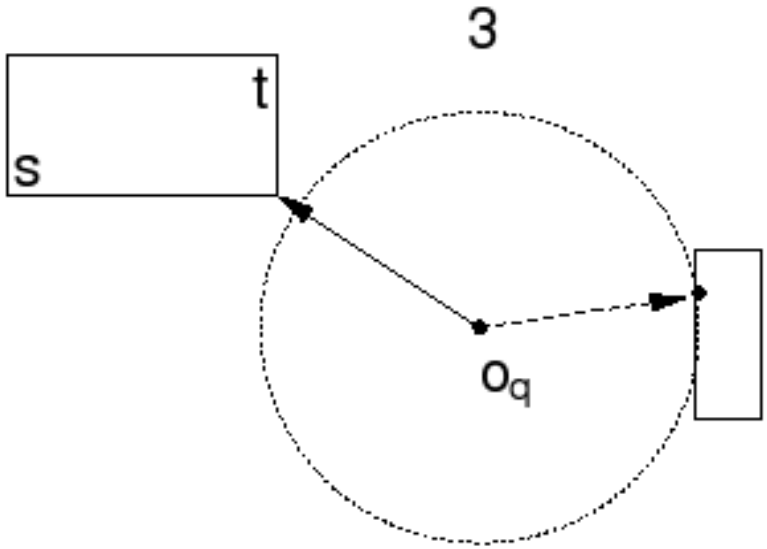
obereGrenze
MINMAXDIST
MINDIST



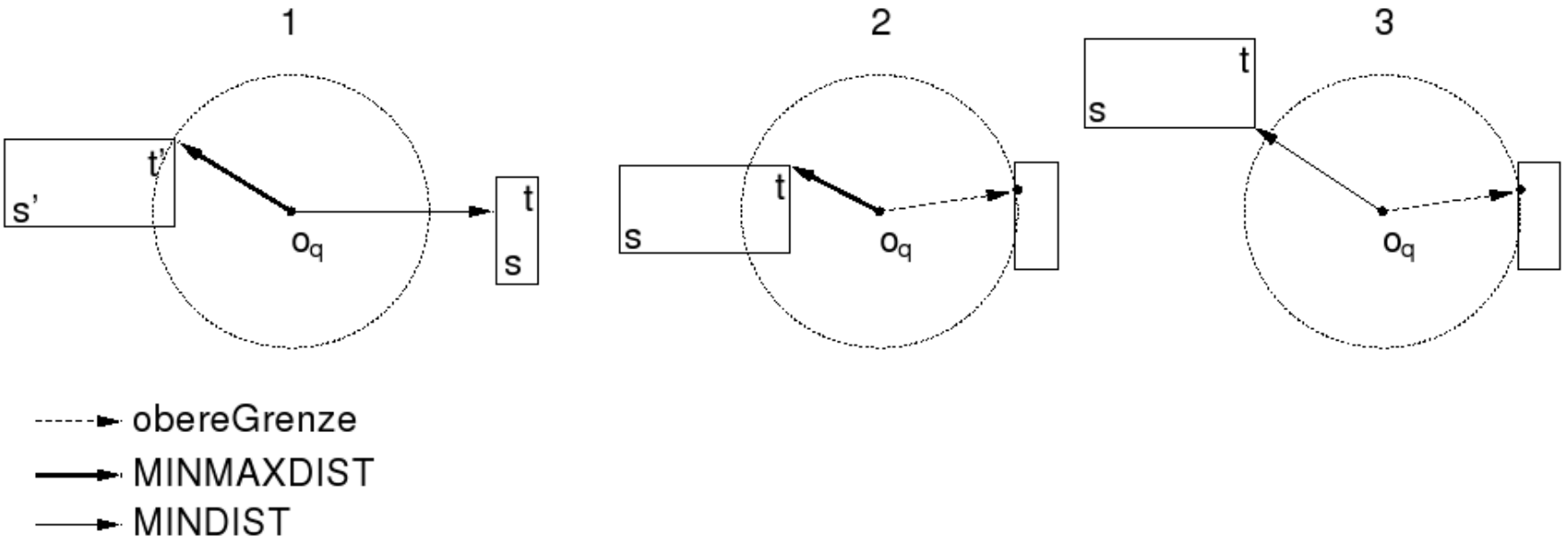
3 Strategien zur Suchaufwandreduzierung
(`obereGrenze` ist Distanz zum NN-Kandidaten)

1. $MINDIST(q,(s,t)) > MINMAXDIST(q,(s',t')):$
   MBR (s,t) braucht nicht aufgesucht zu werden
2. $obereGrenze > MINMAXDIST(q,(s,t)):$
   $\rightarrow obereGrenze = MINMAXDIST(q,(s,t))$
3. $obereGrenze < MINDIST(q,(s,t)):$
   MBR (s,t) braucht nicht aufgesucht zu werden

obereGrenze
MINMAXDIST
MINDIST



Reduzierung des Suchaufwands graphisch



RKV-Algorithmus


```
[1]  procedure RKV(punkt q,knoten T,real obereGrenze,
[2]                objekt naechsterNachbar)
[3]      knoten neuerKnoten
[4]      brancharray branchList
[5]      real distanz
[6]      objekt o
[7]      if T ist Blattknoten then
[8]        for each o in T do
[9]          distanz ist Distanz zwischen q und o
[10]         if distanz <= obereGrenze then do
[11]           obereGrenze = distanz
[12]           naechsterNachbar = o
[13]         end if
[14]       end for
[15]     else do
[16]       branchList sei Liste von Kindknoten aus T
[17]       branchList nach MINDIST oder MINMAXDIST sortieren
[18]       branchList nach Strategie 1 kürzen
[19]       obereGrenze nach Strategie 2 reduzieren
[20]       branchList nach Strategie 3 kürzen
[21]       for each neuerKnoten in branchList do
[22]         RKV(q,neuerKnoten,obereGrenze,naechsterNachbar)
[23]         branchList nach Strategie 3 kürzen
[24]       end for
[25]     end do
[26] end procedure
```

Modifikation des Algorithmus:

sortierte Warteschlange zur Verwaltung der
k Nächste-Nachbarschaftskandidaten

`obereGrenze` ist Distanz zum letzten Kandidat

Bewertung RKV-Algorithmus

getNext-Anfragen werden nicht unterstützt
(Problem aufgrund der Tiefensuche)

ursprünglich für euklidsche Distanz entwickelt; funktioniert auch auf anderen Distanzfunktionen, so lange MIN- und MINMAX-Distanzen berechnet werden können

Einschränkung auf MBRs als Clustergeometrien
→ ansonsten statt MINMAX- die MAX-Distanz verwenden

Hanrich/Hjaltason/Samet

Algorithmus für `getNext`-Anfragen

statt Tiefensuche Verwendung einer global sortierten Warteschlange:

- enthält Knoten, Blätter und Feature-Objekte mit minimalen Distanzen `lb` zum Anfragepunkt
- legt Abarbeitungsreihenfolge aufgrund aufsteigender Distanz fest
  → nur Kopfelemente werden entnommen

HS-Algorithmus (2)

1. Initialisierung mit Wurzelelementen

2. wenn entnommenes Kopfelement Knoten, dann werden Kinder eingefügt

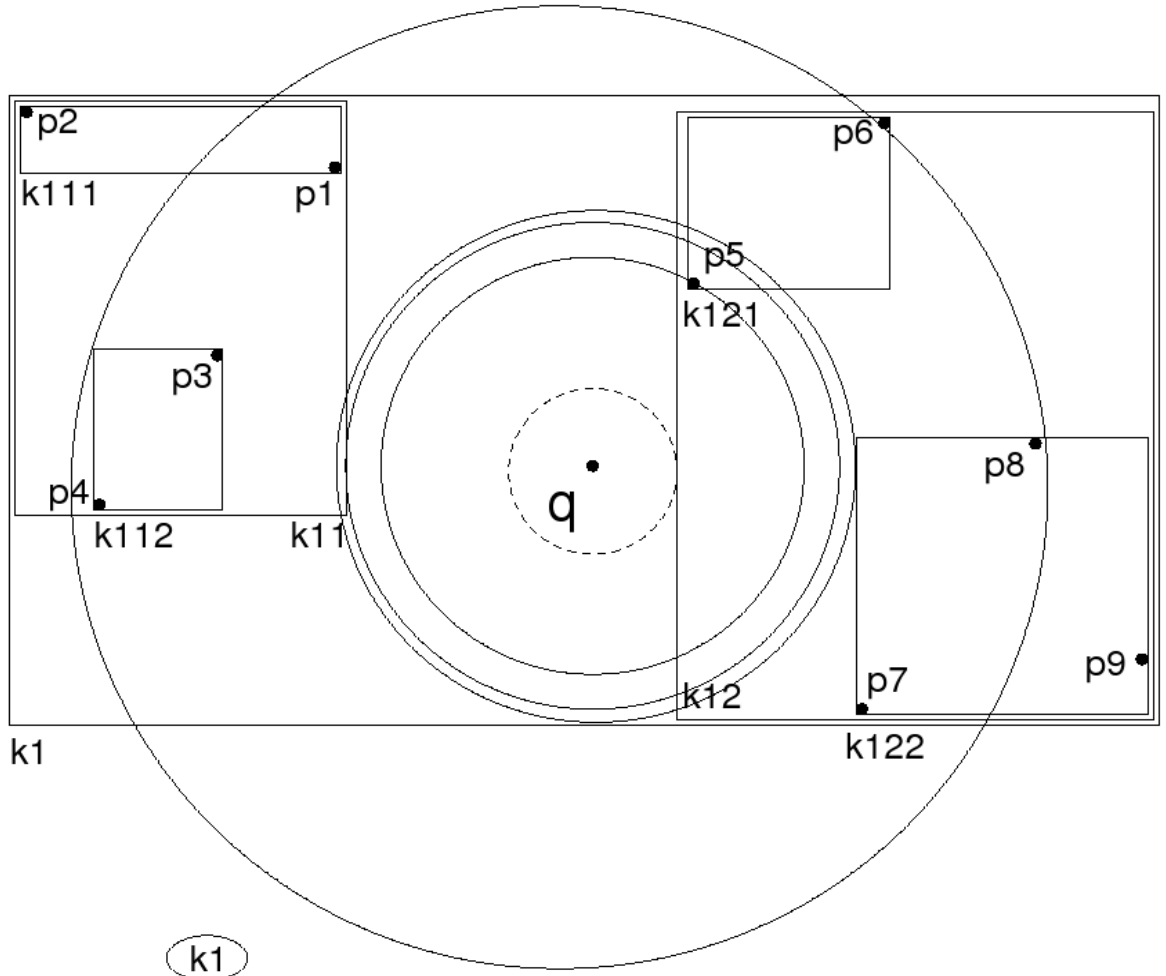
3. wenn entnommenes Kopfelement Blatt, dann werden Feature-Objekte eingefügt

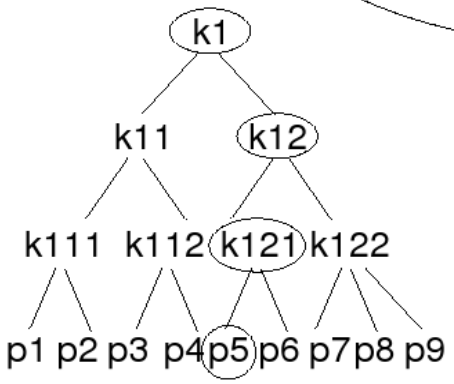
4. wenn entnommenes Kopfelement Feature-Objekt
   → Ausgabe

HS-Algorithmus (3)

```
[1]  procedure HS(punkt q,knoten T)
[2]      pqueue queue // Priority Queue
[3]      queueEintrag element // Priority-Queue-Eintrag
[4]      objekt fo // Feature-Objekt
[5]      knoten k
[6]      enqueue(queue, T, lb(q,T))
[7]      while not isEmpty(queue) do
[8]        element = dequeue(queue)
[9]        if element ist Feature-Objekt then do
[10]         while element = first(queue) do
[11]           deleteFirst(queue) // Duplikate entfernen
[12]         end do
[13]         ausgabe(element) // getNext-Resultat ausgeben
[14]       end do
[15]       else if element ist Blattknoten then
[16]         for each fo in element do
[17]           enqueue(queue, fo, lb(q,fo))
[18]         end do
[19]       else // innerer Knoten
[20]         for each Kind k in element do
[21]           enqueue(queue, k, lb(q,k))
[222]        end do
[23]       end if
[24]     end do
[25] end procedure
```

HS-Algorithmus graphisch





Queue: k1
k12 k11
k121 k11 k122
p5 k11 k122 p6

Bewertung HS-Algorithmus

- gut geeignet für `getNext`-Anfragen
- unabhängig von Cluster-Geometrie
- Problem: Warteschlange kann zu groß für Hauptspeicher werden
  → aufwändige Auslagerung auf Festplatte notwendig
- Navigation „springt“ wegen Warteschlangensortierung; keine Tiefen/Breitensuche
  → teure Festplattenzugriffe