

„M-Baum: Eine effiziente Indexstruktur für Ähnlichkeitsanfragen in metrischen Räumen“^{*}



08. Julii 2004

Vorlesung Multimedia-Datenbanksysteme

Dank an *Olaf Schmitt* für Erstellung der Folien

^{*}P. Ciaccia, M. Patella, P. Zezula; 23. VLDB-Konferenz, 1997, Athen

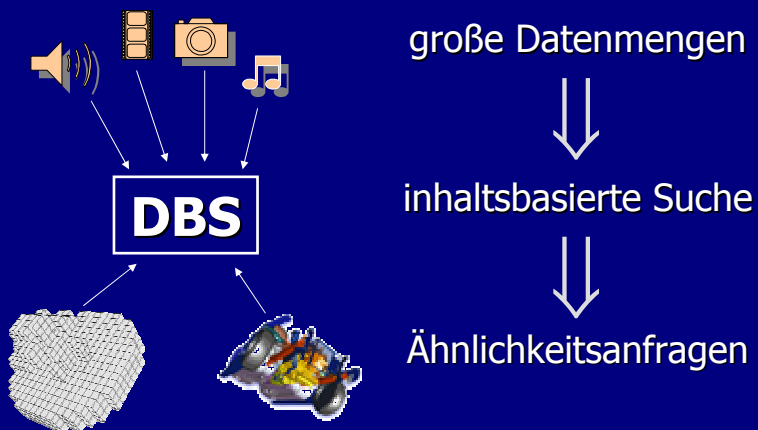
Übersicht

- Motivation
- Grundlagen
- M-Baum
- Ähnlichkeitssuche im M-Baum
- Erstellung eines M-Baums
- Leistungsevaluation
- Zusammenfassung

Übersicht

- **Motivation**
- Grundlagen
- M-Baum
- Ähnlichkeitssuche im M-Baum
- Erstellung eines M-Baums
- Leistungsevaluation
- Zusammenfassung

Motivation



Motivation (i)

- Objekte werden durch mehrdimensionale Feature dargestellt
- mehrdimensionale Distanzfunktionen



Räumliche Indexstrukturen:

z.B. **R-Baum**

ABER:

Voraussetzung für solche Strukturen:

1) **Vektorraum**

nicht alle Daten „passen“ in Vektorräume, z.B. Punktmengen, ...

2) L_p -Metrik (z.B. Euklidische Metrik)

Annahme:

Distanzberechnung \equiv trivialer Operation



bei MM-Anwendungen häufig komplexe Distanzfunktionen !

Motivation (ii)

Suche nach allgemeineren Ansätzen: **Metrischer Baum**

+ lediglich durch Metrik beschränkt

+ Optimierung auf Anzahl der Distanzberechnungen

- bisherige Implementierungen nicht für dynamische DB-Anwendungen geeignet

- keine Optimierung auf I/O-Kosten wie bei SAM's

Ziel:

Suche **dynamischen, balancierten, auf I/O Kosten** optimierenden **metrischen Baum**

Übersicht

- Motivation
- **Grundlagen**
- M-Baum
- Ähnlichkeitssuche im M-Baum
- Erstellung eines M-Baums
- Leistungsevaluation
- Zusammenfassung

Grundlagen

Metrischer Raum:

Ein Metrischer Raum ist ein Paar $M = (D, d)$, wobei D der Wertebereich der Featurewerte ist und d eine totale Distanzfunktion (Metrik) mit folgenden Eigenschaften ist:

1. $d(O_x, O_y) = d(O_y, O_x)$ symmetrisch
2. $d(O_x, O_y) > 0$ für $(O_x \neq O_y)$ nicht negativ
 $d(O_x, O_x) = 0$
3. $d(O_x, O_y) \leq d(O_x, O_z) + d(O_z, O_y)$ Dreiecksungl.

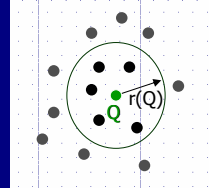
Grundlagen (i)

Ähnlichkeitsanfragen:

Rangequery:

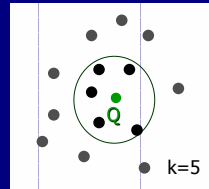
finde alle Objekte, die von Q max. $r(Q)$ entfernt sind

$$range(Q, r(Q)) = \{O_j \mid d(O_j, Q) \leq r(Q)\}$$



k NN Query:

finde die k nächsten Nachbarn von Q



Grundlagen (ii)

Darstellung des Metrischen Raums im \mathbb{R}^2 :

- nur relative Abstände zwischen Objekten
- keine absoluten Positionen

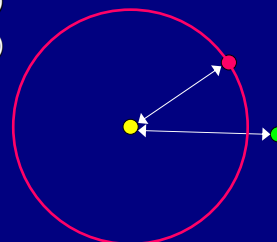
bekannt: $d(A, B)$

bekannt: $d(A, B)$

$d(A, C)$

Darstellung:

Darstellung:



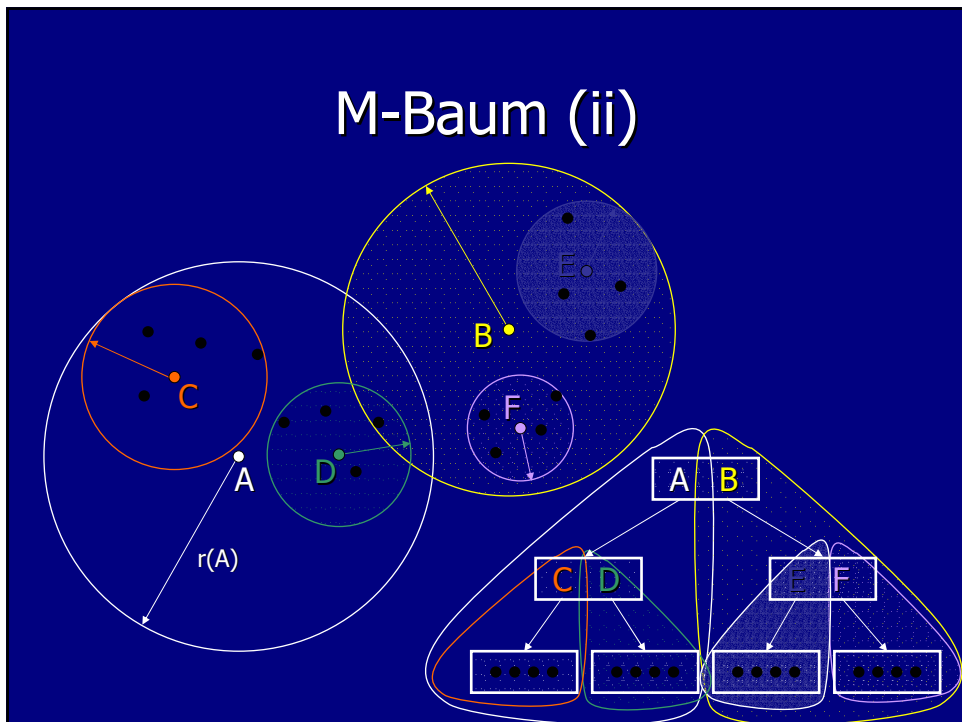
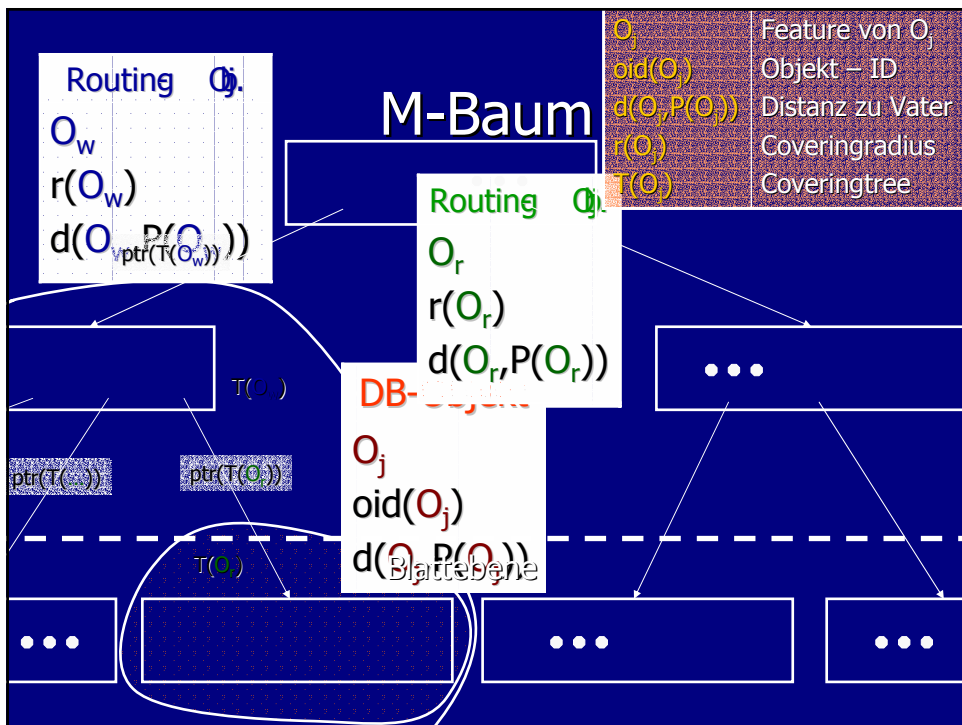
Übersicht

- Motivation
- Grundlagen
- **M-Baum**
- Ähnlichkeitssuche im M-Baum
- Erstellung eines M-Baums
- Leistungsevaluation
- Zusammenfassung

M-Baum

M-Baum:

- + metrisch
- + seitenorientiert
- + balanciert
- + für große dynamische Datenmengen geeignet
- + erweitert Anwendungsgebiet von z.B. R-Baum beträchtlich
- + optimiert auf Distanzberechnungen und I/O



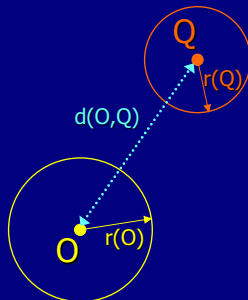
Übersicht

- Motivation
- Grundlagen
- M-Baum
- **Ähnlichkeitssuche im M-Baum**
- Erstellung eines M-Baums
- Leistungsevaluation
- Zusammenfassung

Ähnlichkeitssuche im M-Baum

Wann kann ein Teilbaum $T(O)$ von der Suche ausgeschlossen werden?

Lemma 1: $d(O, Q) > r(O) + r(Q) \Rightarrow \forall Q \in T(O) : d(O, Q) > r(Q)$



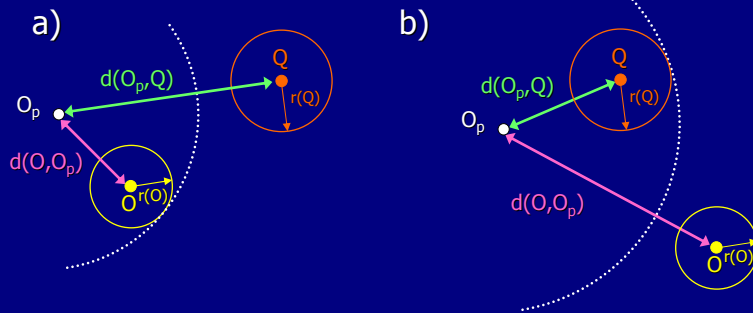
ABER: Berechnung von $d(O, Q)$ nötig!

Ähnlichkeitssuche im M-Baum (i)

Wie kann ein Teilbaum $T(O)$ ohne neue Distanzberechnung ausgeschlossen werden?

Lemma 2:

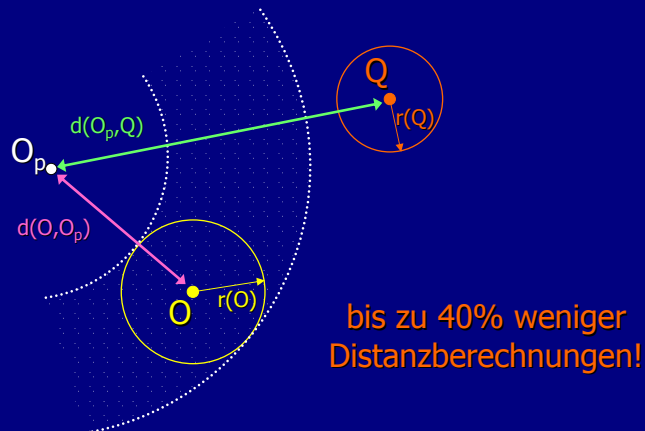
$$|d(O_p, Q) - d(O, O_p)| > r(Q) + r(O) \Rightarrow d(O, Q) > r(Q) + r(O)$$



Ähnlichkeitssuche im M-Baum (ii)

$$|d(O_p, Q) - d(O, O_p)| > r(Q) + r(O) \Rightarrow d(O, Q) > r(Q) + r(O)$$

a) + b)

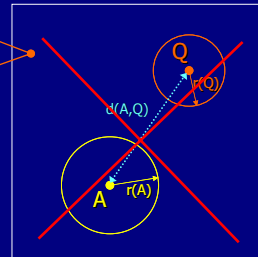
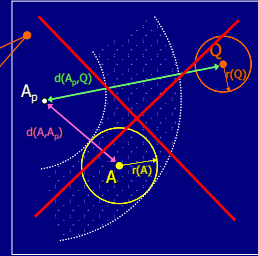


Rangequery

```

01 RS(N:node,Q:query_object,r(Q):search_radius)
02 {
03     let  $O_p$  be parent object of node N;
04     if N is not a leaf then
05     {
06          $\forall O$  in N do:
07         if  $|d(O_p, Q) - d(O, O_p)| \leq r(Q) + r(O)$  then
08         {
09             Compute  $d(O, Q)$ ;
10             if  $d(O, Q) \leq r(Q) + r(O)$  then
11                 RS(*ptr(T(O), Q, r(Q));
12         }
13     }
14     else // N is a leaf
15     {
16          $\forall O$  in N do:
17         if  $|d(O_p, Q) - d(O, O_p)| \leq r(Q)$  then
18         {
19             Compute  $d(O, Q)$ ;
20             if  $d(O, Q) \leq r(Q)$  then
21                 add oid(O) to result;
22         }
23     }
24 }

```



k-NN-Query

zwei globale Strukturen:

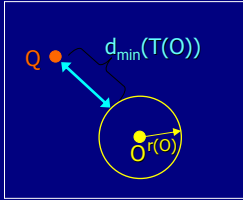
Array NN der Größe k

- enthält die aktuell k nächsten Nachbarn
- Struktur: $NN[i] = [oid(O), d(O, Q)]$
- d_i = Distanz von $NN[i]$
- d_k = dynamischer Suchradius

Priority Queue PR

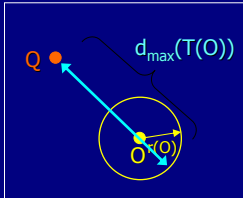
- Zeiger auf „aktive“ Teilbäume, d.h. in denen potentielle Kandidaten enthalten sind
- Struktur: $\{..., [T(O), d_{min}(T(O))], ...\}$

k-NN-Query (i)



untere Schranke:

- $d_{\min}(T(O)) = \max\{d(O, Q) - r(O), 0\}$
- wenn $d_{\min}(T(O)) > d_k$ schneide $T(O)$ ab



obere Schranke:

- $d_{\max}(T(O)) = d(O, Q) + r(O)$
- wenn $d_{\max}(T(O)) \leq d_k$, füge $T(O)$ als Platzhalter in NN ein
- Suchradius kann frühzeitig verringert werden

k-NN-Query (ii)

Auswahl des nächsten zu durchsuchenden Knotens:

- dynamisches Pruningkriterium ($> d_k$)
- daher ist Reihenfolge, in der Teilbäume durchsucht werden für die Laufzeit bedeutsam
- experimentell bestes Verfahren: wähle den Teilbaum $T(O)$ mit minimalem $d_{\min}(T(O))$

k-NN-Query (iii)

```

01 k-NN-Search(T:root_node,Q:query_object,k:integer)
02 {
03     PR := [T,_];
04     for i:=1 to k do: NN[i] = [_,\infty];
05     while PR\neq\emptyset do:
06     {
07         Next_Node=ChooseNode(PR);
08         k-NN_NodeSearch(Next_Node, Q, k);
09     }
10 }

```

```

01 ChooseNode(PR:priority_queue): node
02 {
03     let  $d_{\min}(T(O^*)) = \min\{d_{\min}(T(O))\}$ , considering all
04     the entries in PR;
05     remove entry [ptr(T(O^*)),  $d_{\min}(T(O^*))$ ] from PR;
06     return T(O^*);
07 }

```

k-NN-Query (iv)

```

01 k-NN_NodeSearch(N:node,Q:query_object,k:integer)
02 {
03     let  $O_p$  be the parent object of node N;
04     if N is not a leaf then {
05          $\forall O$  in N do:
06         if  $|d(O_p,Q) - d(O,O_p)| \leq d_k + r(O)$  then {
07             Compute  $d(O,Q)$ ;
08             if  $d_{\min}(T(O)) \leq d_k$  then {
09                 add [ptr(T(O)),  $d_{\min}(T(O))$ ] to PR;
10                 if  $d_{\max}(T(O)) < d_k$  then {
11                      $d_k = \text{NN\_Update}([_, d_{\max}(T(O))])$ ;
12                     Remove from PR all entries for which  $d_{\min}(T(O)) > d_k$ ;
13                 }
14             }
15         }
16     }
17     else { /* N is a leaf */
18          $\forall O$  in N do:
19         if  $|d(O_p,Q) - d(O,O_p)| \leq d_k$  then {
20             Compute  $d(O,Q)$ ;
21             if  $d(O,Q) \leq d_k$  then {
22                  $d_k = \text{NN\_Update}([oid(O), d(O,Q)])$ ;
23                 Remove from PR all entries for which  $d_{\min}(T(O)) > d_k$ ;
24             }
25         }
26     }
27 }

```

Übersicht

- Motivation
- Grundlagen
- M-Baum
- Ähnlichkeitssuche im M-Baum
- **Erstellung eines M-Baums**
- Leistungsevaluation
- Zusammenfassung

Erstellung eines M-Baums

In welchen Blattknoten soll O_n eingefügt werden?

- von der Wurzel beginnend bis auf Blattebene absinken
- wähle jeweils den Teilbaum $T(O)$ für den $r(O)$ nicht erhöht werden muss
- existieren mehrere mit $d(O, O_n) \leq r(O)$, wähle den $T(O)$, wobei $d(O, O_n)$ minimal ist
- sonst wähle den Teilbaum, dessen $r(O)$ am wenigsten wächst

Erstellung eines M-Baums (i)

```

01 Insert(N:node,entry(On):M-tree_entry)
02 {
03     let S be the set of entries in node N;
04     if N is not a leaf then {
05         let Sin = entries such that d(O,On) ≤ r(O);
06         if Sin ≠ ∅ then let entry(O*) ∈ Sin: d(O*,On) is minimum;
07         else {
08             let entry(O*) ∈ S: d(O*,On) - r(O*) is minimum;
09             let r(O*) = d(O*,On);
10         }
11         Insert(T(O*),entry(On));
12     }
13     else { /* N is a leaf */
14         if N is not full
15             then store entry(On) in N
16         else Split(N,entry(On));
17     }
18 }

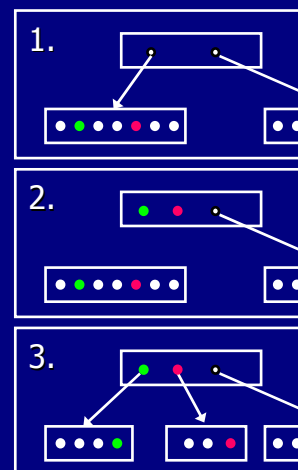
```

Erstellung eines M-Baums (ii)

Splitmanagement:

- Baum wächst „bottom up“
 - Überlauf der Knoten möglich
- geeignete Behandlung:
 - wähle zwei Routingobjekte O_{p1} O_{p2}
 - „promote“ diese in den Vaterknoten
 - partitioniere restliche Objekte um O_{p1} und O_{p2}

1.+2.+3. = Splitpolicy



Erstellung eines M-Baums (iii)

Anforderungen an eine Splitpolicy:

- minimales Volumen:
 - Coveringradius minimal
 - weniger indizierter „toter“ Raum
- minimale Überlappung
 - weniger Teilbäume müssen durchsucht werden

Erstellung eines M-Baums (iv)

Promote Strategien:

m_RAD: komplexeste bzgl. Distanzberechnungen. Für alle Paare wird eine Partitionierung durchgeführt und das Paar gewählt für das $r(O_{p1}) + r(O_{p2})$ minimal ist.

mM_RAD: wie m_RAD, nur wird das Paar gewählt, dessen $\max(r(O_{p1}), r(O_{p2}))$ minimal ist.

RANDOM: ein Paar wird zufällig ausgewählt
billig + als Referenz geeignet

Partitionierungsstrategie:

Generalized Hyperplane: jedes Objekt wird seinem nächsten Routingobjekt zugeordnet.

Übersicht

- Motivation
- Grundlagen
- M-Baum
- Ähnlichkeitssuche im M-Baum
- Erstellung eines M-Baums
- **Leistungsevaluation**
- Zusammenfassung

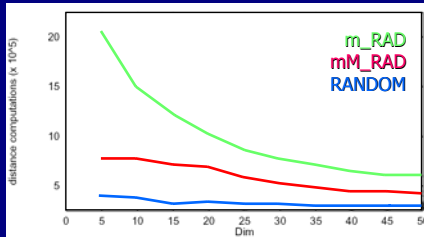
Leistungsevaluation

Betrachte Anzahl der **Distanzberechnungen** und
Anzahl der **I/O Operation** bei

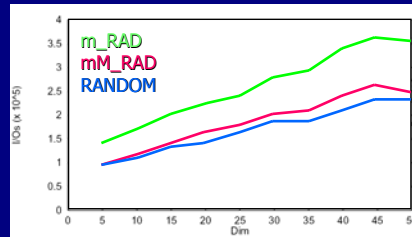
1. steigender Dimensionalität
2. wachsender Anzahl der Objekte

Leistungsevaluation (i)

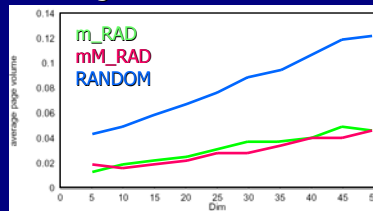
Erstellung eines M-Baums:



Anzahl Distanzberechnungen



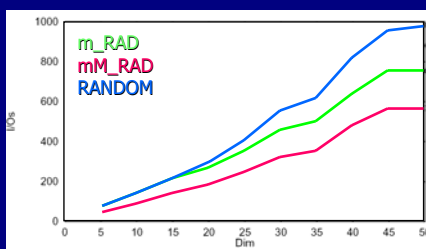
Anzahl I/O-Operationen



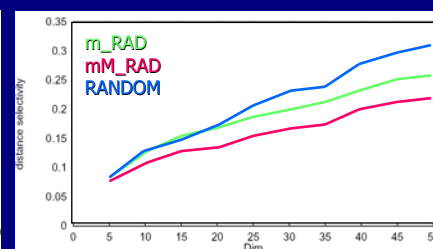
Güte des Baums

Leistungsevaluation (ii)

Ausführen einer 10-N-Query



Anzahl I/O-Operationen

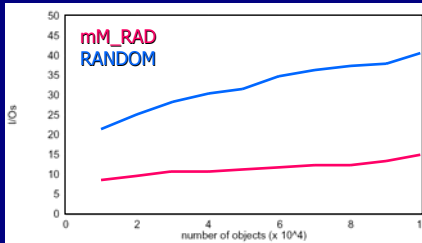


distance-selectivity

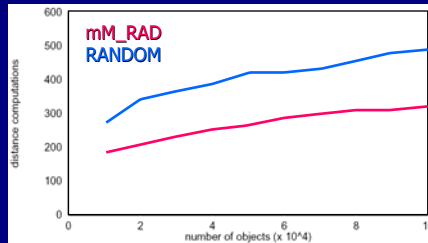
$$\text{distance selectivity} = \frac{\text{\#benötigter Dist. Berechn.}}{\text{\#Objekte}}$$

Leistungsevaluation (iii)

Ausführen einer 10 N Query



Anzahl I/O-Operationen

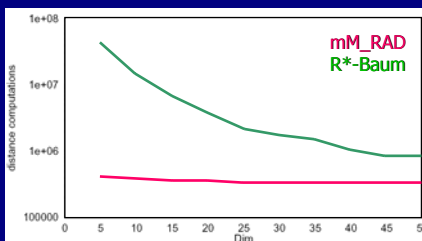


Anzahl Distanzberechnungen

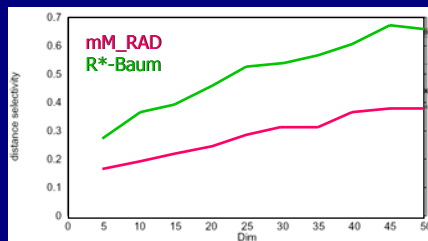
skaliert gut mit wachsender Datenmenge

Leistungsevaluation (iv)

M Baum vs R*- Baum



Anzahl Distanzberechnungen bei Baumerstellung



distance-selectivity bei 10-NN-Query

I/O Kosten sind ähnlich zwischen M und R*- Baum

Übersicht

- Motivation
- Grundlagen
- M-Baum
- Ähnlichkeitssuche im M-Baum
- Erstellung eines M-Baums
- Leistungsevaluation
- **Zusammenfassung**

Zusammenfassung

M-Baum ist eine Indexstruktur mit folgenden Eigenschaften:

- seitenorientiert, dynamisch
- kann metrischen Raum indizieren
- ist sowohl auf CPU ~~last~~ als auch I/O ~~last~~ optimiert
- skaliert mit wachsender Datenmenge und steigender Dimension gut

Weitere Aspekte

- Integration in DB
- Filtering M-tree

Extensible Indexing Framework

SQL '99

deklarative Einbettung

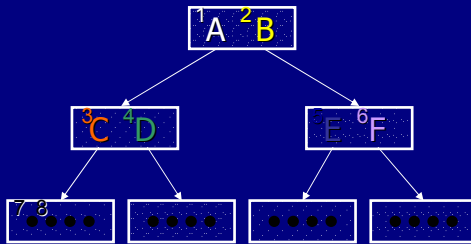
eigene Indexstruktur

Verwaltung	Anfragebearbeitung
index_create()	index_open()
index_drop()	index_fetch()
index_insert()	index_close()

relationale Abbildung

Datenbank-Kern

Relationale Abbildung



Relationale Abbildung des M-Baums:

par-id	son-id	mt-obj	row-id	...
root	1
root	2
1	3
1	4
2	5
2	6
3	7	...	R1	...
3	8	...	R2	...
...

Indizierte Tabelle:

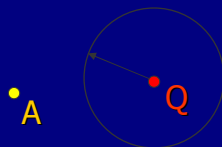
row-id	mt-obj	...
R1
R2
...

Filtering M-tree

Motivation: weitere Reduktion der Distanzberechnungen

- Filter:**
- Filterdistanz \leq exakte Distanz
 - Filterdistanz effizienter zu berechnen

Beispiel: range query



$\text{filter}(A, Q) \notin \text{range}(Q)$



$\text{exakt}(A, Q) \notin \text{range}(Q)$