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

m-tree

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

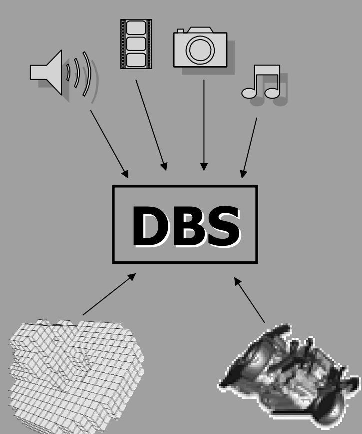
Übersicht

- 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



große Datenmengen

$\Downarrow$

inhaltsbasierte Suche

$\Downarrow$

Ähnlichkeitsanfragen

Motivation (i)

- Objekte werden durch mehrdimensionale Feature dargestellt
- mehrdimensionale Distanzfunktionen

$\Downarrow$

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. Euklidsche Metrik)

Annahme:

Distanzberechnung ≡ trivialer Operation

bei MM-Anwendungen häufig komplexe Distanzfunktionen !

Motivation (ii)

Suche nach allgemeineren Ansätzen: Metrischer Baum

+ lediglich durch Metrik beschränkt	- bisherige Implementierungen nicht für dynamische DB-Anwendungen geeignet
+Optimierung auf Anzahl der Distanzberechnungen	- 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. $d(O_x, O_y) > 0 \quad \textit{für}\ (O_x \neq O_y)$ 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.

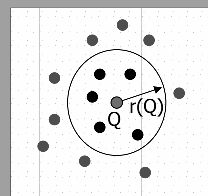
Grundlagen (i)

Ähnlichkeitsanfragen:

Rangequery:

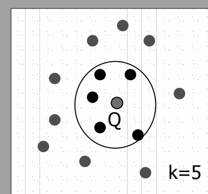
finde alle Objekte, die von Q max. r(Q) entfernt sind

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



k NN Query:

finde die k nächsten Nachbarn von Q



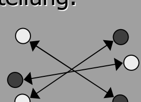
Grundlagen (ii)

Darstellung des Metrischen Raums im $\Re^2$:

- nur relative Abstände zwischen Objekten
- keine absoluten Positionen

bekannt: d(A,B)

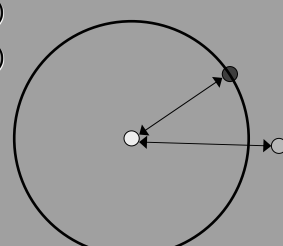
Darstellung:



bekannt: d(A,B)

d(A,C)

Darstellung:



Übersicht

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

M-Baum

M-Baum:

+ metrisch

+ seitenorientiert

+ balanciert

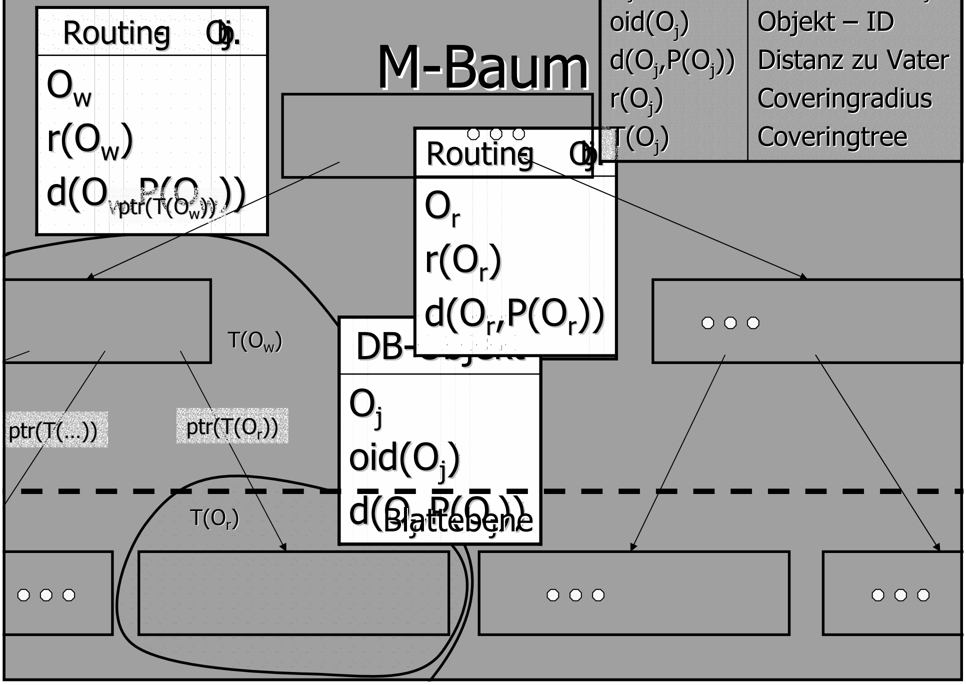
+ für große dynamische Datenmengen geeignet

+ erweitert Anwendungsgebiet von z.B. R-Baum beträchtlich

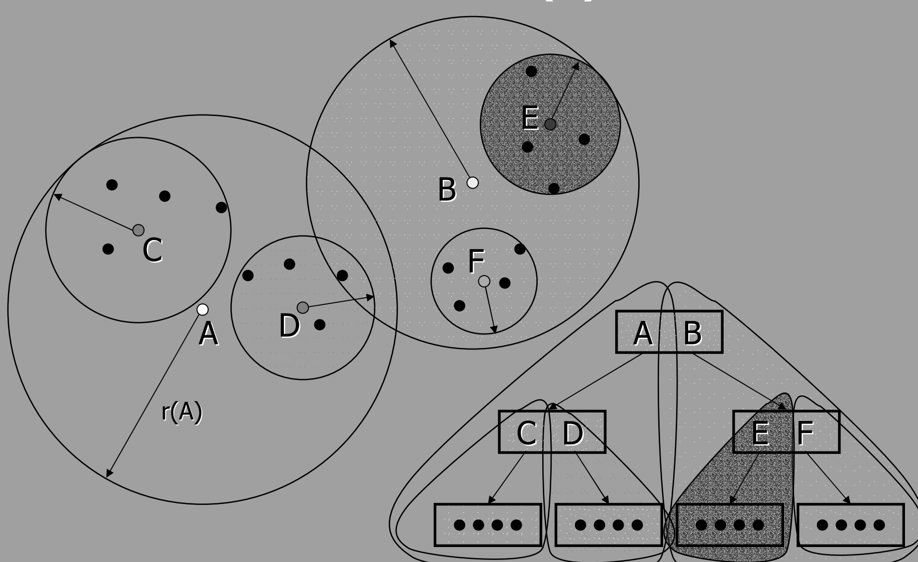
+ optimiert auf Distanzberechnungen und I/O

M-Baum

O_j	Feature von O_j
$oid(O_j)$	Objekt – ID
$d(O_j,P(O_j))$	Distanz zu Vater
$r(O_j)$	Coveringradius
$T(O_j)$	Coveringtree



M-Baum (ii)



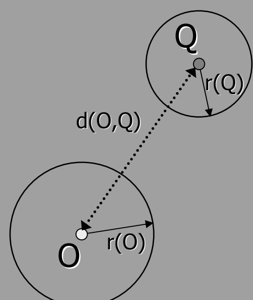
Übersicht

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



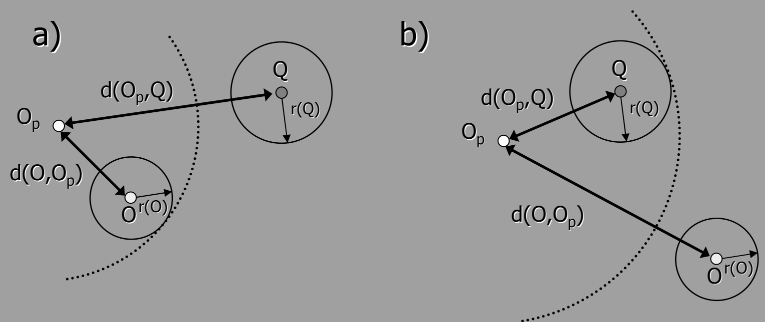
ABER: Berechnung von d(O,Q) nötig!

Ähnlichkeitssuche im M-Baum (i)

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

Lemma 2:

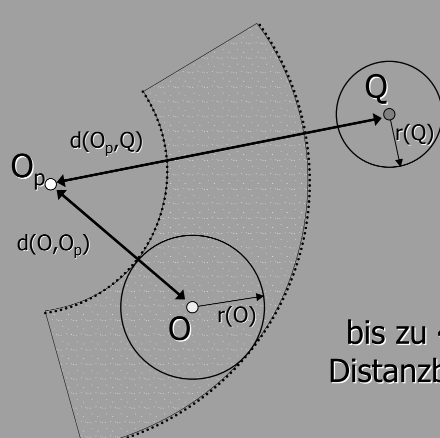
$$|d(O_p,Q)-d(O,O_p)| > r(Q)+r(O) \Rightarrow d(O,Q) > r(Q)+r(O)$$



Ähnlichkeitssuche im M-Baum (ii)

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

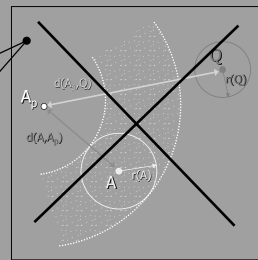
a) + b)

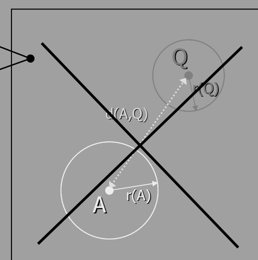


bis zu 40% weniger Distanzberechnungen!

Rangequery

```
RS(N:node,Q:query_object,r(Q):search_radius)
{
    let Op be parent object of node N;
    if N is not a leaf then
    {
        ∀ O in N do:
        if |d(Op,Q)-d(O,Op)|≤r(Q)+r(O) then
        {
            Compute d(O,Q);
            if d(O,Q)≤r(Q)+r(O) then
                RS(*ptr(T(O),Q,r(Q));
        }
    }
    else // N is a leaf
    {
        ∀O in N do:
        if |d(Op,Q)-d(O,Op)|≤r(Q) then
        {
                Compute d(O,Q);
                if d(O,Q)≤r(Q) then
                add oid(O) to result;
        }
    }
}
```





k-NN-Query

zwei globale Strukturen:

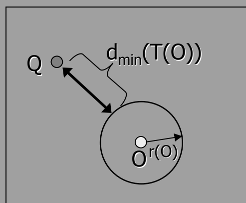
Array NN der Größe k

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

Priority Queue PR

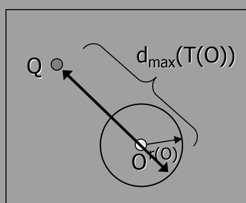
- Zeiger auf „aktive“ Teilbäume, d.h. in denen potentielle Kandidaten enthalten sind
- Struktur: {…, [T(O),d_{min}(T(O))], …}

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)

```
k-NN-Search(T:root_node,Q:query_object,k:integer)
{
    PR := [T,_];
    for i:=1 to k do: NN[i] = [_,∞];
    while PR≠∅ do:
    {
        Next_Node=ChooseNode(PR);
        k-NN_NodeSearch(Next_Node, Q, k);
    }
}
```

```
ChooseNode(PR:priority_queue): node
{
    let d_min(T(O*))=min{d_min(T(O))}, considering all
    the entries in PR;
    remove entry [ptr(T(O*), d_min(T(O*))] from PR;
    return T(O*);
}
```

k-NN-Query (iv)

```
k-NN_NodeSearch(N:node,Q:query_object,k:integer)
{
    let O_p be the parent object of node N;
    if N is not a leaf then {
        ∀O in N do:
        if |d(O_p,Q)-d(O,O_p)|≤d_k+r(O) then {
            Compute d(O,Q);
            if d_min(T(O))≤d_k then {
                add [ptr(T(O)),d_min(T(O))] to PR;
                if d_max(T(O))<d_k then {
                d_k=NN_Update([_,d_max(T(O))]);
                Remove from PR all entries for which d_min(T(O))>d_k;
                }}}}
    else {   /* N is a leaf */
        ∀O in N do:
        if |d(O_p,Q)-d(O,O_p)≤d_k then {
            Compute d(O,Q);
            if d(O,Q)≤d_k then {
                d_k=NN_Update([oid(O),d(O,Q)]);
                Remove from PR all entries for which d_min(T(O))>d_k;
    }}}
}
```

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 absenken
- 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)

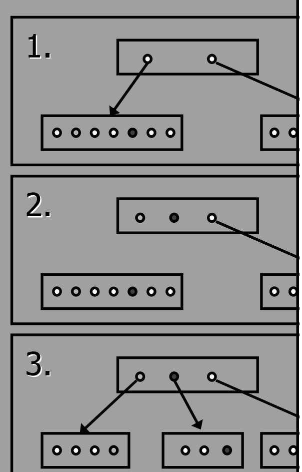
```
Insert(N:node,entry(On):M-tree_entry)
{
    let S be the set of entries in node N;
    if N is not a leaf then {
        let S_in = entries such that d(O,O_n)≤r(O);
        if S_in≠∅ then let entry(O*)∈S_in:d(O*,O_n) is minimum;
        else {
                let entry(O*)∈S:d(O*,O_n)-r(O*) is minimum;
                let r(O*)=d(O*,O_n);
            }
        Insert(T(O*),entry(O_n));
    }
    else {   /* N is a leaf */
        if N is not full
        then store entry(O_n) in N
        else Split(N,entry(O_n));
    }
}
```

Erstellung eines M-Baums (ii)

Splitmanagement:

- Baum wächst „bottom up“
  - Überlauf der Knoten möglich
- geeignete Behandlung:
  1. wähle zwei Routingobjekte O_{p1} O_{p2}
  2. „promote“ diese in den Vaterknoten
  3. 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 indizierzer „toter“ Raum
- minimale Überlappung
  - weniger Teilbäume müssen durchsucht werden

Erstellung eines M-Baums (iv)

Promote Strategien:

m_RAD: komplexeste bzgl. Distanzberechungen. 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

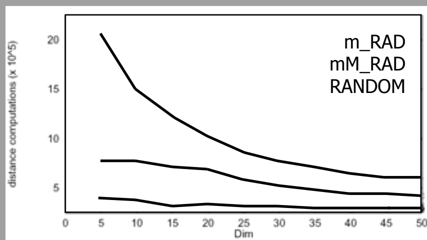
Leistungsevaluation

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

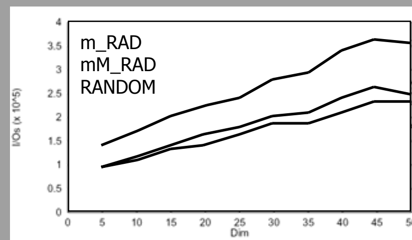
1. steigender Dimensionalität
2. wachsender Anzahl der Objekte

Leistungsevaluation (i)

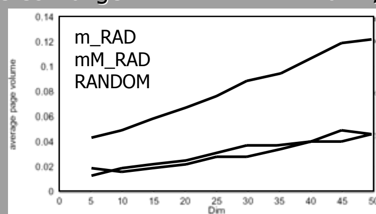
Erstellung eines M-Baums:



Anzahl Distanzberechnungen



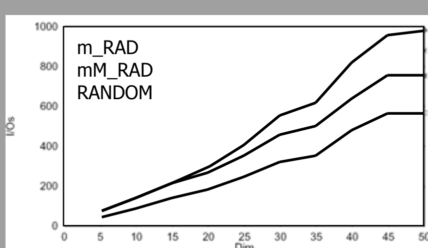
Anzahl I/O-Operationen



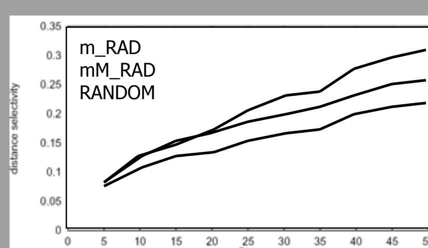
Güte des Baums

Leistungsevaluation (ii)

Ausführen einer 10-NN-Query



Anzahl I/O-Operationen

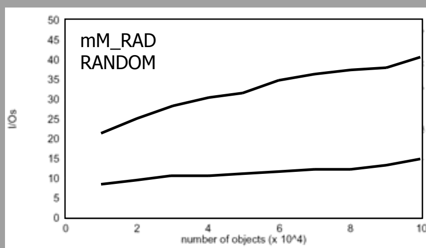


distance-selectivity

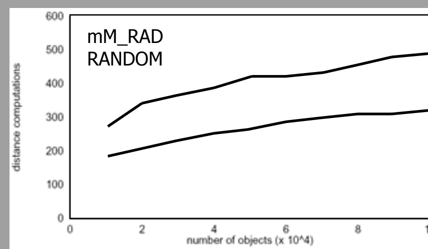
$$\text{distance-selectivity} = \frac{\text{\#benötigter Dist. Berechn.}}{\text{\#Objekte}}$$

Leistungsevaluation (iii)

Ausführen einer 10-NN-Query



Anzahl I/O-Operationen

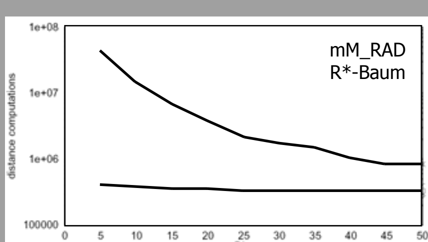


Anzahl Distanzberechnungen

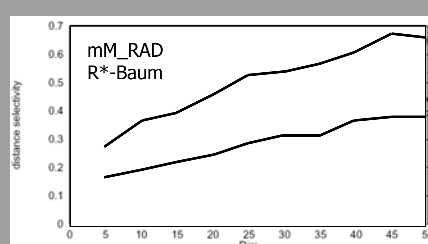
skaliert gut mit wachsender Datenmenge

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

relationale Abbildung

Verwaltung

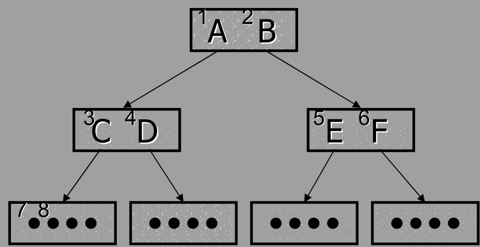
index_create()
index_drop()
index_insert()
index_delete()
index_update()

Anfragebearbeitung

index_open()
index_fetch()
index_close()

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:

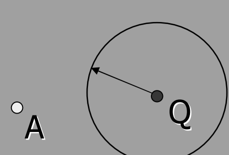
row-id	mt-obj	...
R1	...	...
R2	...	...
...	...	...

Filtering M-tree

Motivation: weitere Reduktion der Distanzberechnungen

Filter:
- Filterdistanz ≤ exakte Distanz
- Filterdistanz effizienter zu berechnen

Beispiel: range query



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

$\Downarrow$

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