
Non-Standard-Datenbanken und Data Mining

First-n-, Top-k- und Skyline-Anfragen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

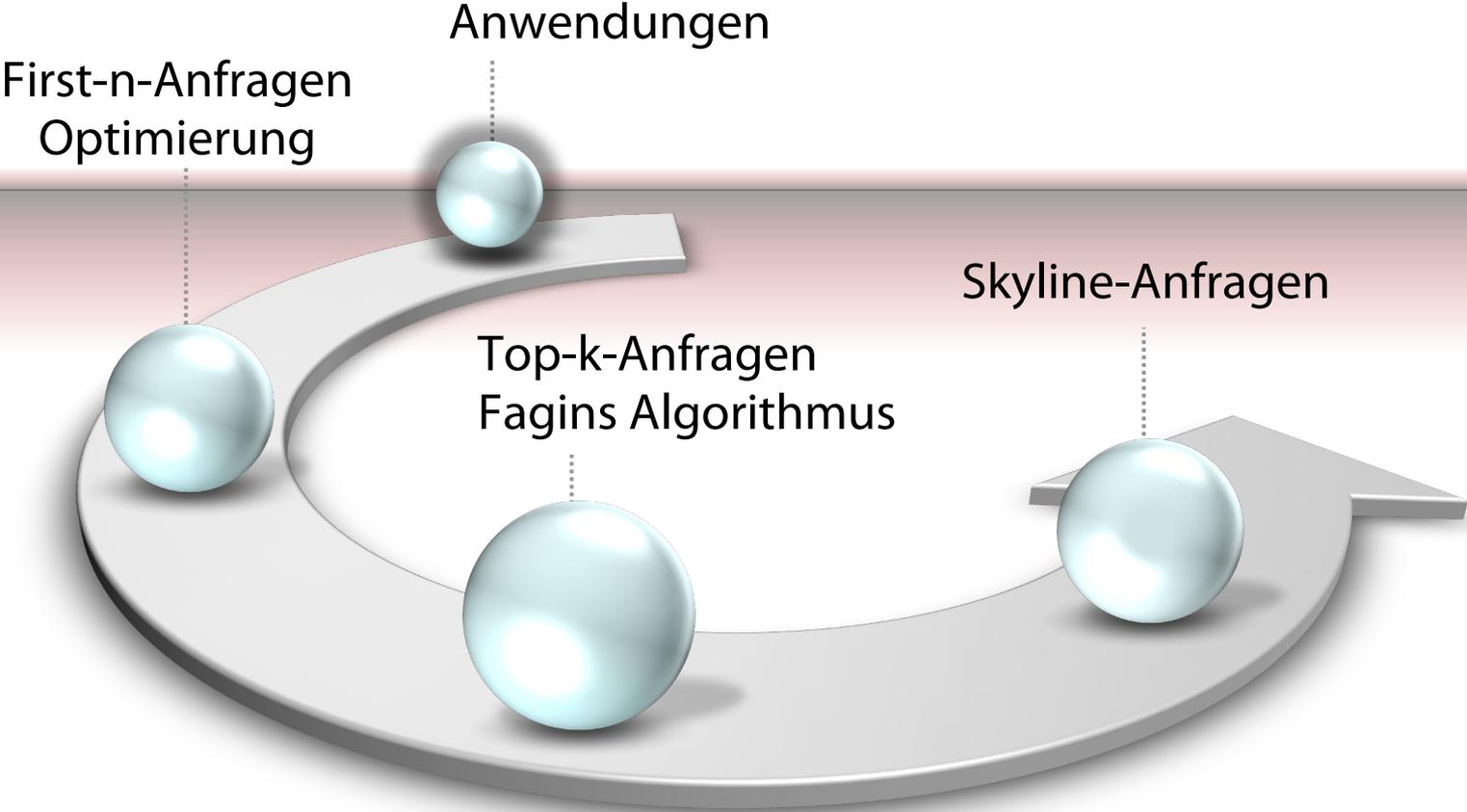


Übersicht

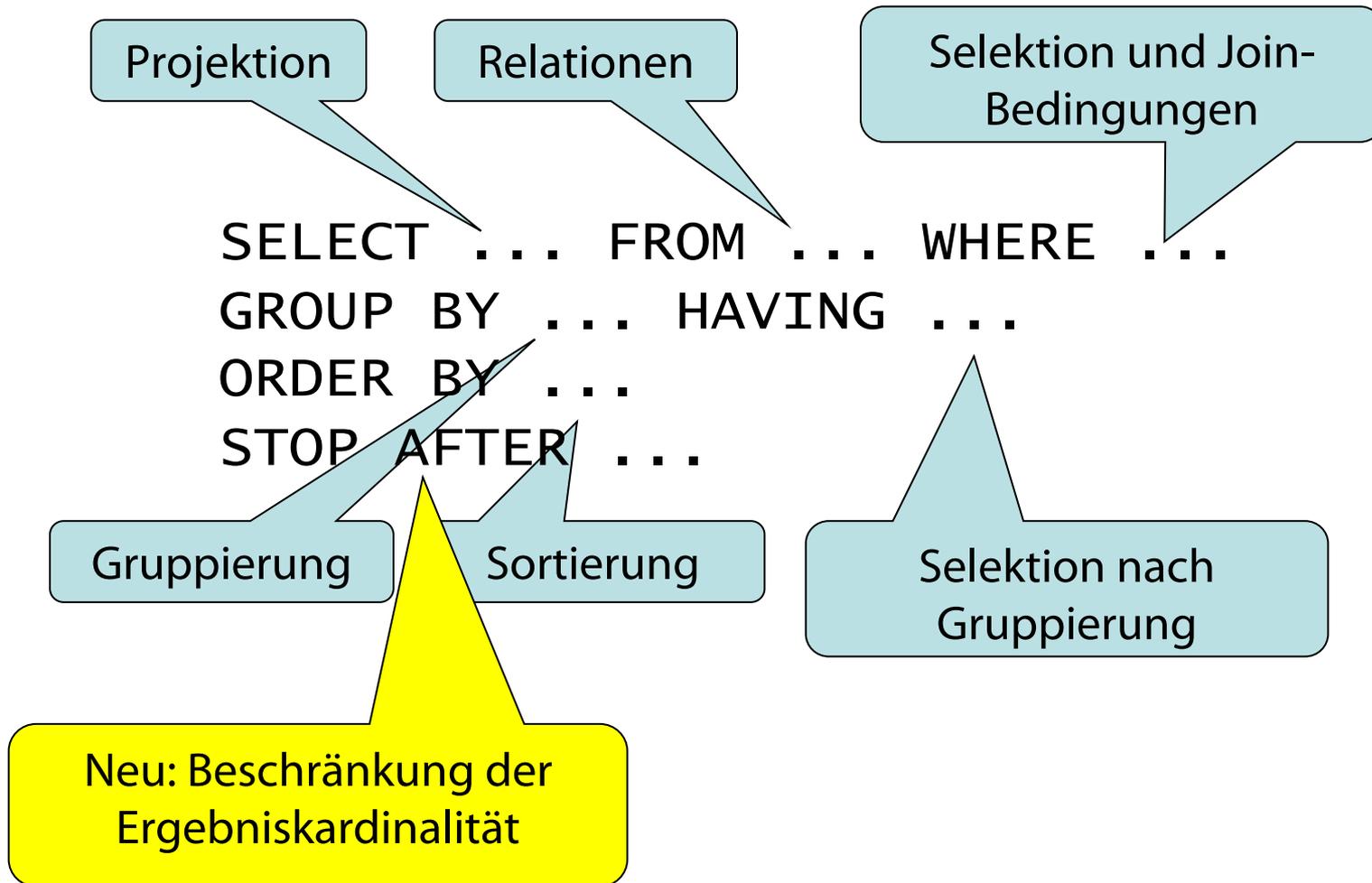
- Semistrukturierte Datenbanken (JSON, XML) und Volltextsuche
- Information Retrieval
- Mehrdimensionale Indexstrukturen
- Cluster-Bildung
- Einbettungstechniken
- **First-n-, Top-k-, und Skyline-Anfragen**
- Probabilistische Datenbanken, Anfragebeantwortung, Top-k-Anfragen und Open-World-Annahme
- Probabilistische Modellierung, Bayes-Netze, Anfragebeantwortungsalgorithmen, Lernverfahren,
- Temporale Datenbanken und das relationale Modell,
- Probabilistische Temporale Datenbanken
- SQL: neue Entwicklungen (z.B. JSON-Strukturen und Arrays), Zeitreihen (z.B. TimeScaleDB)
- Stromdatenbanken, Prinzipien der Fenster-orientierten inkrementellen Verarbeitung
- Approximationstechniken für Stromdatenverarbeitung, Stream-Mining
- Probabilistische raum-zeitliche Datenbanken und Stromdatenverarbeitungssysteme: Anfragen und Indexstrukturen, Raum-zeitliches Data Mining, Probabilistische Skylines
- Von NoSQL- zu NewSQL-Datenbanken, CAP-Theorem, Blockchain-Datenbanken

Non-Standard-Datenbanken und Data Mining

First-n und Top-k-Anfragen



STOP AFTER – Syntax



M.J. Carey, Donald Kossmann: On Saying "Enough Already!"
in SQL. SIGMOD Conference: 219-230, 1997

STOP AFTER – Semantik

- Ohne Sortierung
 - Willkürlich n Tupel aus dem SQL-Ergebnis
- Mit Sortierung
 - Erste n Tupel aus dem SQL-Ergebnis (sortiert)
 - Bei Gleichheit bzgl. des Sortierprädikats willkürliche Auswahl falls $n+1$ -tes Tupel usw. gleich n -tem Tupel
- Daraus folgt:
Falls Ergebnis nur bis zu n Tupel → keine Wirkung

STOP AFTER – Beispiel

```
SELECT h.name, h.adresse, h.tel  
FROM   hotels h, flughäfen f  
WHERE  f.name = 'LBC'  
ORDER BY distance(h.ort, f.ort)  
STOP AFTER 5
```

- Ergebnis: 5 Hotels mit aufsteigender Entfernung zu LBC
- Können wir Indexstrukturen für distance() nutzen?

STOP AFTER: Berechnete Anzahl

- Liste Name und Umsatz der 10% umsatzstärksten Softwareprodukte

```
SELECT p.name, v.umsatz
FROM   Produkte p, Verkäufe v
WHERE  p.typ = 'software'
AND    p.id = v.prod_id
ORDER BY v.umsatz DESC
STOP AFTER (
        SELECT count(*) / 10
        FROM   Produkte p, Verkäufe v
        WHERE  p.typ = 'software'
        AND    p.id = v.prod_id
    )
```

Praktische Ausprägungen

PostgreSQL:

```
SELECT ...  
FROM ...  
...  
LIMIT { count | ALL }  
OFFSET start
```

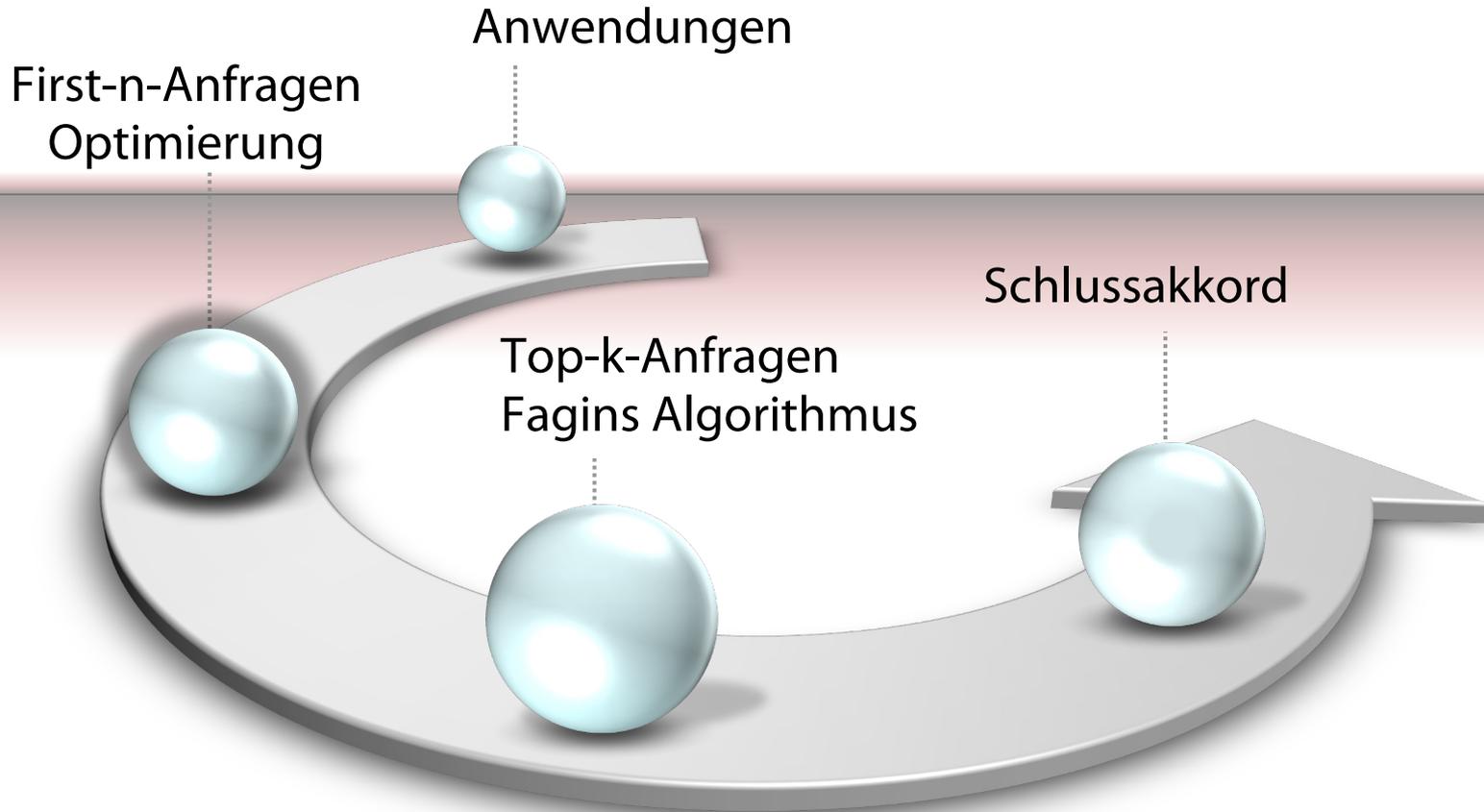
- Verwendung in Kombination mit ORDER BY möglich und sinnvoll
- Falls der Ausdruck **count** sich zu NULL evaluiert, wird ALL angenommen.
- Falls der Ausdruck **start** sich zu NULL evaluiert, wird 0 angenommen.

SQL Server:

```
SELECT TOP n WITH TIES FROM tablename
```

Non-Standard-Datenbanken und Data Mining

First-n und Top-k-Anfragen



Danksagung an Felix Naumann für Material
aus VL Informationsintegration, WS 05/06

Optimierung mit Stop-Operator

- Platzierung des Stop Operators im Anfrageplan
- Fundamentales Problem: Frühe Platzierung vorteilhaft aber risikoreich
 - Vorteil: Kleine Zwischenergebnisse \Rightarrow geringe Kosten
 - Risiko: Endergebnis nicht groß genug \Rightarrow Erneute Ausführung
- Zwei Strategien
 - „Konservativ“ und „aggressiv“

Optimierung mit Stop-Operator

- Konservative Strategie
 - Kostenminimal:
Platziere Stop so früh wie möglich in Plan.
 - Korrekt: Platziere Stop nie so, dass Tupel entfernt werden, die später eventuell gebraucht werden.
 - Operatoren, die Tupel filtern, müssen also früher ausgeführt werden.

STOP AFTER: Optimierung

```
SELECT *  
FROM  mitarbeiter m, abteilung a  
WHERE m.abt_id = a.id  
ORDER BY m.gehalt DESC  
STOP AFTER 10
```

- Wie würden Sie den Anfragebeantwortungsplan gestalten?
- Unter welchen Bedingungen ist eine Optimierung möglich?

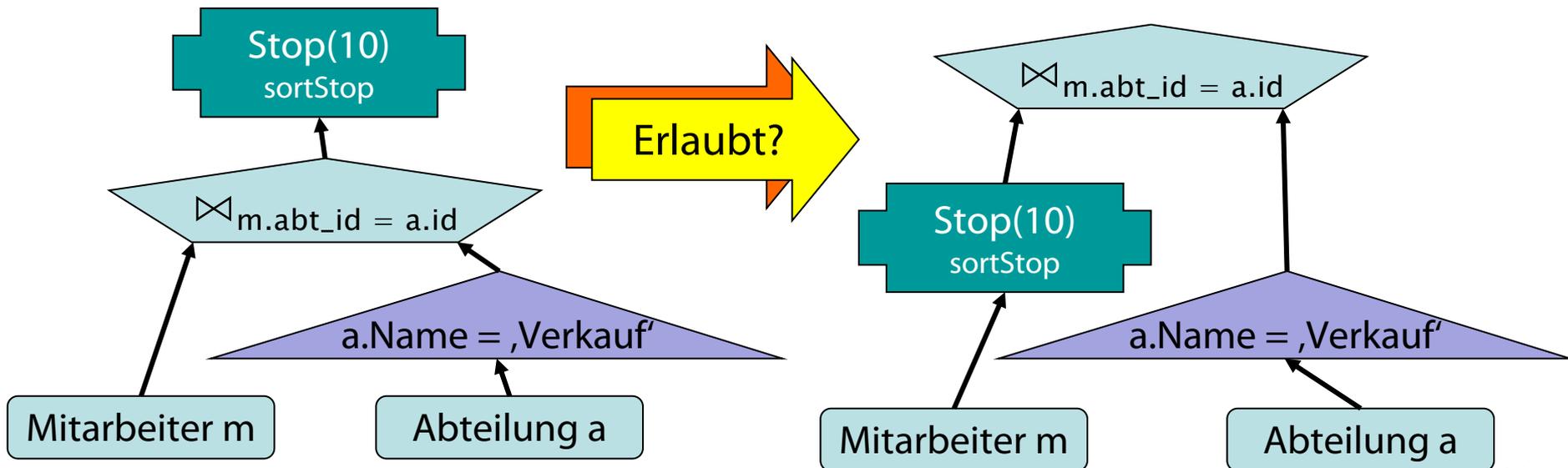
Optimierung mit Stop-Operator

```
SELECT *  
FROM  mitarbeiter m, abteilung a  
WHERE m.abt_id = a.id  
ORDER BY m.gehalt DESC  
STOP AFTER 10
```



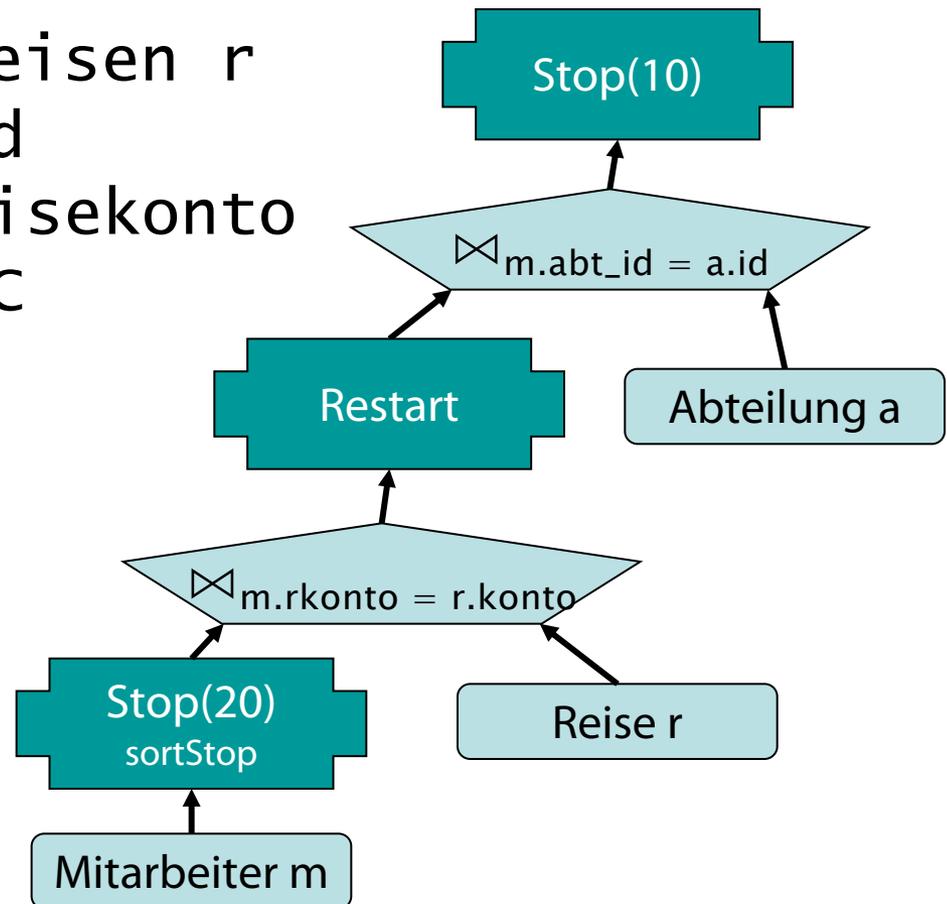
Optimierung mit Stop-Operator

```
SELECT *  
FROM  mitarbeiter m, abteilung a  
WHERE m.abt_id = a.id  
AND   a.name = ,Verkauf'  
ORDER BY m.gehalt DESC  
STOP AFTER 10
```



Optimierung mit Stop-Operator (aggressiv)

```
SELECT *  
FROM   mitarbeiter m,  
       abteilung a, reisen r  
WHERE  m.abt_id = a.id  
AND    r.konto = m.reisekonto  
ORDER BY m.gehalt DESC  
STOP AFTER 10
```



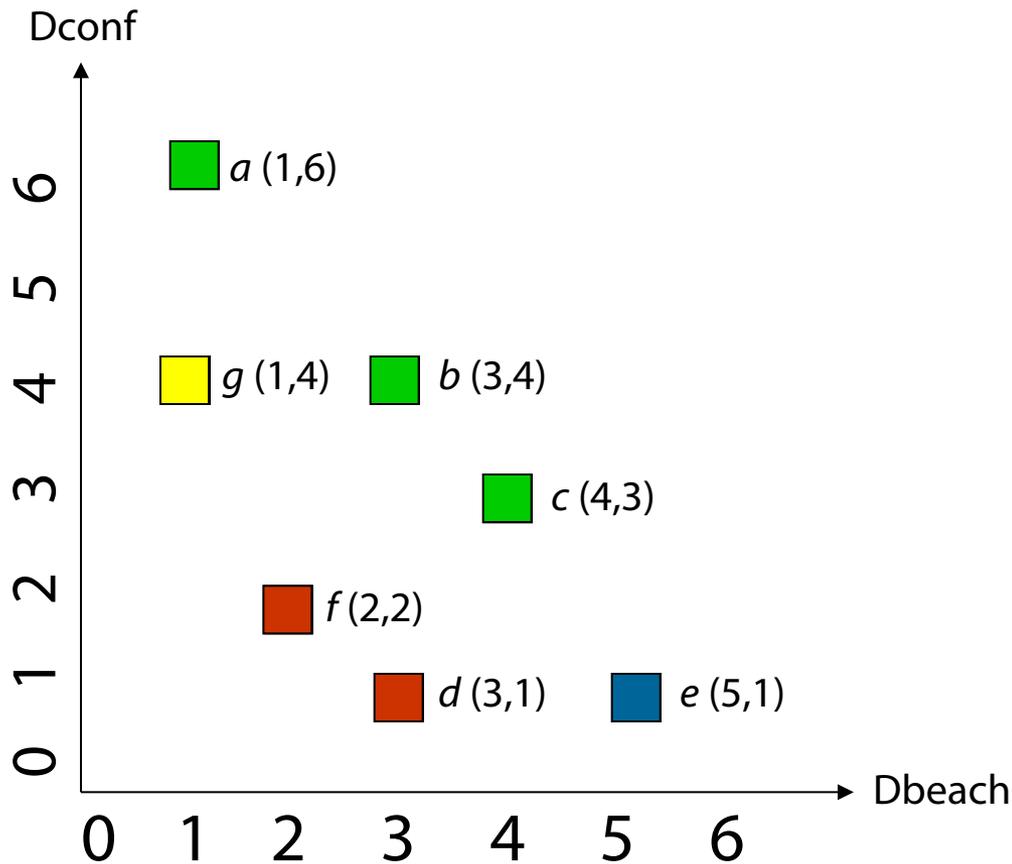
Top-k-Anfragen

- Gegeben eine Menge von Tupeln jeweils mit einer Bewertung hergeleitet aus ggf. mehreren Attributen
- Beispiele für Bewertungsfunktionen:
 - Min, Max, Linearkombination
- Beispiel: Hotels für Dienstreise
 - Distanz Strand
 - Distanz Conference Center

id	Dbeach	Dconf	Score
<i>a</i>	1	6	7
<i>b</i>	3	4	7
<i>c</i>	4	3	7
<i>d</i>	3	1	4
<i>e</i>	5	1	6
<i>f</i>	2	2	4
<i>g</i>	1	4	5

Ziel: $D_{\text{beach}}(x) + D_{\text{conf}}(x)$ minimieren

Bewertung basierend auf $D_{\text{beach}}(x) + D_{\text{conf}}(x)$



id	Dbeach	Dconf	Score
<i>a</i>	1	6	7
<i>b</i>	3	4	7
<i>c</i>	4	3	7
<i>d</i>	3	1	4
<i>e</i>	5	1	6
<i>f</i>	2	2	4
<i>g</i>	1	4	5

Beste Hotels: *d, f*

Zweitbeste: *g*

Drittbeste: *e*

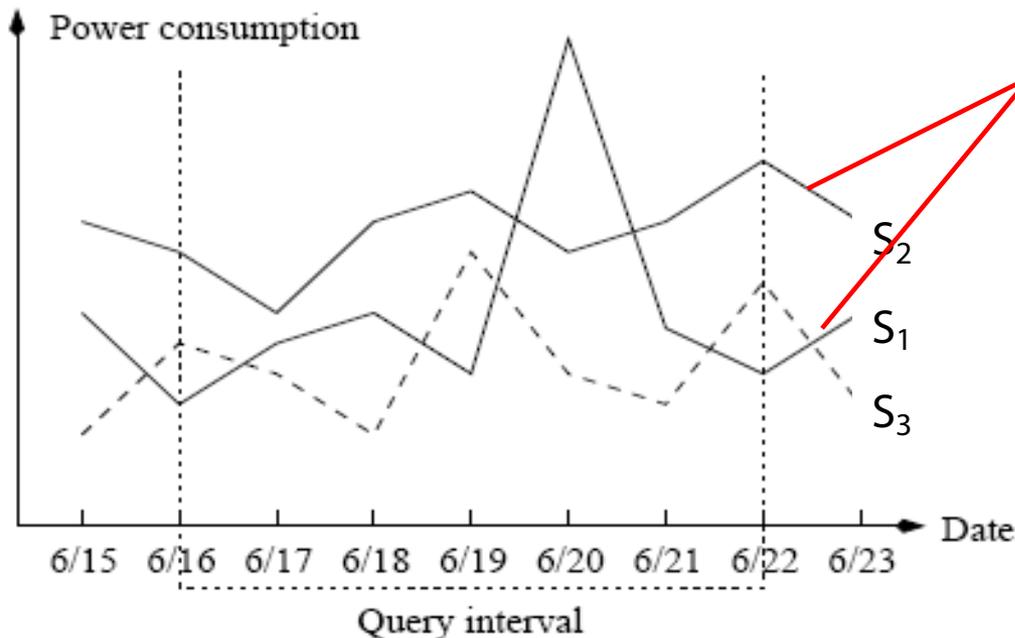
Nächstbeste: *a, b, c*

Alternative: Skyline-Anfragen



Intervall-Skyline

- Gegeben eine Menge S von Zeitreihen und ein Intervall $[i: j]$. Eine **Intervall-Skyline** ist eine Zeitreihe $s \in S$, die nicht von anderen Zeitreihen aus S dominiert wird.
- Wir schreiben: $Sky[i: j] = \{s \in S \mid \nexists s' \in S, s' \succ_{[i:j]} s\}$



Bsp: $S = \{S_1, S_2, S_3\}$

S_1 und S_2 sind in $Sky[16:22]$, während S_3 durch S_2 dominiert wird.

Wird im Rahmen von Stromdatenbanken später behandelt

Agenda

- Top-k-Anfragen
 - Fagins Algorithmus (FA)
 - Threshold-Algorithmus (TA), No Random Access Algorithmus (NRA) und Kombinationen davon (CA)
- Skyline-Anfragen
 - Nested-Block-Loop, Teile-und-Herrsche, Nächste Nachbarn
 - Branch-and-Bound-Skyline-Algorithmus (Verwendung von R-Bäumen)

Top-k-Berechnung

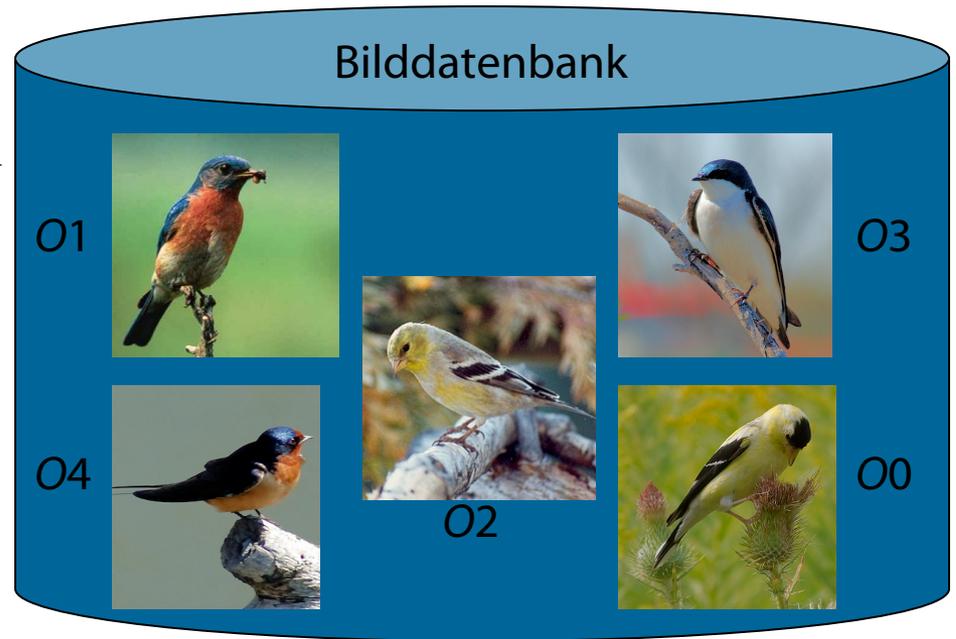
Nehmen wir an, die Datenbasis enthält 5 Bildobjekte
 O_0 , O_1 , O_2 , O_3 and O_4

Anfragebild Q



?

Top-2
Bilder



Top- k -Berechnung

Datenbasis entspricht $n \times m$ Bewertungsmatrix für die Bewertungen eines jeden Objekts bzgl. aller Attribute

a1	a2	a3	a4	a5
$O_3, 99$	$O_1, 91$	$O_1, 92$	$O_3, 74$	$O_3, 67$
$O_1, 66$	$O_3, 90$	$O_3, 75$	$O_1, 56$	$O_4, 67$
$O_0, 63$	$O_0, 61$	$O_4, 70$	$O_0, 56$	$O_1, 58$
$O_2, 48$	$O_4, 07$	$O_2, 16$	$O_2, 28$	$O_2, 54$
$O_4, 44$	$O_2, 01$	$O_0, 01$	$O_4, 19$	$O_0, 35$

Für jedes Attribut sind die Bewertungen absteigend sortiert

Top- k -Berechnung – Algorithmus FA

Schritt 1:

- Lese Tupel für jede sortierte Liste (**sortierter Zugriff**)
- Halte, wenn k Objekte mit allen Attributwerten bekannt

Schritt 2:

- Generiere Tabellenzugriffe, um fehlende Werte zu erhalten (**zufälliger Zugriff**)

Schritt 3:

- Berechne die Bewertungen der gesehenen Objekte
- Gebe k höchstbewertete Objekte zurück

Top-k-Berechnung – Algorithmus FA

Schritt 1:

Lese Tupel für jede sortierte Liste (sortierter Zugriff)

Halte, wenn k Objekte mit allen Attributwerten bekannt

a1	a2	a3	a4	a5
O3, 99	O1, 91	O1, 92	O3, 74	O3, 67
O1, 66	O3, 90	O3, 75	O1, 56	O4, 67
O0, 63	O0, 61	O4, 70	O0, 56	O1, 58
O2, 48	O4, 07	O2, 16	O2, 28	O2, 54
O4, 44	O2, 01	O0, 01	O4, 19	O0, 35

id	a1	a2	a3	a4	a5
O3	99	90	75	74	67
O1	66	91	92	56	58
O4			70		67
O0	63	61		56	

Kein weiterer sortierter Zugriff nötig, denn es sind k=2 Objekte mit allen Attributwerten bekannt (Objekte O1 and O3).

Top-k-Berechnung – Algorithmus FA

Schritt 2:

- Generiere Tabellenzugriffe, um fehlende Werte zu erhalten
(zufälliger Zugriff)

a1	a2	a3	a4	a5
O3, 99	O1, 91	O1, 92	O3, 74	O3, 67
O1, 66	O3, 90	O3, 75	O1, 56	O4, 67
O0, 63	O0, 61	O4, 70	O0, 56	O1, 58
O2, 48	O4, 07	O2, 16	O2, 28	O2, 54
O4, 44	O2, 01	O0, 01	O4, 19	O0, 35

id	a1	a2	a3	a4	a5
O3	99	90	75	74	67
O1	66	91	92	56	58
O4	44	07	70	19	67
O0	63	61	01	56	35

Alle fehlenden Werte für gesehene Objekte sind bestimmt. Keine weiteren Zugriffe nötig.

Top-k-Berechnung – Algorithmus FA

Schritt 3:

- Berechne die Bewertungen der gesehenen Objekte
- Gebe k höchstbewertete Objekte zurück

id	a1	a2	a3	a4	a5	
O3	99	90	75	74	67	405
O1	66	91	92	56	58	363
O4	44	07	70	19	67	207
O0	63	61	01	56	35	216

Totale Bewertung

405 } Top-2
363 }
207
216

Daher sind die besten Objekte für die Anfrage:
O3 mit Bewertung 405 and O1 mit Bewertung 363.

Die besten sind nicht notwendigerweise unter den ersten k.

Top-k: Fagins Algorithmus

- **Gegeben:** Tabelle mit Objekten und Bewertungsfunktion $g_A(x)$ mit $A = A_1 \wedge A_2 \wedge \dots \wedge A_m$
- **Behauptung:**
 - Fagins Algorithmus findet die Top-k Objekte gemäß $g_A(x)$.
- **Beweis:**
 - Idee: Wir zeigen für jedes ungesehene Objekt y , dass es nicht unter den Top-k sein kann:
 - Notation x : gesehene Objekte, y : ungesehene Objekte
 - Wir haben absteigend sortiert: Für jedes x der Joinmenge nach Phase 1 und jedes Prädikat A_i gilt: $g_{A_i}(y) \leq g_{A_i}(x)$, wenn $g_{A_i}(x)$ schon definiert (und das ist ja schon für k Objekte der Fall: Abbruch-Kriterium Phase 1)
 - Phase 3: $g_{A_1 \wedge A_2 \wedge \dots \wedge A_m}(x) = \text{reduce}(+, \{g_{A_1}(x), g_{A_2}(x), \dots, g_{A_m}(x)\}, 0)$
 - Da **jedes** Attribut der vollständig gesehenen Objekte z besser, gilt:
$$g_{A_1 \wedge A_2 \wedge \dots \wedge A_m}(y) \leq g_{A_1 \wedge A_2 \wedge \dots \wedge A_m}(z)$$
 - Es gibt kein y , das besser ist als die vollst. gesehenen Objekte z (vielleicht sind ja die in Phase 2 vervollständigten Objekte z noch besser als die k vollständig nach Phase 1 gesehenen Objekte x , aber die z werden ja in Phase 3 auch überprüft)

Top-k: Fagins Algorithmus

- Aufwand: $O(n^{(m-1)/m}k^{1/m})$ (Beweis: siehe [Fa96])
 - n = DB-Größe; m = Anzahl der Konjunkte (Attribute/DBs)
 - Beispiel: 10000 Objekte, 3 Konjunkte, Top 10
 - $10.000^{2/3} \times 10^{1/3} = 1.000$
 - Gilt falls A_i -Werte unabhängig.
- Zum Vergleich: Naiver Algorithmus in $O(nm)$
 - Im Beispiel: $10.000 \times 3 = 30.000$
- Weiterentwicklung: **Threshold Algorithmus** (TA),
No Random Access Algorithmus (NRA),
Combined Algorithm (CA)

Top- k -Berechnung – Algorithmus TA

FA benötigt relativ viel Pufferspeicher, daher wurden Verbesserung vorgeschlagen, z.B. den Algorithmus TA (Threshold Algorithm)

Hauptidee:

Einführung eines **Schwellwerts**, um zu bestimmen, wann der Zugriff auf die sortierten Werte beendet werden kann

Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '01). ACM, New York, NY, USA, 102-113., **2001**

Top- k -Berechnung – Algorithmus TA

Überblick über TA

Schritt 1:

- Lese Attributwerte aus jeder sortierten Liste (**sortierter Zugriff**)
- Für jedes gesehene Objekt x :
 - Verwende **zufälligen Zugriff**, um fehlende Werte zu bestimmen
 - Bestimme die Bewertung $F(x)$ von Objekt x .
 - Falls Objekt unter den ersten top- k , behalte es im Puffer

Schritt 2:

- Bestimme Schwellwert T basierend auf mit sortiertem Zugriff schon gesehener Objekte
- $T = a_1(p) + a_2(p) + \dots + a_m(p)$ wobei p die aktuelle Position des sortierten Zugriffs ist
- Falls es k Objekte mit Gesamtbewertung $\geq T$ gibt
dann HALTE und bestimme Ergebnis
sonst $p := p + 1$ und GOTO Schritt 1

Top-k-Berechnung – Algorithmus TA

Schritt 1:

- Lese Attribute aus jeder sortierten Liste (**sortierter Zugriff**)
- Für jedes gesehene Objekt x:
 - Verwende **zufälligen Zugriff**, um fehlende Werte zu bestimmen
 - Bestimme die Bewertung $F(x)$ von Objekt x.
 - Falls Objekt unter den ersten top-k, behalte es im Puffer

BUFFER:
(O3, 405)
(O1, 363)

$p=1$ →

a1	a2	a3	a4	a5
O3, 99	O1, 91	O1, 92	O3, 74	O3, 67
O1, 66	O3, 90	O3, 75	O1, 56	O4, 67
O0, 63	O0, 61	O4, 70	O0, 56	O1, 58
O2, 48	O4, 07	O2, 16	O2, 28	O2, 54
O4, 44	O2, 01	O0, 01	O4, 19	O0, 35

id	a1	a2	a3	a4	a5	F
O3	99	90	75	74	67	405
O1	66	91	92	56	58	363

Top-k-Berechnung – Algorithmus TA

Schritt 2:

- Bestimme Schwellwert T basierend auf mit sortiertem Zugriff schon gesehene Objekte
- $T = a_1(p) + a_2(p) + \dots + a_m(p)$ wobei p die aktuelle Position des sortierten Zugriffs ist
- Falls es k Objekte mit Gesamtbewertung $\geq T$ gibt dann HALTE und bestimme Ergebnis
- sonst $p := p + 1$ und GOTO Schritt 1

BUFFER:
(O3, 405)
(O1, 363)

$p=1$ →

a1	a2	a3	a4	a5
O3, 99	O1, 91	O1, 92	O3, 74	O3, 67
O1, 66	O3, 90	O3, 75	O1, 56	O4, 67
O0, 63	O0, 61	O4, 70	O0, 56	O1, 58
O2, 48	O4, 07	O2, 16	O2, 28	O2, 54
O4, 44	O2, 01	O0, 01	O4, 19	O0, 35

id	a1	a2	a3	a4	a5	F
O3	99	90	75	74	67	405
O1	66	91	92	56	58	363

$$T = 99 + 91 + 92 + 74 + 67 = 423$$

Es gibt keine k Objekte mit Bewertung $\geq T$, GOTO Schritt 1...

Top-k-Berechnung – TA algorithm

Schritt 1: (zweite Ausführung)

- Lese Attribute aus jeder sortierten Liste (**sortierter Zugriff**)
- Für jedes gesehene Objekt x:
 - Verwende **zufälligen Zugriff**, um fehlende Werte zu bestimmen
 - Bestimme die Bewertung $F(x)$ von Objekt x.
 - Falls Objekt unter den ersten top-k, behalte es im Puffer

BUFFER:
(O3, 405)
(O1, 363)

$p=2$ →

a1	a2	a3	a4	a5
O3, 99	O1, 91	O1, 92	O3, 74	O3, 67
O1, 66	O3, 90	O3, 75	O1, 56	O4, 67
O0, 63	O0, 61	O4, 70	O0, 56	O1, 58
O2, 48	O4, 07	O2, 16	O2, 28	O2, 54
O4, 44	O2, 01	O0, 01	O4, 19	O0, 35

id	a1	a2	a3	a4	a5	F
O3	99	90	75	74	67	405
O1	66	91	92	56	58	363
O4	44	07	70	19	67	207

Top-k-Berechnung – TA algorithm

Schritt 2: (zweite Ausführung)

- Bestimme Schwellwert T basierend auf mit sortiertem Zugriff schon gesehene Objekte
- $T = a_1(p) + a_2(p) + \dots + a_m(p)$ wobei p die aktuelle Position des sortierten Zugriffs ist
- Falls es k Objekte mit Gesamtbewertung $\geq T$ gibt dann HALTE und bestimme Ergebnis
- sonst $p := p + 1$ und GOTO Schritt 1

BUFFER:
(O3, 405)
(O1, 363)

	a1	a2	a3	a4	a5
$p=2$ →	O3, 99	O1, 91	O1, 92	O3, 74	O3, 67
	O1, 66	O3, 90	O3, 75	O1, 56	O4, 67
	O0, 63	O0, 61	O4, 70	O0, 56	O1, 58
	O2, 48	O4, 07	O2, 16	O2, 28	O2, 54
	O4, 44	O2, 01	O0, 01	O4, 19	O0, 35

id	a1	a2	a3	a4	a5	F
O3	99	90	75	74	67	405
O1	66	91	92	56	58	363
O4	44	07	70	19	67	207

$$T = 66 + 90 + 75 + 56 + 67 = 354$$

Beide Objekte im Puffer haben Bewertung $> T$. STOP und erzeuge Antwort

Top- k -Berechnung - FA vs. TA

- TA betrachtet i.a. weniger Objekte als FA
 - TA hält mindestens so früh wie FA
 - Wenn wir k Objekte in FA sehen, ist ihre Bewertung größer oder gleich dem TA-Schwellwert
- TA **könnte** mehr zusätzliche zufällige Zugriffe erzeugen als FA
 - In TA, $(m-1)$ zufällige Zugriffe pro Objekt
 - In FA, zufällige Zugriffe am Ende, nur für fehlende Werte
- TA benötigt nur **begrenzten Pufferspeicher** (k) bei möglicherweise mehr zufälligen Zugriffen
- FA verwendet unbegrenzten Pufferspeicher (ggf. alle Tupel)

Top-k-Berechnung – Andere Methoden

Fagin et al. haben weitere relevante Varianten vorgeschlagen:

- Algorithmus **NRA** (**N**o **R**andom **A**ccess): verwendet nur sortierten Zugriff, **keinen zufälligen** Zugriff
- Algorithmus **CA** (**C**ombined **A**lgorithm): **Kombination** von TA und NRA für bessere Performanz

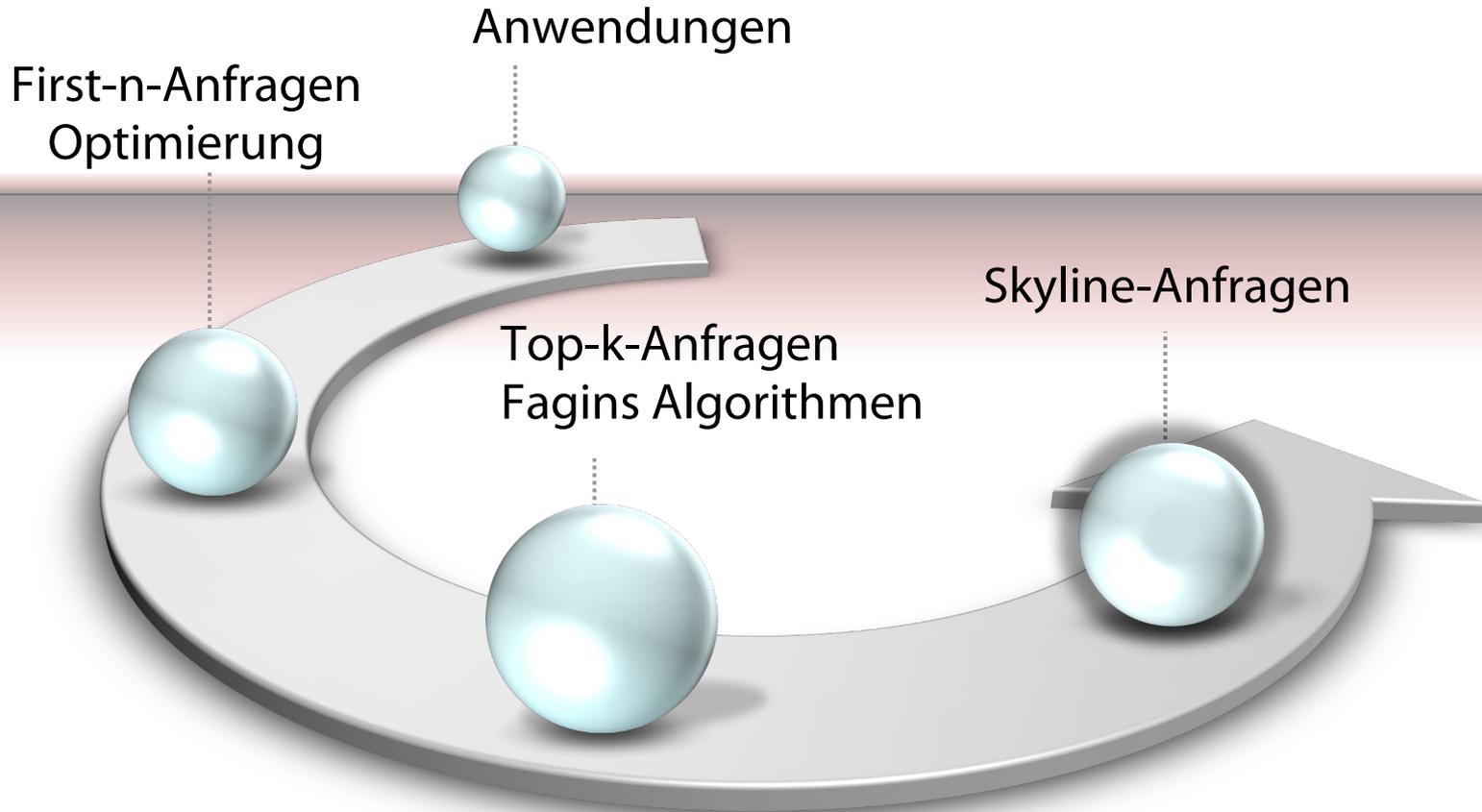
Weitere Entwicklungen:

- Verteilte Top-k-Berechnung
- Top-k-Berechnung mit Joins für Bewertung
- Top-k mit probabilistischen Daten (kommt später)
- Vermeidung der Angabe von k und der Aggregation der Attributwerte

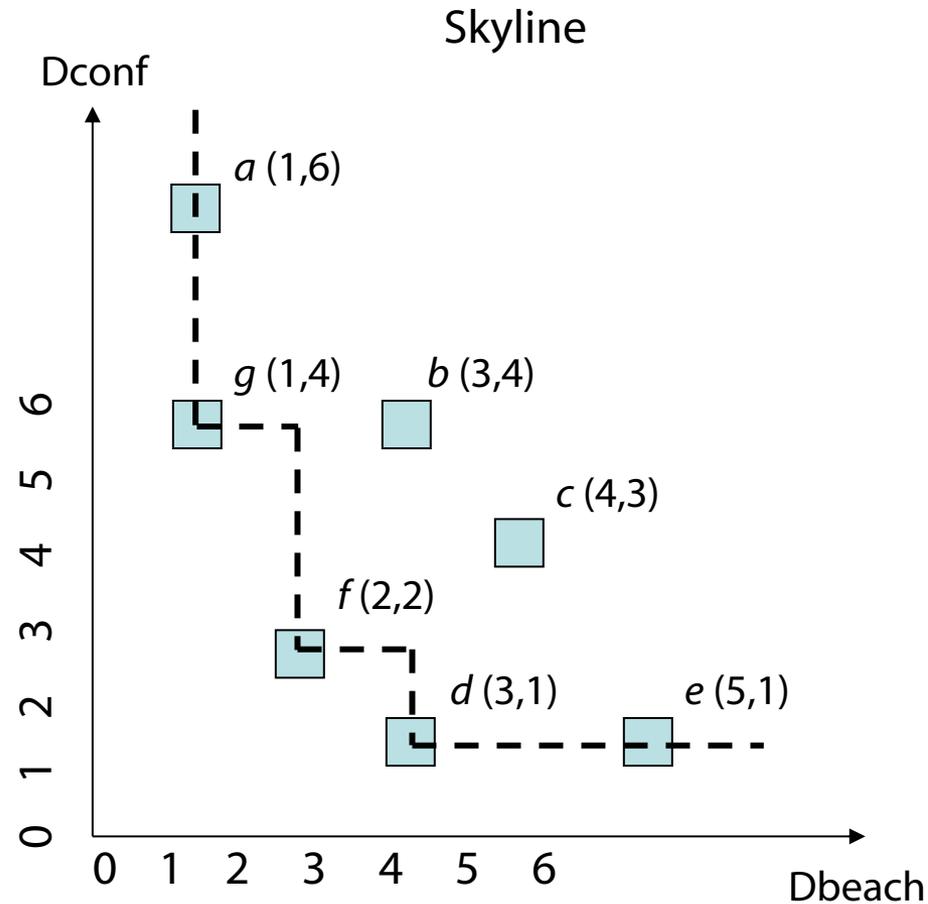
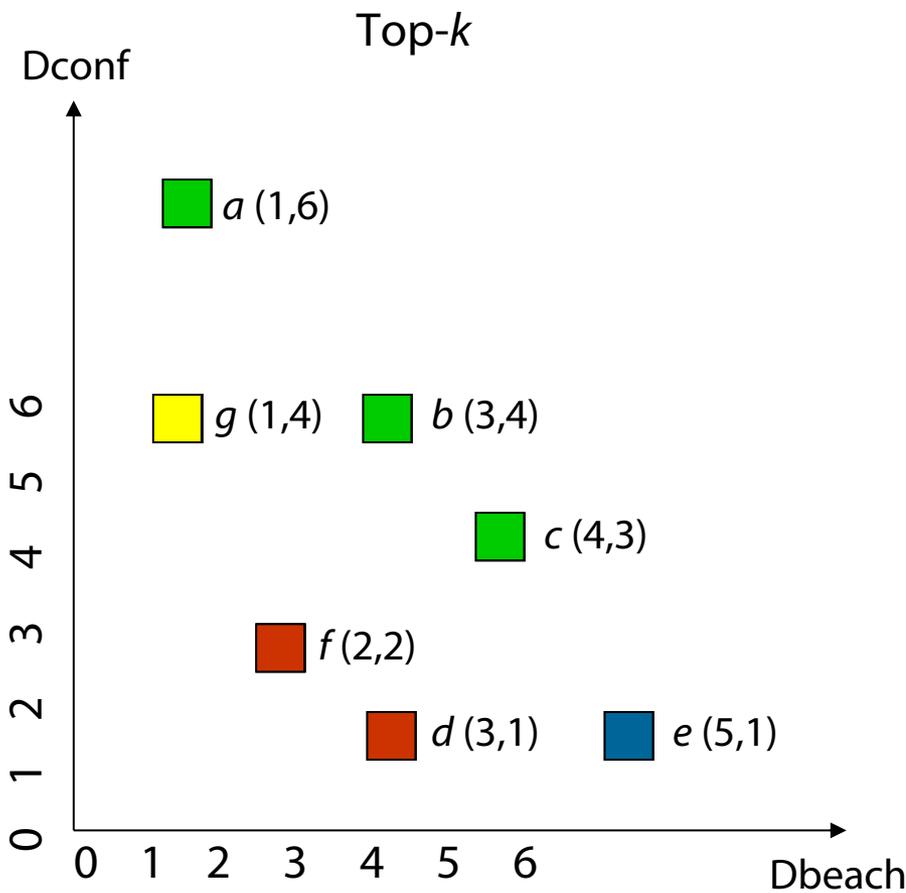
Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '01). ACM, New York, NY, USA, 102-113., **2001**

Non-Standard-Datenbanken und Data Mining

Von First-n- und Top-k-Anfragen zu Skyline-Anfrage



Skyline-Berechnung



■ *f, d* (best objects)

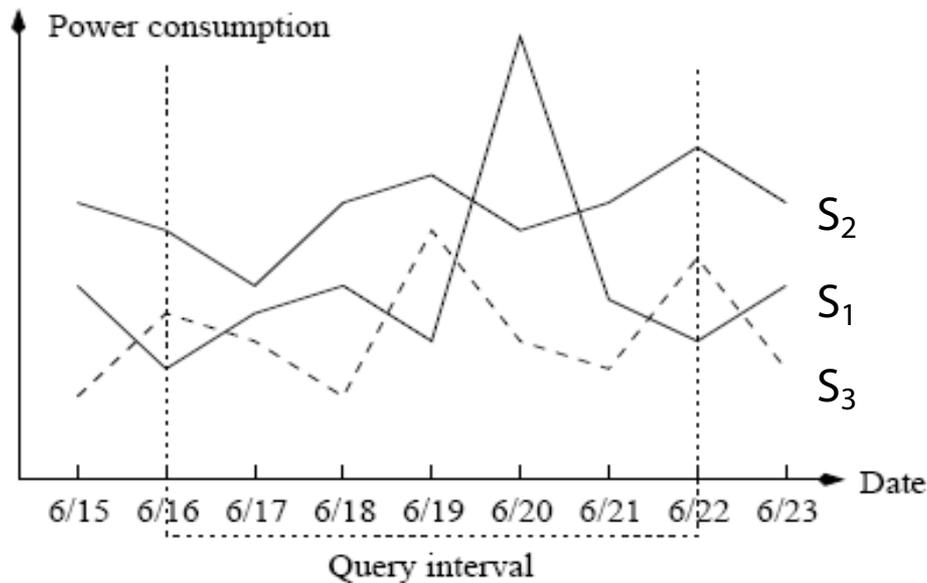
■ *g* (next best)

■ *e* (next best)

Skyline objects: *g, f, d*

Skyline (auch: Intervall-Skyline)

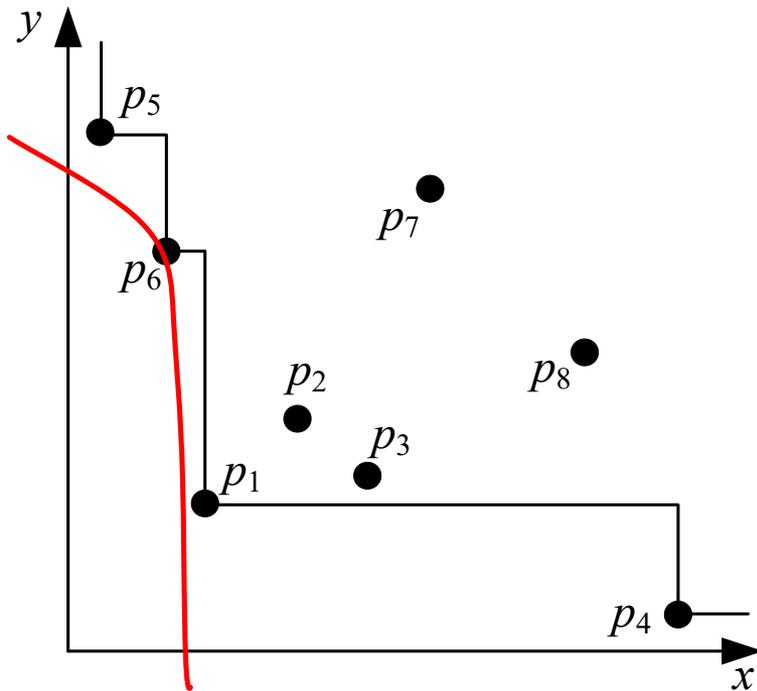
- Gegeben eine Menge S von Zeitreihen und ein Intervall $[i: j]$. Eine **Intervall-Skyline** ist eine Zeitreihe $s \in S$, die nicht von anderen Zeitreihen aus S dominiert wird.
- Wir schreiben: $Sky[i: j] = \{s \in S \mid \nexists s' \in S, s' \succ_{[i:j]} s\}$



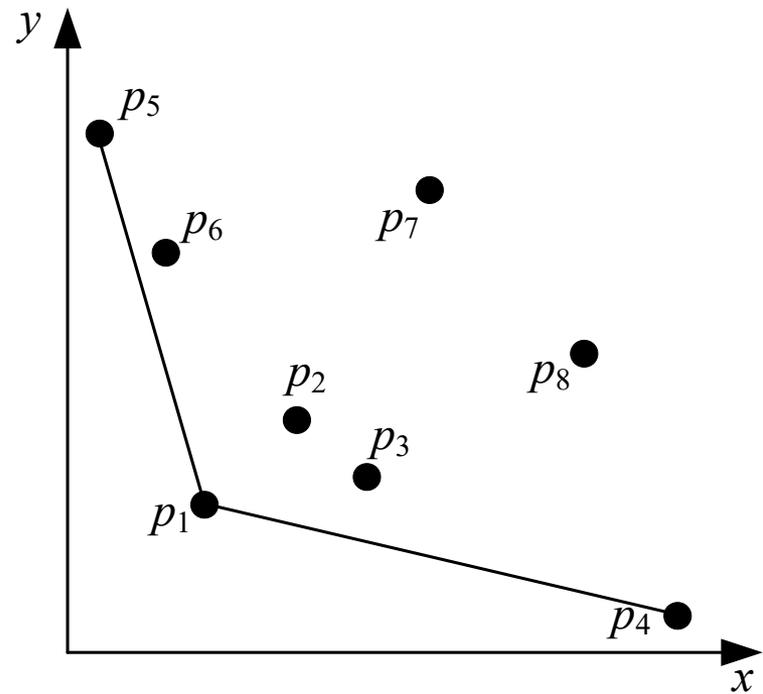
Stephan Börzsönyi, Donald Kossmann, Konrad Stocker.
"The Skyline Operator". In Proceedings 17th International
Conference on Data Engineering: 421–430, 2001

Skyline vs. konvexe Hülle

Skyline



Konvexe Hülle

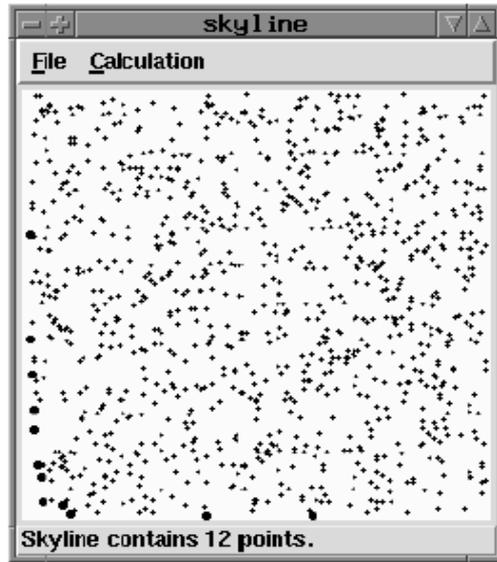


Skyline in Standard-SQL?

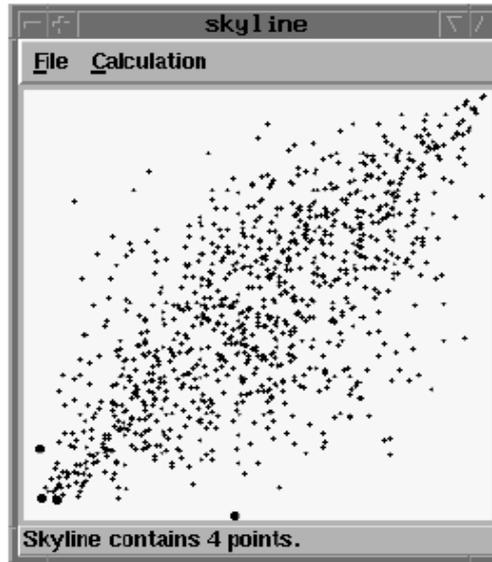
```
select k.Ort
from KostenVergleich k
where not exists
  (select * from KostenVergleich dom
   where dom.Miete <= k.Miete and dom.Beitrag <= k.Beitrag and
        (dom.Miete < k.Miete or dom.Beitrag < k.Beitrag))
```

Datenbanksysteme werden kaum in der Lage sein,
eine solche Anfrage angemessen zu optimieren

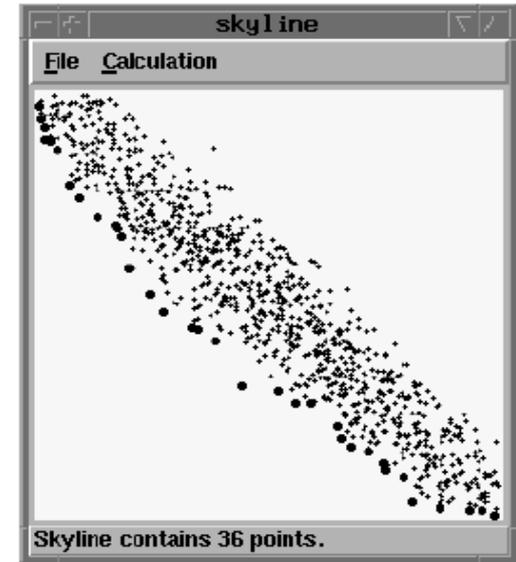
Pures SQL ineffektiv auf großen Datenmengen



independent



correlated



anti-correlated

Je nach Korrelation der Daten
kann Skyline groß oder klein sein

Skyline als SQL-Erweiterung

with KostenVergleich **as** (**select** m.Ort, m.Miete, k.Beitrag
from Mietspiegel m, Kindergarten k
where m.Ort=k.Ort)

select k.Ort
from KostenVergleich k
skyline of k.Miete **min**, k.Beitrag **min**

```
SELECT ... FROM ... WHERE ...  
GROUP BY ... HAVING ...  
SKYLINE OF [DISTINCT] d1 [MIN | MAX | DIFF ],  
                ..., dm [MIN | MAX | DIFF ]  
ORDER BY ...
```

MIN: Minimierung der Skyline-Werte, MAX: Maximierung, DIFF: Unterschiedlich (different)

Verfahren zur Skyline-Berechnung

- **Index-basierte Skyline**
- **Block Nested Loop (BNL)**: Führt geschachtelte Schleifen über Datenblöcke aus, blockweises Lesen
- **Divide and Conquer (DC)**: Teilt den Raum auf, löst das Problem in Teilräumen und fügt die Teillösungen zusammen
- **Nearest-Neighbor (NN)**: Verwendet R-Baum-Index und führt Folge von Nächste-Nachbarn-Anfragen aus, bis Skyline-Objekte gefunden sind

Indexbasierte Skyline-Bestimmung

- Organisiere Datenpunkte $p=(x_1, x_2, \dots, x_d)$ in d Listen, so dass p in Liste i kommt, wenn p_i am kleinsten
- Listen L_i aufsteigend sortiert (nach Attribut p_i)
- Bearbeitung von Teilmengen aus L_i mit gleichem Wert p_i

list 1		list 2	
$a(1, 9)$	$minC=1$	$k(9, 1)$	$minC=1$
$b(2, 10)$	$minC=2$	$i(3, 2), m(6, 2)$	$minC=2$
$c(4, 8)$	$minC=4$	$h(4, 3), n(8, 3)$	$minC=3$
$g(5, 6)$	$minC=5$	$l(10, 4)$	$minC=4$
$d(6, 7)$	$minC=6$	$f(7, 5)$	$minC=5$
$e(9, 10)$	$minC=9$		

Indexbasierte Skyline-Bestimmung

- Berechnung der Skyline auf Teilmengen der Punkte
- Füge die nicht dominierten Punkte in Skyline-Liste ein

list 1		list 2	
$a(1, 9)$	$minC=1$	$k(9, 1)$	$minC=1$
$b(2, 10)$	$minC=2$	$i(3, 2), m(6, 2)$	$minC=2$
$c(4, 8)$	$minC=4$	$h(4, 3), n(8, 3)$	$minC=3$
$g(5, 6)$	$minC=5$	$l(10, 4)$	$minC=4$
$d(6, 7)$	$minC=6$	$f(7, 5)$	$minC=5$
$e(9, 10)$	$minC=9$		

- ▶ Skyline := {}
- ▶ Lade erste Elemente aus jeder Liste, behandle Element mit kleinstem $minC$ (hier a, k), wähle a und füge a zur Skyline-Liste hinzu.

- ▶ Betrachte Elemente b und k bzgl. $minC$, füge k zur Skyline hinzu, b dominiert von a entfällt
- ▶ Lade $\{i, m\}$; Betrachte Elemente in $\{i, m\}$
- ▶ Füge i zur Skyline-Liste hinzu
- ▶ Algorithmus hält, weil keine anderen Elemente besser sind als i
 - ▶ Für den Test wird B-Baum in der jeweiligen Dimension benötigt
- ▶ Skyline ist $\{a, k, i\}$

Warum hält der Algorithmus mit i in der Skyline?

- $i = (3, 2)$
- Verwende B-Baum der Dimension 1, um 3 zu finden

- Scanne nach links bis MinC_1 :
 b schon gesehen
 - Bei $i=(4, 2)$ wäre c noch zu prüfen

list 1		list 2	
$a(1, 9)$	$\text{minC}=1$	$k(9, 1)$	$\text{minC}=1$
$b(2, 10)$	$\text{minC}=2$	$i(3, 2), m(6, 2)$	$\text{minC}=2$
$c(4, 8)$	$\text{minC}=4$	$h(4, 3), n(8, 3)$	$\text{minC}=3$
$g(5, 6)$	$\text{minC}=5$	$l(10, 4)$	$\text{minC}=4$
$d(6, 7)$	$\text{minC}=6$	$f(7, 5)$	$\text{minC}=5$
$e(9, 10)$	$\text{minC}=9$		

- Verwende B-Baum der Dimension 2, um 2 zu finden
- Scanne nach links bis MinC_2 :
 k schon gesehen
 - Bei $i=(3, 3)$ wäre h noch zu prüfen (m käme auch noch vorher)

Nur, effektiv,
wenn B-Baum-Indexe
vorhanden!

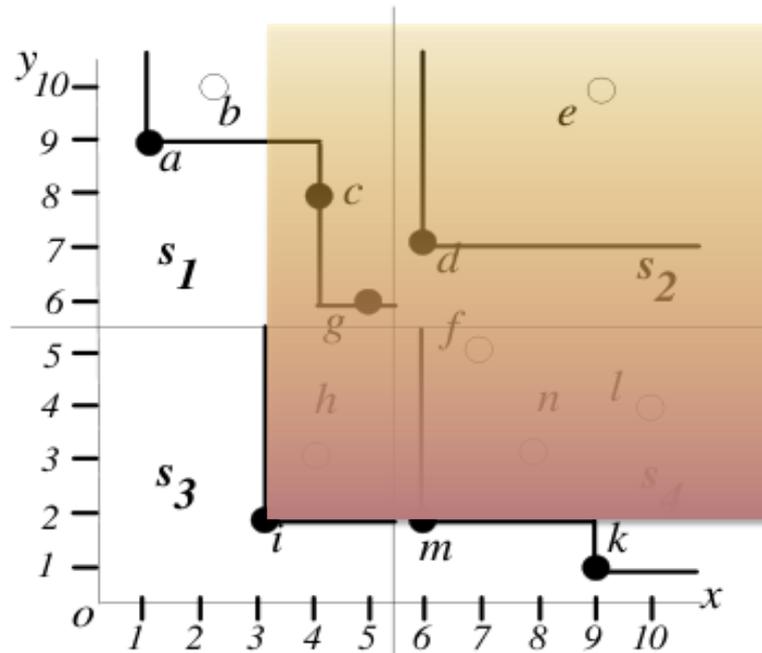
Meist für Punkte nicht
vorgesehen

Algorithmus Block-Nested-Loop (BNL)

- Gegeben:
 - Datei 1 mit Tupeln, Hilfsdatei Datei 2
 - Puffer (Block, klein) und Fenster (klein) im Hauptspeicher
- Bis Datei 1 leer:
 - Lese nächste Tupel blockweise aus Datei 1 in Puffer
 - Für alle Tupel t im Block:
 - Für alle Tupel t' im Fenster
 - Vergleiche t mit t' , prüfe, ob t dominiert wird
 - Entferne t' aus Fenster, wenn durch t dominiert
 - Übernehme t in Fenster, wenn nicht dominiert
 - Falls Fenster voll, schreibe t in Datei 2
- Wenn Tupel in Datei 2 gefunden, schreibe Fenster in Datei 2
mache mit Datei 2 als Datei 1 weiter

Divide and Conquer (Sykline-Algorithmus DC)

- Zerlege Datenmenge in Partitionen, so dass jede Partition in den Speicher passt (ggf. mit Vorabfilterung pro Block – Early Skyline – siehe indexbasierte Verfahren)
- Bestimme Skyline pro Partition (z.B. mit BNL)
- Kombiniere Teillösungen zur Gesamtlösung



Teillösungen

{a,c,g}

{d}

{i}

{m,k}

Gesamtlösung

{a,i,k}

Nachteil: Hohe IO-Kosten

Funktioniert nur bei kleiner Skyline

Early Skyline

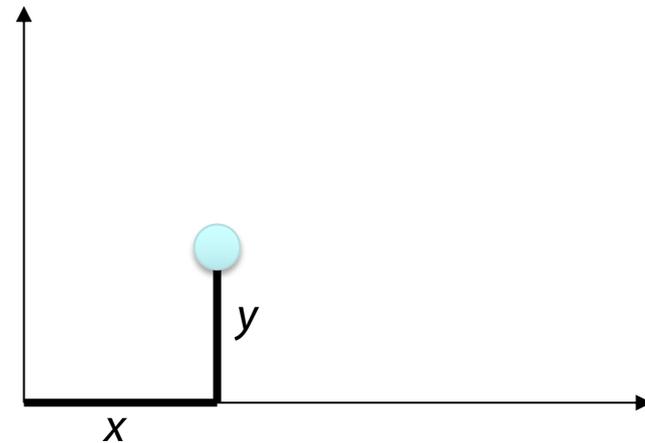
- Für jeden Block eliminiere Punkte, die von anderen dominiert werden
- Falls Index vorhanden, kann dieser verwendet werden
- Erhöhte CPU-Kosten
- Weniger IO, da weniger Punkte/Tupel in den nachfolgenden Partitionierungsschritten
- Besonders effektiv bei kleiner Skyline

Bewertung

- BNL i.a. besser als DC, sofern Blockgröße hoch
- Early Skyline effektiv für DC
 - Kleinere Partitionen : Algorithmus terminiert schnell
- DC ohne Early Skyline:
 - Schwach: Hohe I/O-Komplexität
- BNL-Varianten gut, wenn Skyline klein
 - Bei mehr Dimensionen (Skyline wird größer) wird DC besser
 - Dito bei mehr Speicher (weniger Partitionen)

Annahme

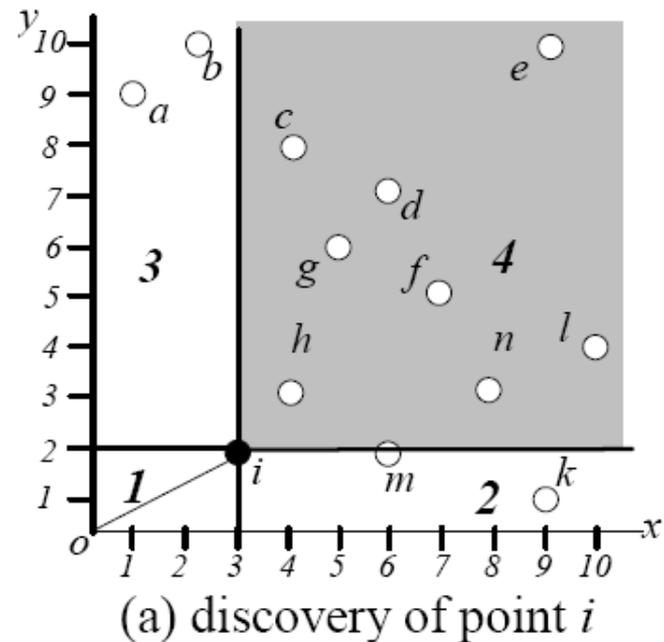
- Datenpunkte in k-d-Baum eingetragen
- Abstand eines Punktes vom Ursprung als Manhattan-Abstand modelliert
- NN-Anfrage mit k-d-Baum



$$\text{mindist}(mbr(e)) = x + y$$

Skyline-Algorithmus Nächste Nachbarn (NN)

- Führe NN-Anfrage vom Ursprung aus zur Bestimmung des nächsten Nachbarn von o
- Kein Punkt in der dominierten Region wird mehr betrachtet
- Ergebnis der NN-Suche verwendet zur Partitionierung des Raumes



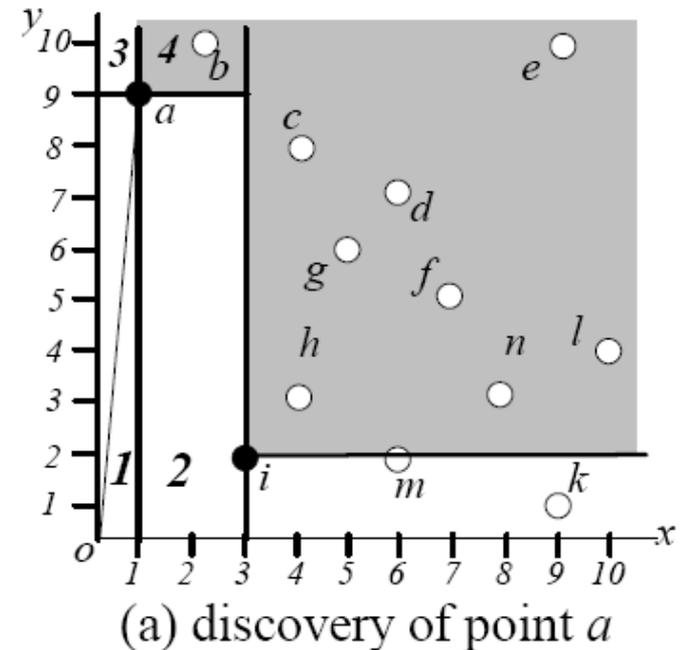
Zwei Partitionen

Partition 1: Fläche 3

Partition 2: Fläche 2

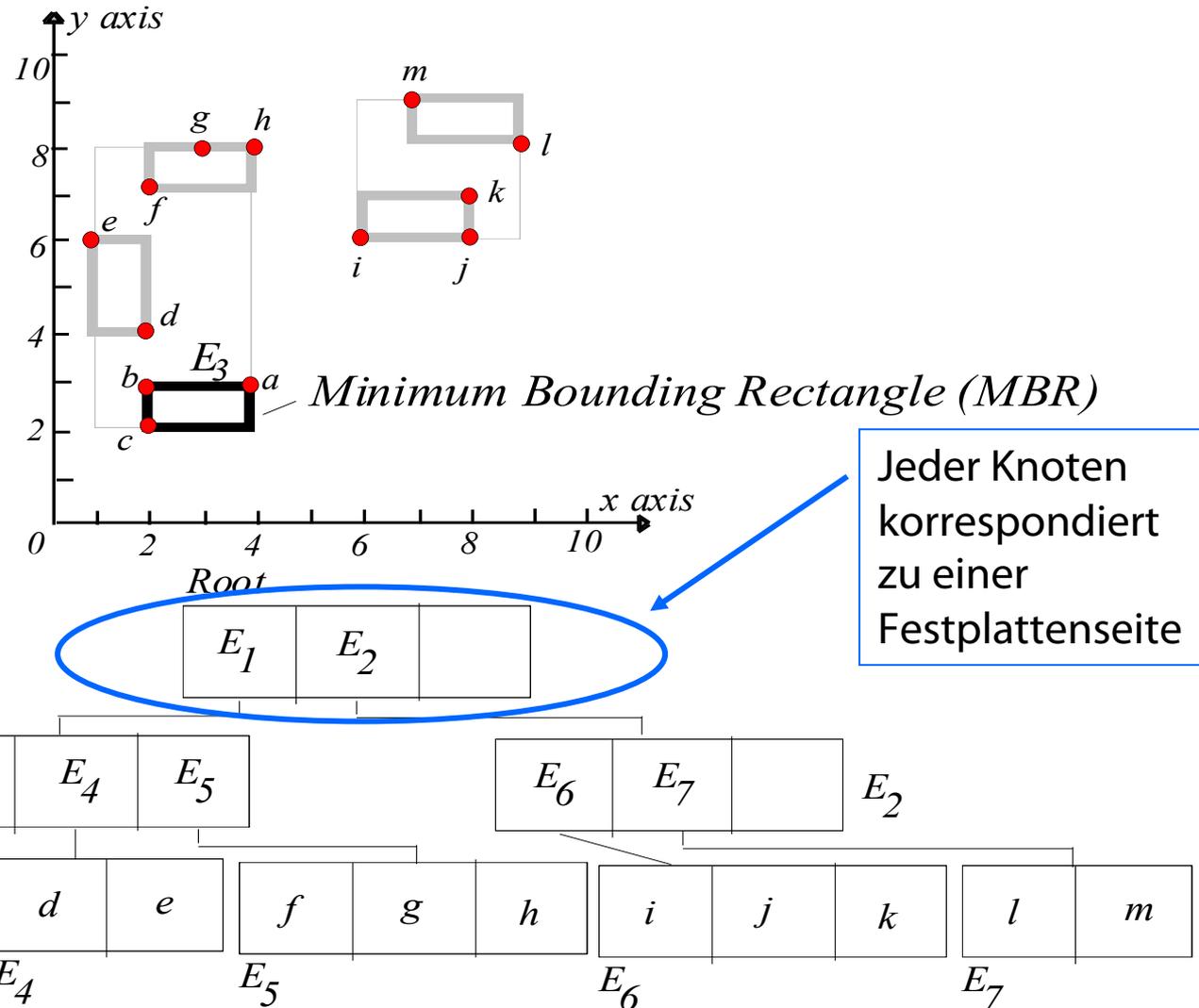
Nächste Nachbarn (NN)

- Partitionen kommen auf Agenda
- Solange Agenda nicht leer:
 - Nehme Partition von Agenda und führe NN aus
- Vergleich mit BNL, DC?
 - **K-d-Baum** notwendig

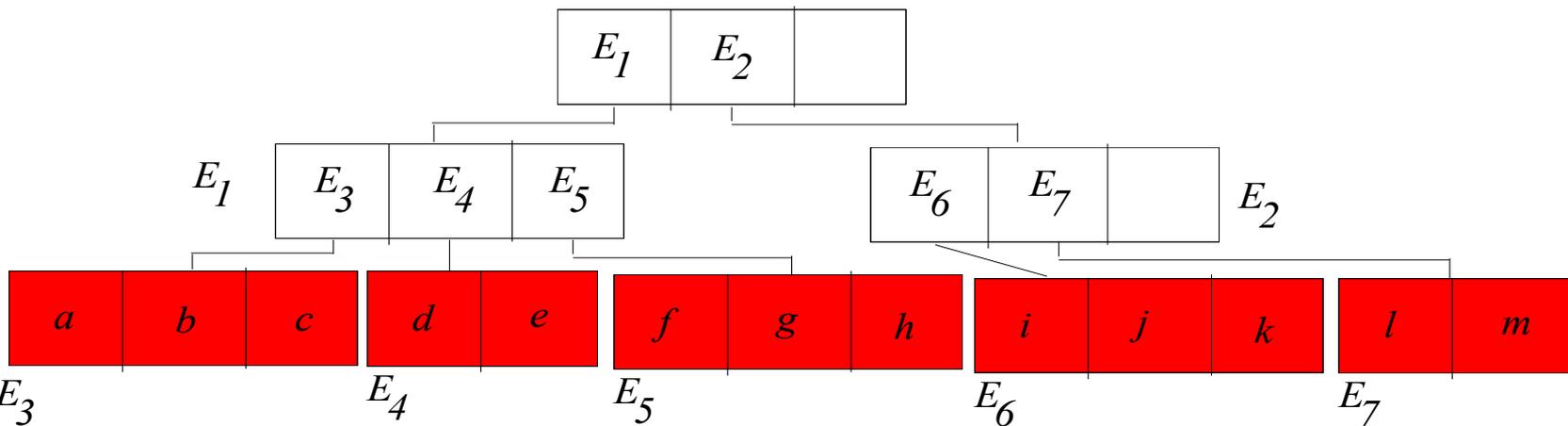
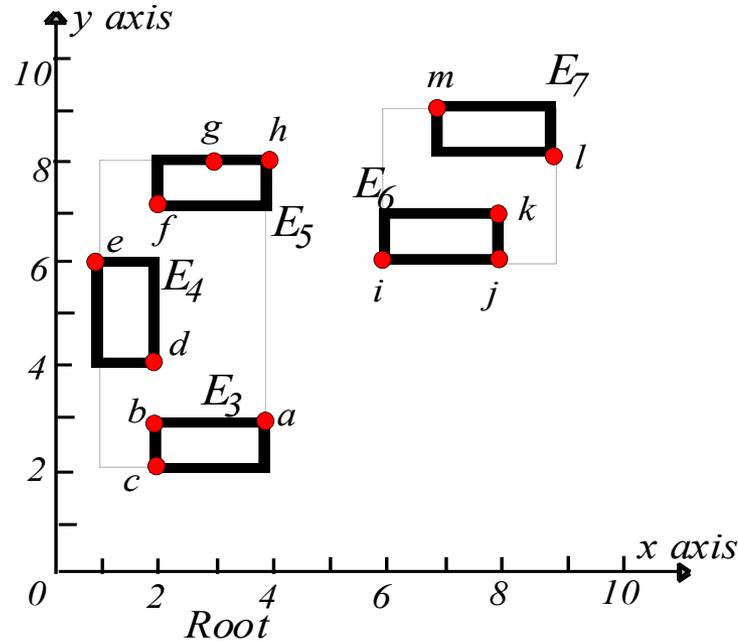


- Skyline mit **R-Bäumen über Punktdaten?**
 - Branch and Bound Skyline (BBS)

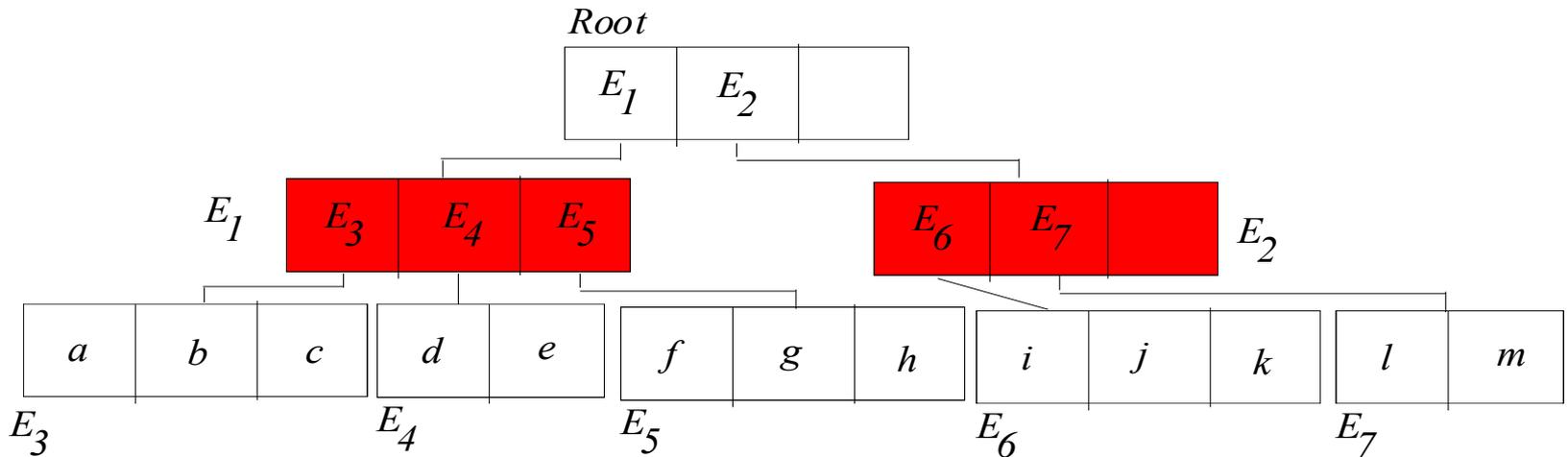
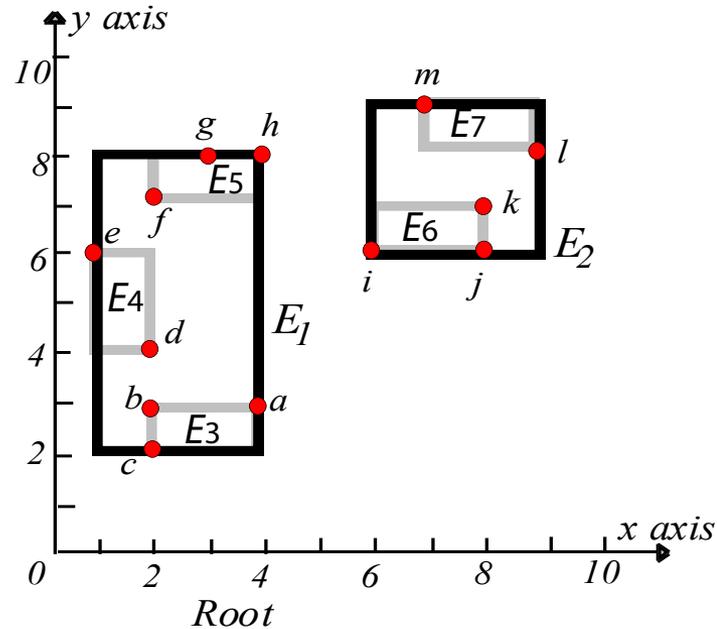
BBS-Algorithmus: R-Bäume mit Punktdaten



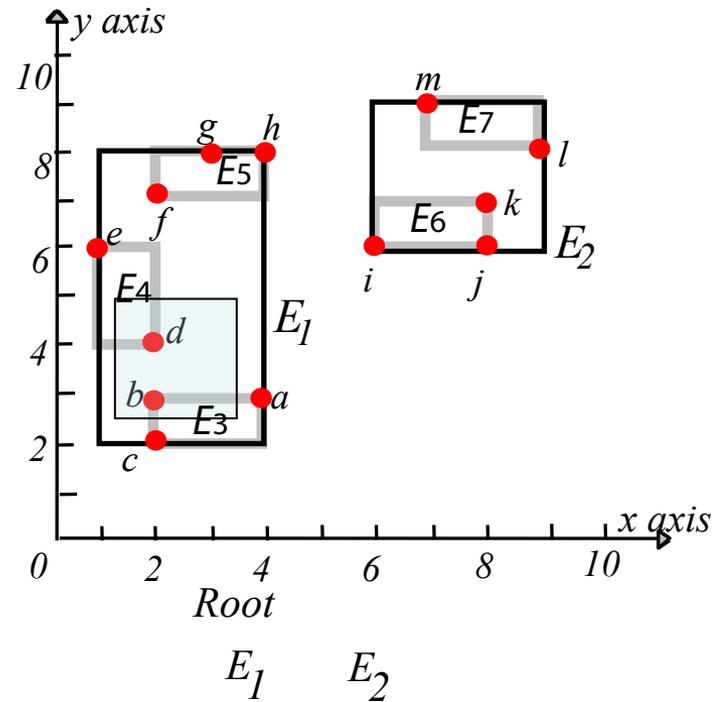
R-Bäume – Wiederholung



R-Bäume – Wiederholung



R-Bäume – Wiederholung

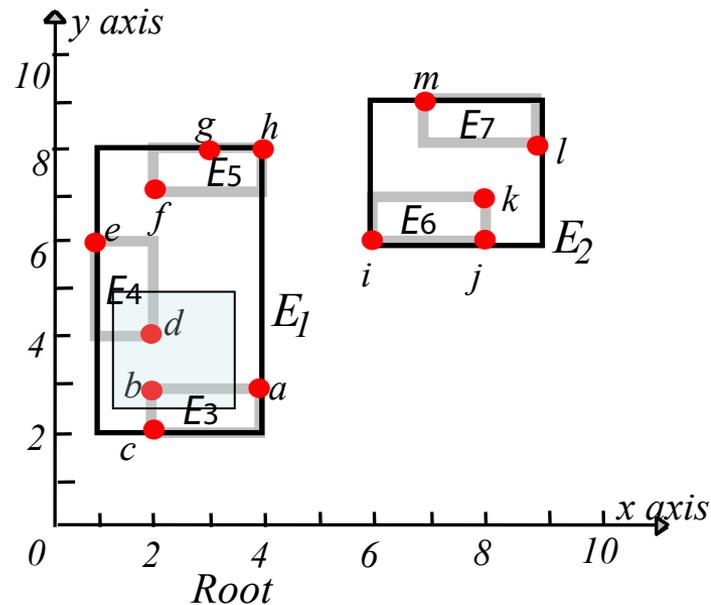


E_1 E_3 E_4 E_5 E_6 E_7 E_2

a b c d e f g h i j k l m

E_3 E_4 E_5 E_6 E_7

R-Bäume – Wiederholung



Root
 E_1 E_2

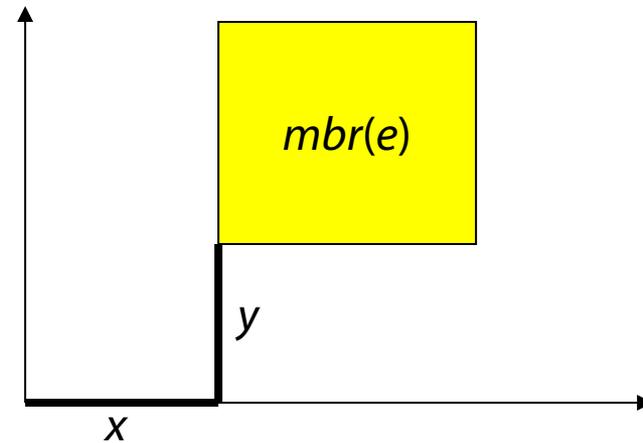
E_1 E_3 E_4 E_5 E_6 E_7 E_2

a b c d e f g h i j k l m
 E_3 E_4 E_5 E_6 E_7

BBS Algorithmus

Verwendung einer Prioritätswarteschlange, in der R-Baum-Einträge e nach **Manhattan**-Abstand verwaltet werden:

Links-unten von $mbr(e)$ bis Nullpunkt



$$\text{mindist}(mbr(e)) = x + y$$

D. Papadias, Y. Tao, G. Fu und B. Seeger, „An Optimal and Progressive Algorithm for Skyline Queries,“ ACM SIGMOD, pp. 467-478, **2003**

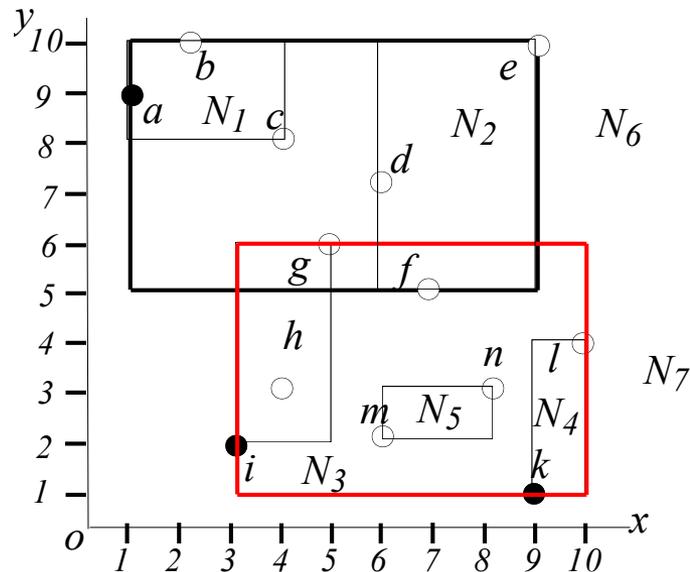
Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. ACM Trans. Database Syst. 30, 1, 41–82. **2005**.

Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. Aggregate nearest neighbor queries in spatial databases, ACM Trans. Database Syst. 30, 2, 529-576, **2005**.

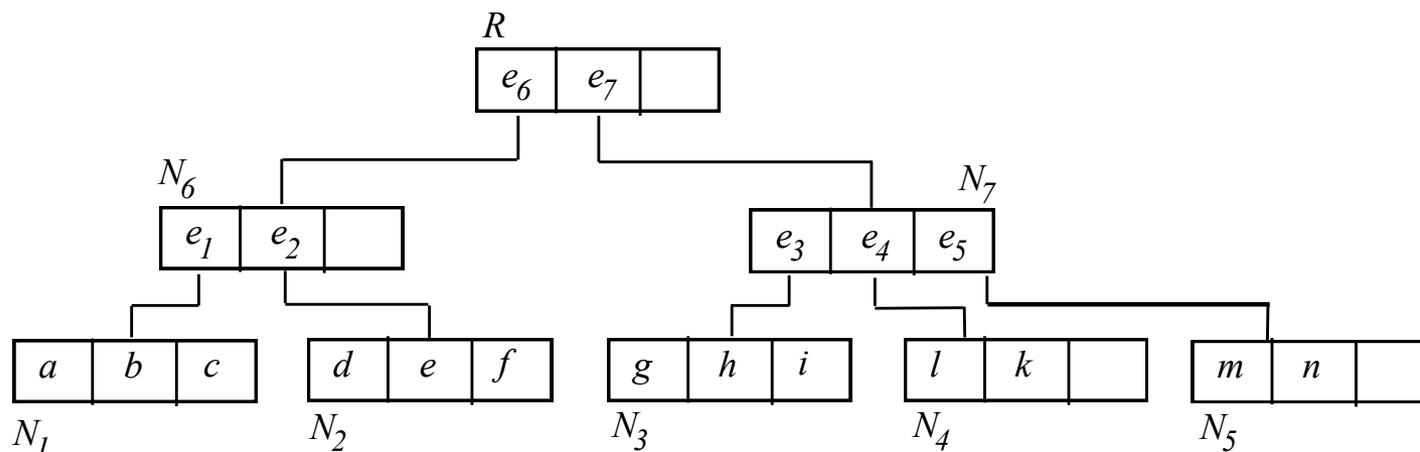
BBS Algorithmus – Entscheidungen

1. Wann verzweigen (**branch**): Welchen Teil des Suchraums soll als nächstes betrachtet werden?
2. Wie begrenzen (**bound**): Welche Teile des Suchraums können sicher eliminiert werden

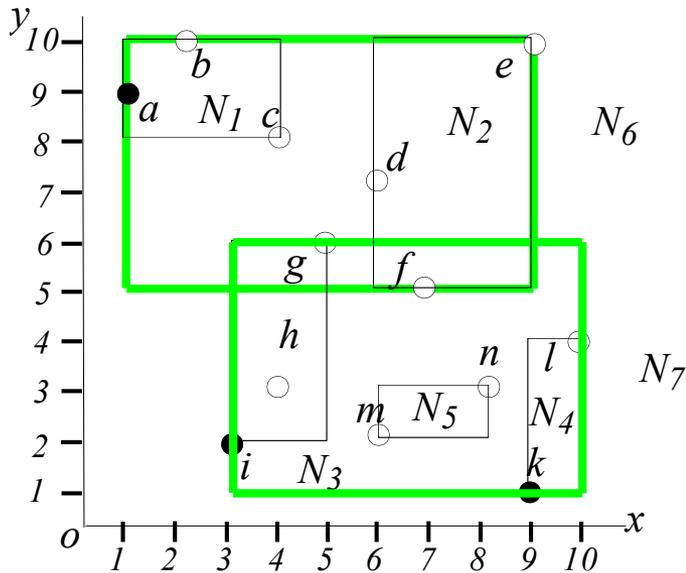
BBS Algorithmus – Beispiel



- Nehme an, alle Punkte sind in einem R-Baum eingetragen
- $\text{mindist}(\text{mbr}) = \text{Manhattan-Distanz zwischen Punkt links-unten und Ursprung}$



BBS Algorithmus – Beispiel

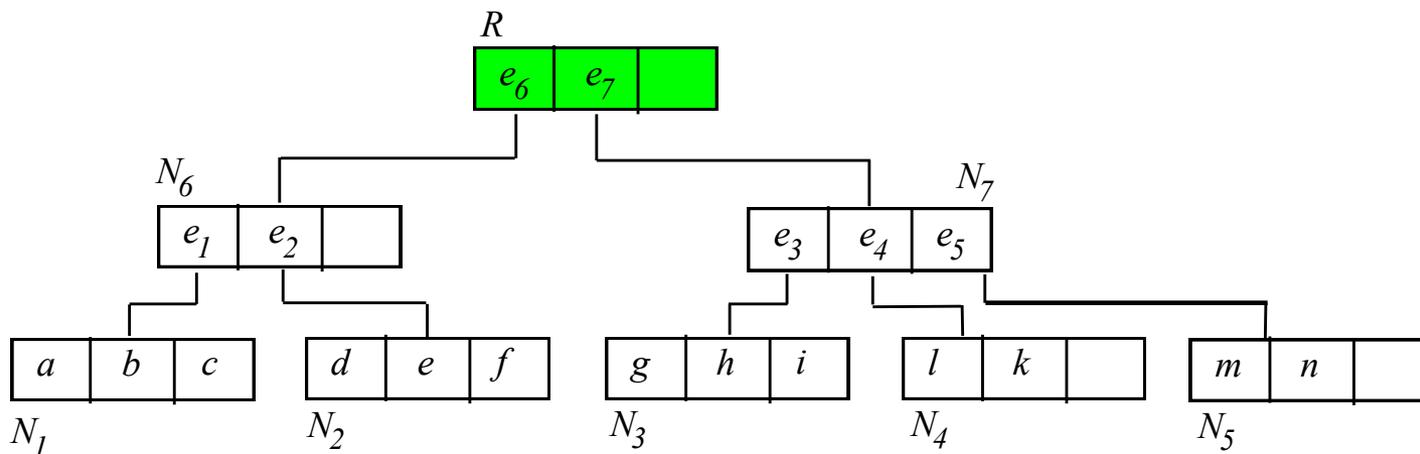


- Warteschlangenschlüssel = **mindist** vom MBR.

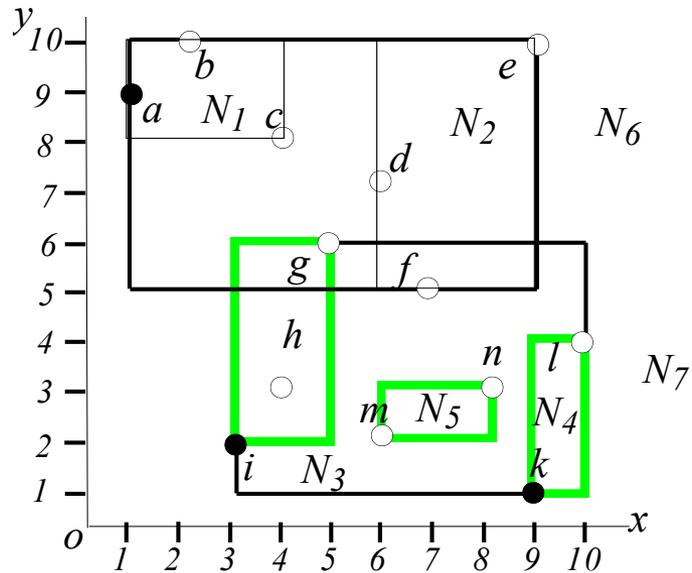
Aktion
access root

Warteschlange
 $\langle e_{7,4} \rangle \langle e_{6,6} \rangle$

S
 \emptyset

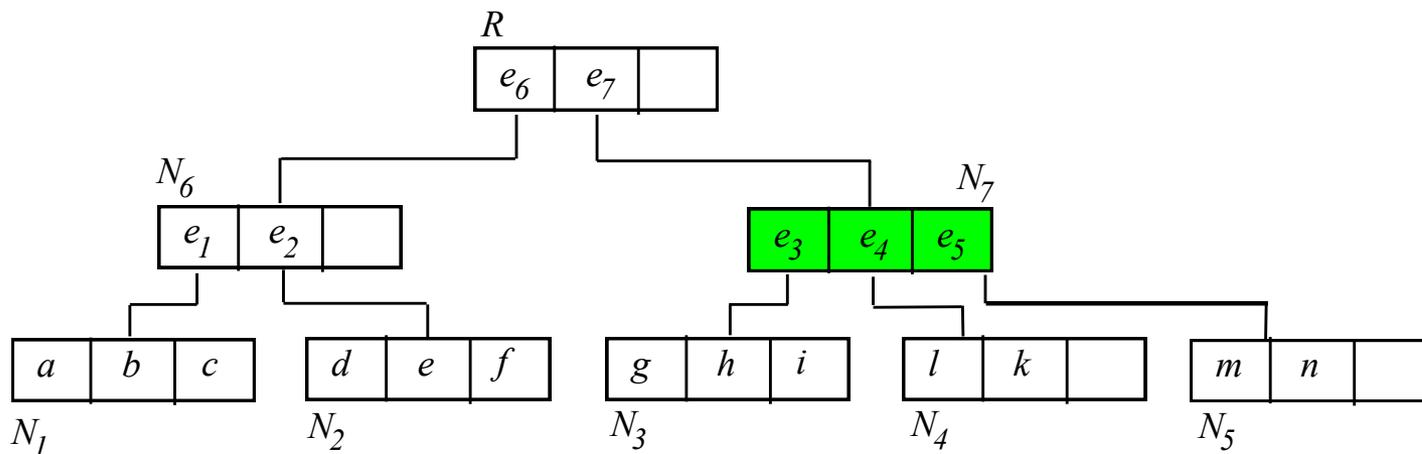


BBS Algorithmus – Beispiel

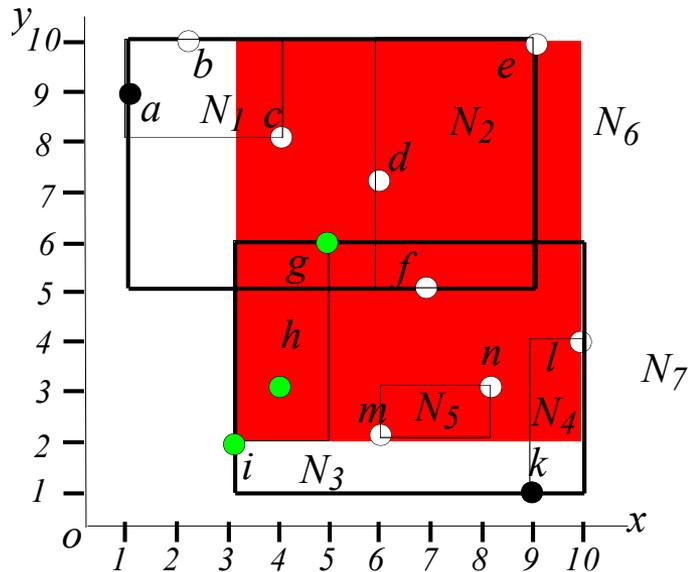


- Verarbeitung nach mindist-Werten sortiert

Aktion	Warteschlange	S
access root	$\langle e_7, 4 \rangle \langle e_6, 6 \rangle$	\emptyset
expand e_7	$\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$	\emptyset



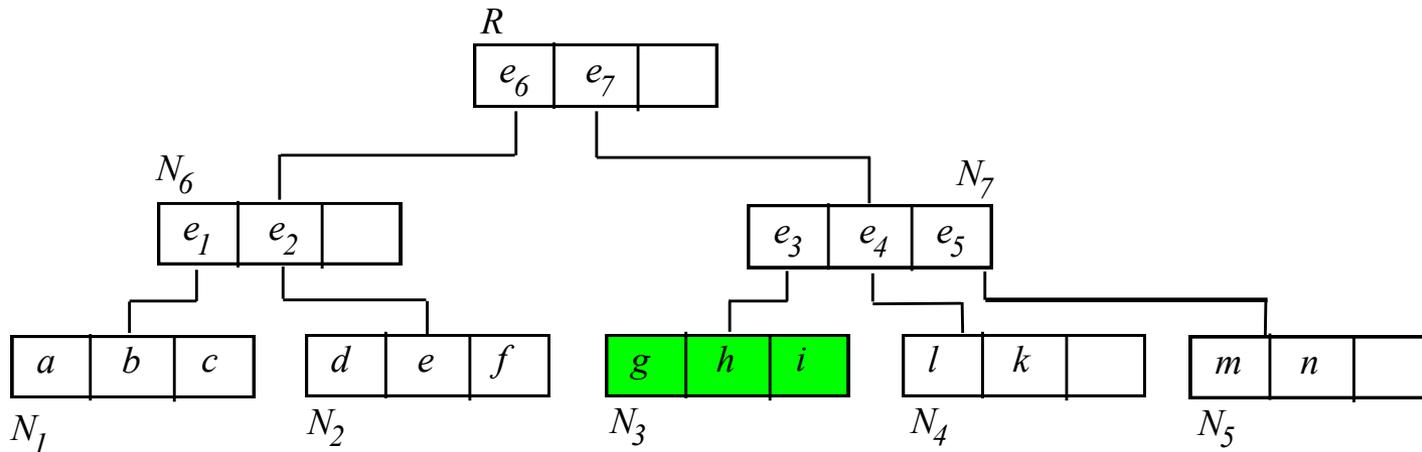
BBS Algorithmus – Beispiel



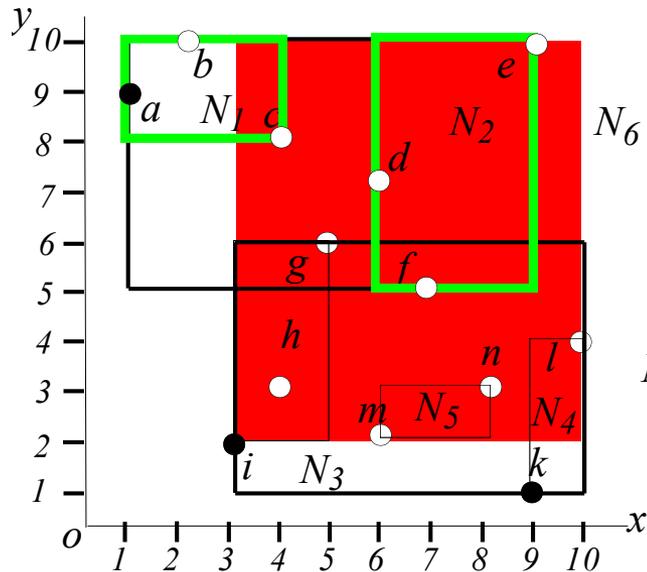
Aktion
 access root
 expand e_7
 expand e_3

Warteschlange
 $\langle e_{7,4} \rangle \langle e_{6,6} \rangle$
 $\langle e_{3,5} \rangle \langle e_{6,6} \rangle \langle e_{5,8} \rangle \langle e_{4,10} \rangle$
 $\langle i,5 \rangle \langle e_{6,6} \rangle \langle e_{5,8} \rangle \langle e_{4,10} \rangle$

S
 \emptyset
 \emptyset
 $\{i\}$



BBS Algorithmus – Beispiel

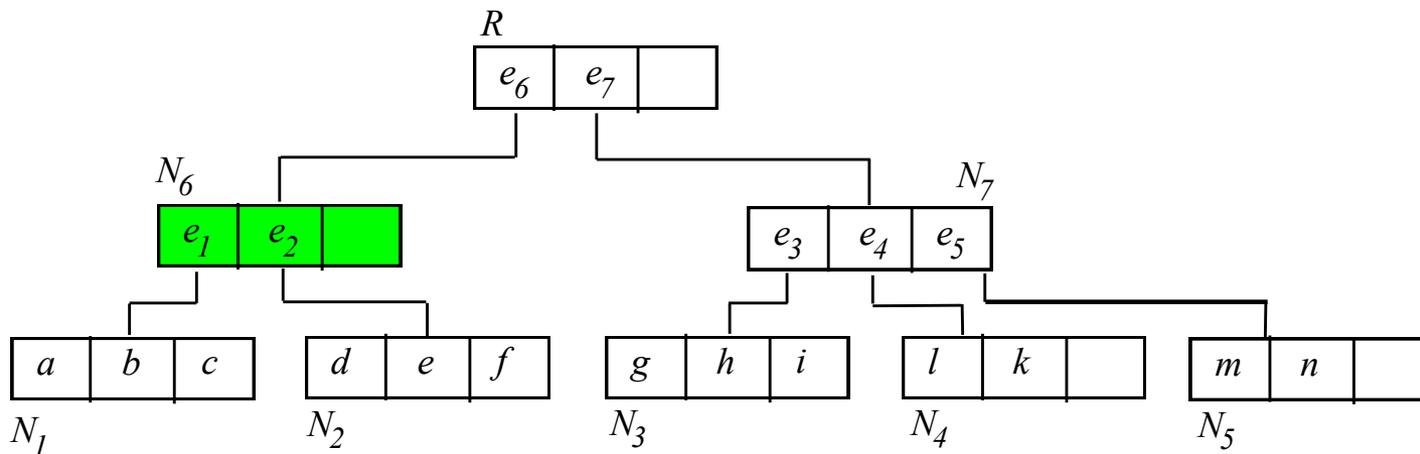


Aktion
 access root
 expand e_7
 expand e_3
 expand e_6

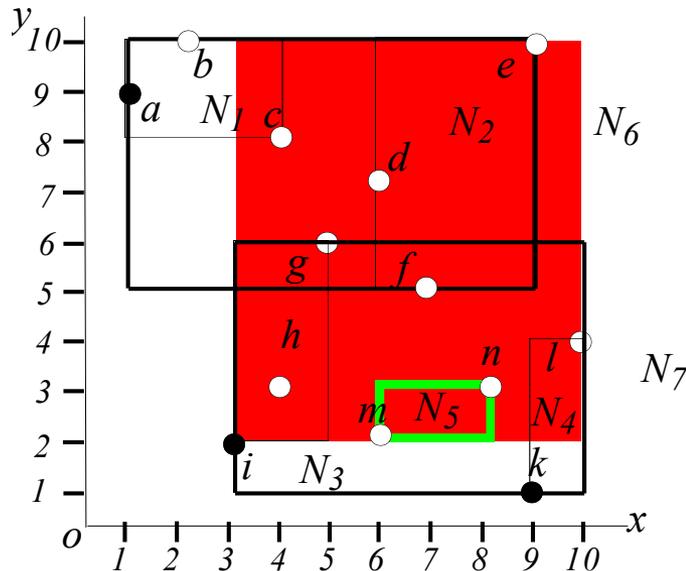
Warteschlange

$\langle e_7, 4 \rangle \langle e_6, 6 \rangle$
 $\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$
 $\langle i, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$
 $\langle e_5, 8 \rangle \langle e_1, 9 \rangle \langle e_4, 10 \rangle$

S
 \emptyset
 \emptyset
 $\{i\}$
 $\{i\}$



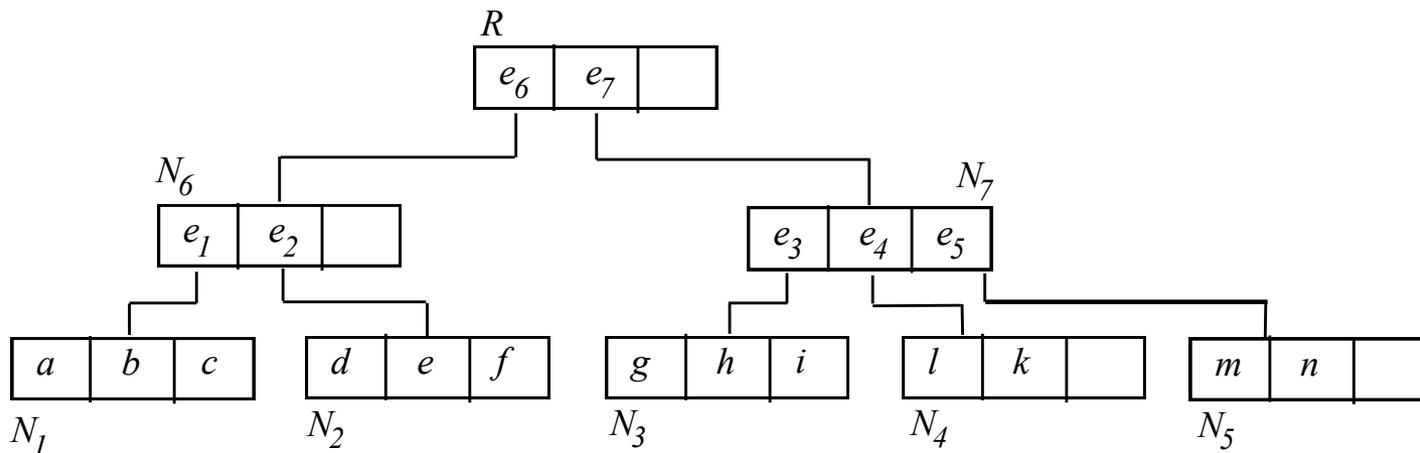
BBS Algorithmus – Beispiel



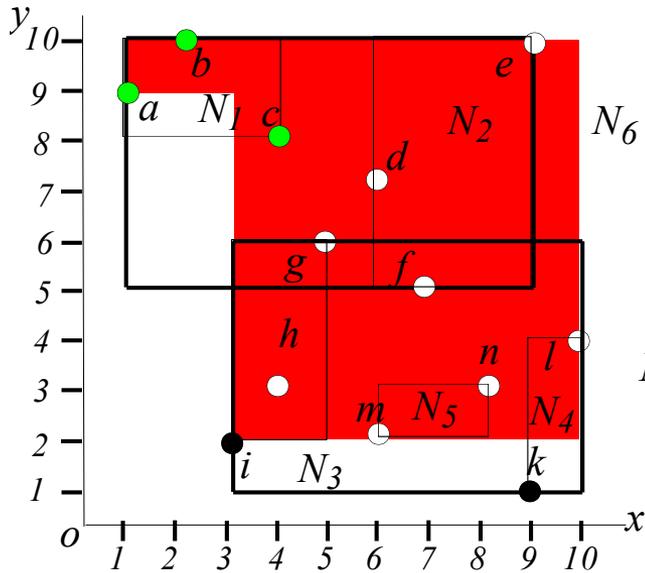
Aktion
 access root
 expand e_7
 expand e_3
 expand e_6
 remove e_5

Warteschlange
 $\langle e_7, 4 \rangle \langle e_6, 6 \rangle$
 $\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$
 $\langle i, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$
 $\langle e_5, 8 \rangle \langle e_1, 9 \rangle \langle e_4, 10 \rangle$
 $\langle e_1, 9 \rangle \langle e_4, 10 \rangle$

S
 \emptyset
 \emptyset
 $\{i\}$
 $\{i\}$
 $\{i\}$



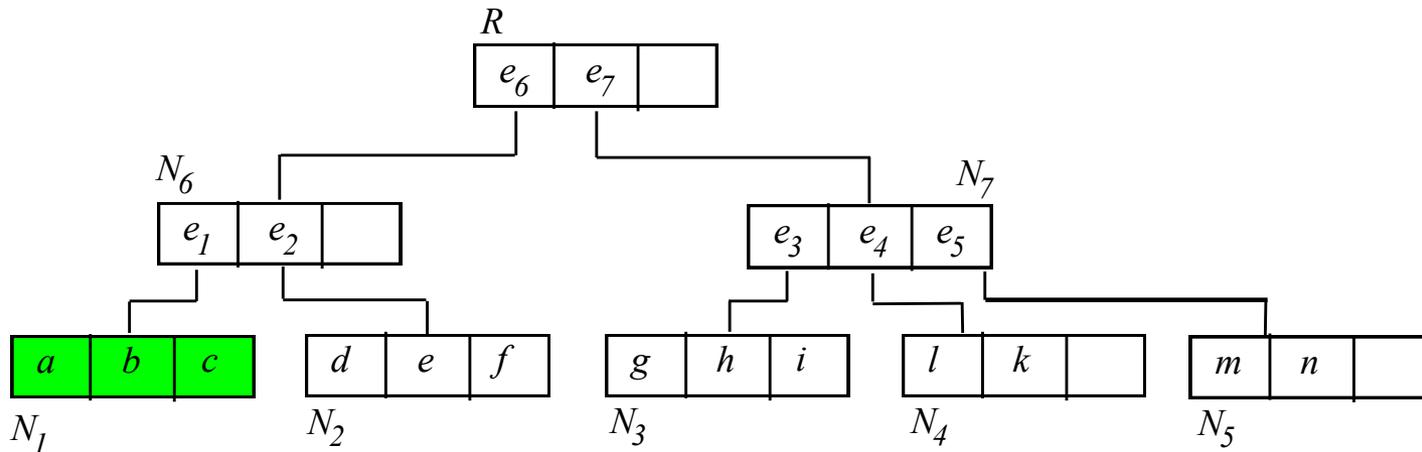
BBS Algorithmus – Beispiel



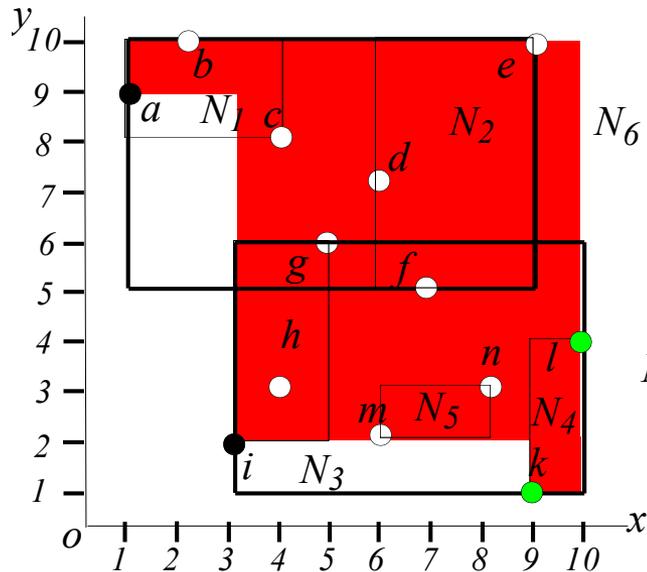
Aktion
 access root
 expand e_7
 expand e_3
 expand e_6
 remove e_5
 expand e_1

Warteschlange
 $\langle e_7, 4 \rangle \langle e_6, 6 \rangle$
 $\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$
 $\langle i, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$
 $\langle e_5, 8 \rangle \langle e_1, 9 \rangle \langle e_4, 10 \rangle$
 $\langle e_1, 9 \rangle \langle e_4, 10 \rangle$
 $\langle a, 10 \rangle \langle e_4, 10 \rangle$

S
 \emptyset
 \emptyset
 $\{i\}$
 $\{i\}$
 $\{i\}$
 $\{i, a\}$



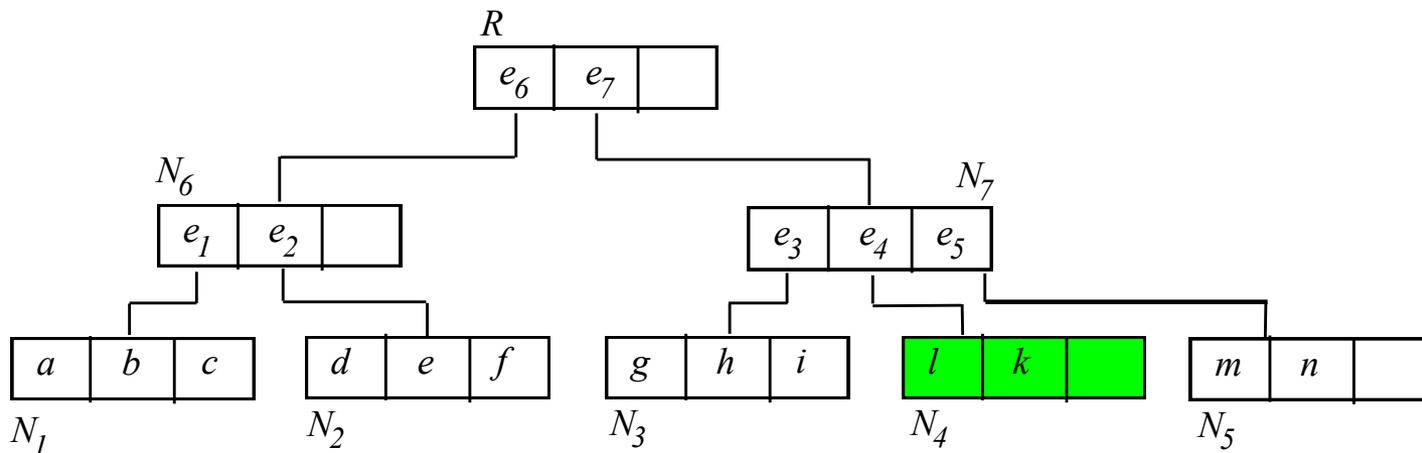
BBS Algorithmus – Beispiel



Aktion
 access root
 expand e_7
 expand e_3
 expand e_6
 remove e_5
 expand e_1
 expand e_4

Warteschlange
 $\langle e_7, 4 \rangle \langle e_6, 6 \rangle$
 $\langle e_3, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$
 $\langle i, 5 \rangle \langle e_6, 6 \rangle \langle e_5, 8 \rangle \langle e_4, 10 \rangle$
 $\langle e_5, 8 \rangle \langle e_1, 9 \rangle \langle e_4, 10 \rangle$
 $\langle e_1, 9 \rangle \langle e_4, 10 \rangle$
 $\langle a, 10 \rangle \langle e_4, 10 \rangle$
 $\langle k, 10 \rangle$

S
 \emptyset
 \emptyset
 $\{i\}$
 $\{i\}$
 $\{i\}$
 $\{i, a\}$
 $\{i, a, k\}$



Branch-and-Bound-Skyline (BBS)

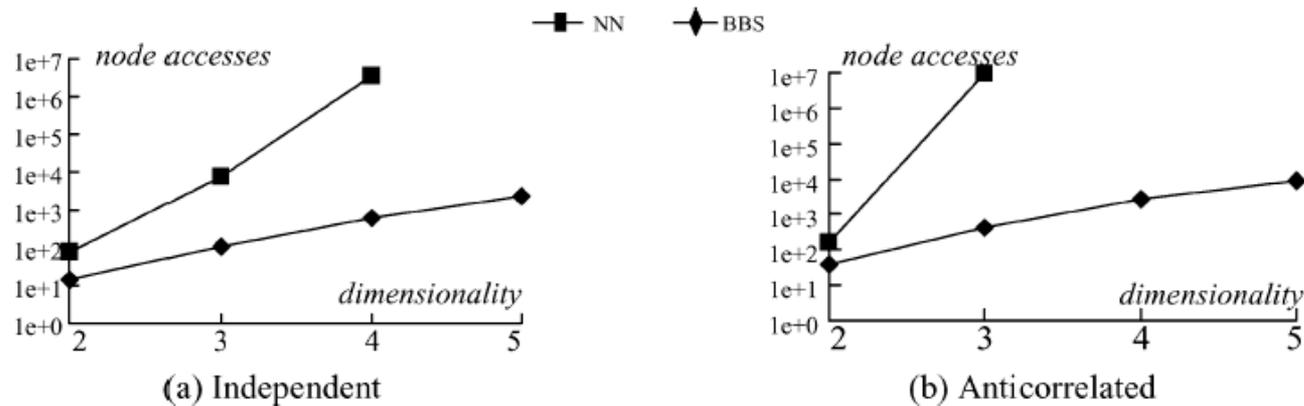
```
algorithm BBS (rt, type, f, test)
  // rt is an R-tree on the dataset, type denotes the type of priority queue
  // f is a point distance function, test is a tester for the data points
  let pq = build_priority_queue(type, map(lambda(o) return (o, f(o)), root(rt))
    result = {} // skyline
    object
  while not empty?(pq) do
    object := delete_next(pq)
    if test(object) then
      if always(lambda(o) not dominates?(o, object), result) then
        if point?(object) then
          result := filter(lambda(o) return not dominates?(object, o), result) U {object}
        else // intermediate object
          for child in children(object) do
            if always(lambda(o) return not dominates?(o, child), result) then
              insert((child, f(child)), pq)
  return result

function mindist(o)
  if point?(o) then
    return point_x(o) + point_y(o)
  else
    return point_x(leftbottom(mbr(o))) + point_y(leftbottom(mbr(o)))

BBS(data-rtree, 'min', mindist, lambda(x) true)
```

BBS Algorithm - Vergleich

BBS besser als vorige Skyline-Algorithmen bzgl.
CPU-Zeit und I/O-Zeit



Number of R-tree node accesses vs dimensionality

Weiterentwicklungen:

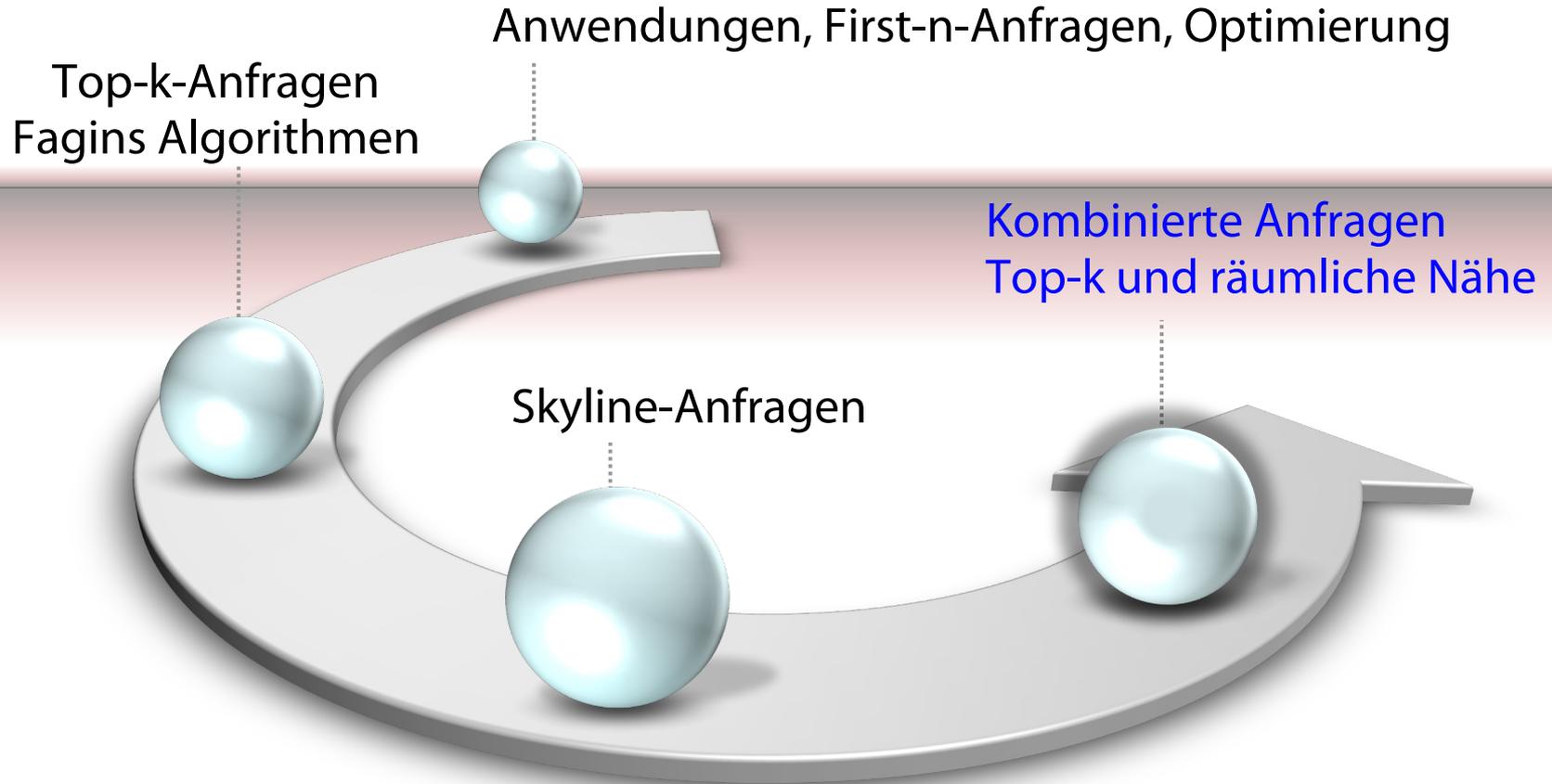
Subspace-Skylines, verteilte Skylines, Approximationen

Wir haben erst die Spitze des Eisbergs betrachtet!

Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui.
Aggregate nearest neighbor queries in spatial databases,
ACM Trans. Database Syst. 30, 2, 529-576, 2005

Non-Standard-Datenbanken und Data Mining

Von First-n- und Top-k-Anfragen zu Skyline-Anfrage



Top-k-Präferenzanfragen für räumliche Objekte

- **Rangordnung** von Objekten definiert auf Basis von benachbarten Objekten mit bestimmten Eigenschaften
- **Räumliche Nähe mit anderen Rangmaßen kombiniert**

Top-k-Präferenzanfragen für räumliche Objekte

- Gegeben

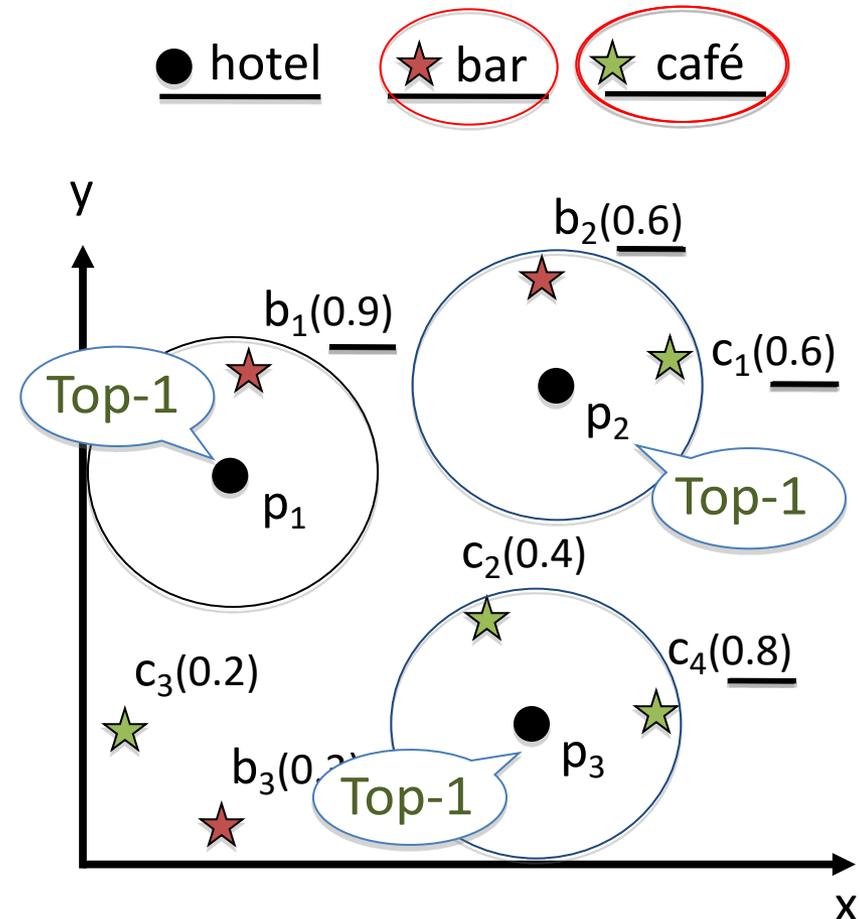
- Menge von **Datenobjekten** und bewertete **Merkmalsobjekte**

- Anfrage

- **Räumliche Nachbarschaft**
- **Merkmale**
 - Typ (z.B. Bar, Café)
 - Rangmaß
- Aggregation von partiellen Rangmaßen
 - Monotone Funktionen: sum, max, and min

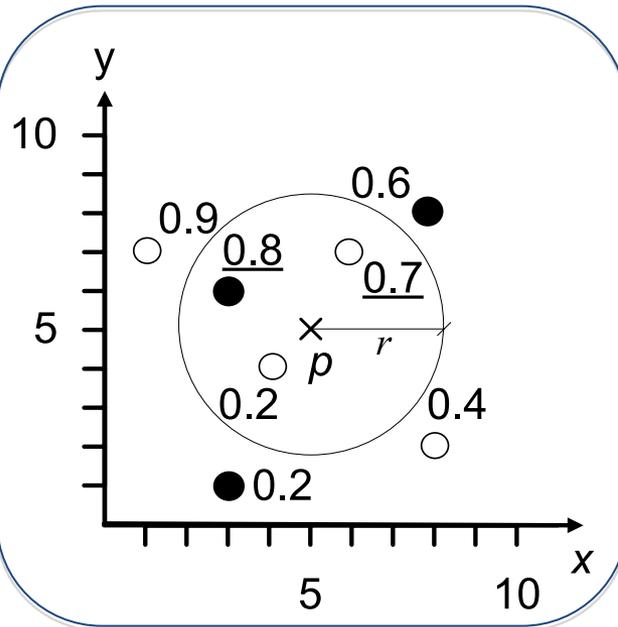
- Ergebnis

- k beste **Datenobjekte**



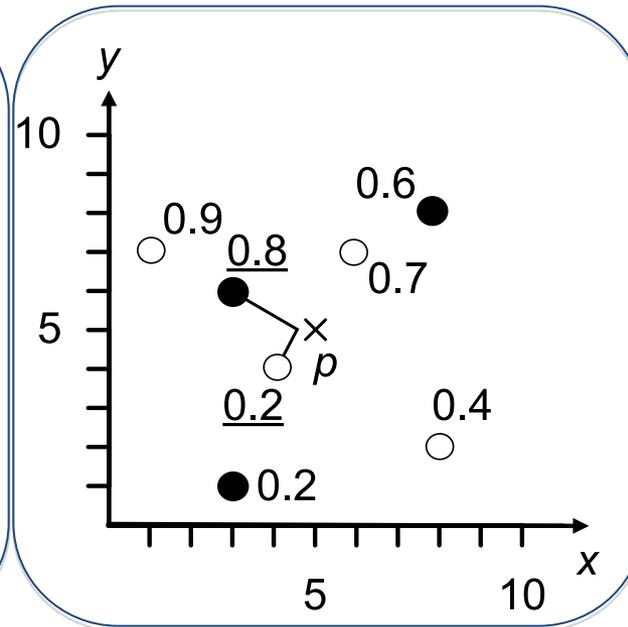
Beispiel (Aggregation = sum)

Range



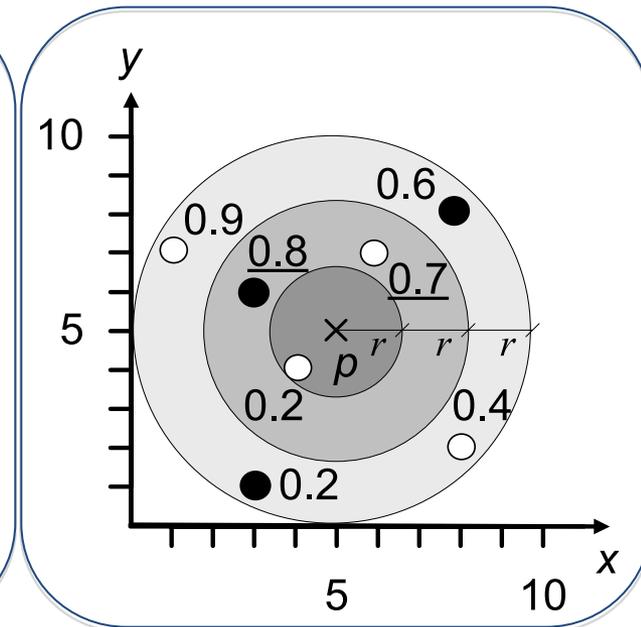
$$\text{score}(p) = 1.5$$

Nearest neighbor



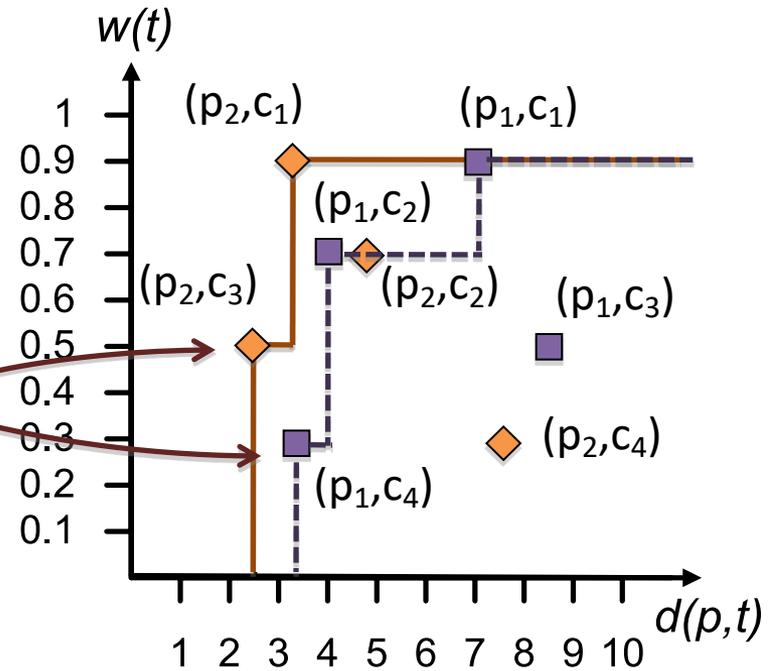
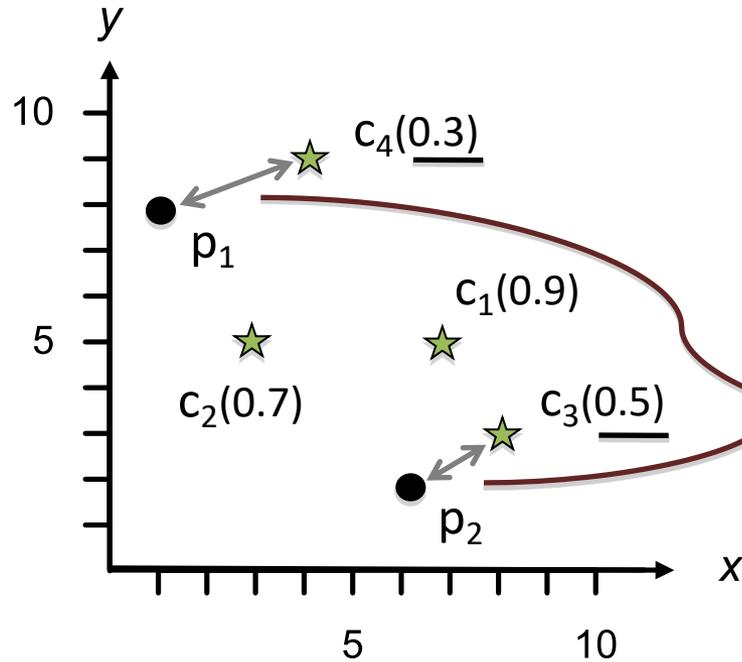
$$\text{score}(p) = 1.0$$

Influence



$$\text{score}(p) = 0.6$$

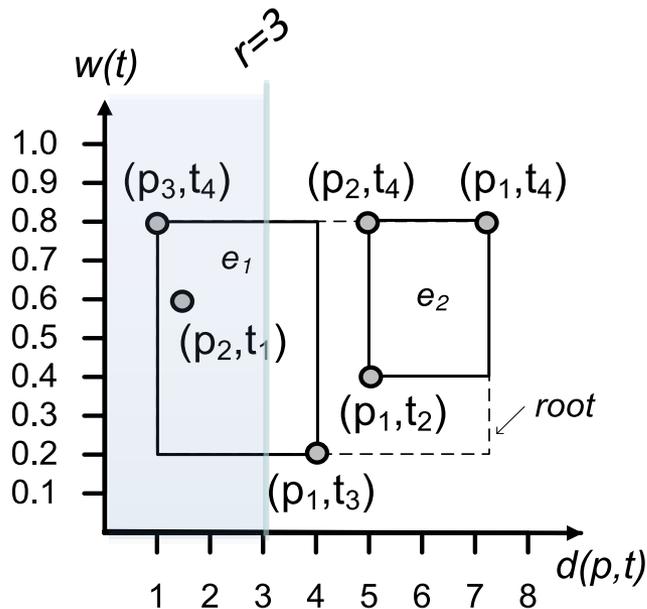
Abbildung auf Distanz-Rangmaß-Raum



- **Abbildung**
 - Paare (object, feature)
 - Raum [distance X score]

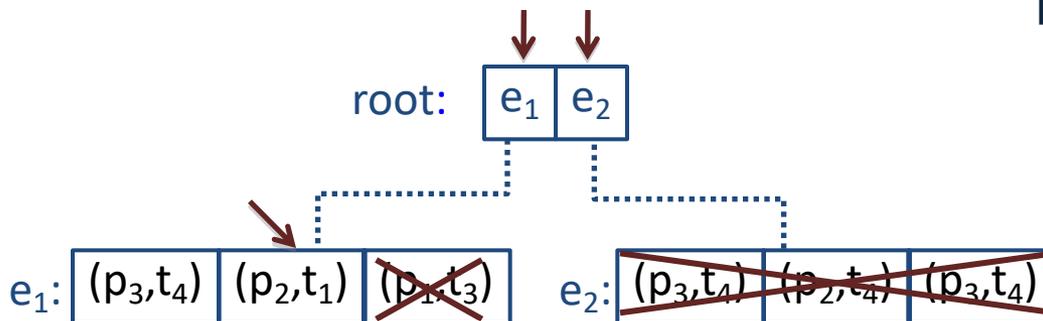
- **Skyline**
 - Minimiere: distance
 - Maximiere: score

Zugriff auf partielle Rangmaße

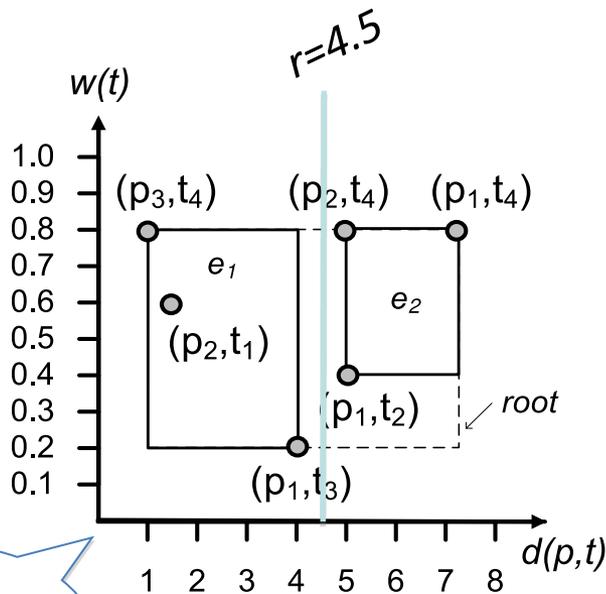


- Nur Knoteneinträge betrachtet, die räumliche Bedingungen erfüllen
 - Einträge werden in absteigender Rangfolge geladen
- Kleine Modifikationen für NN- und Einfluss-Anfragen

Max-PQ: $\langle p_3(0.8), p_2(0.6) \rangle$



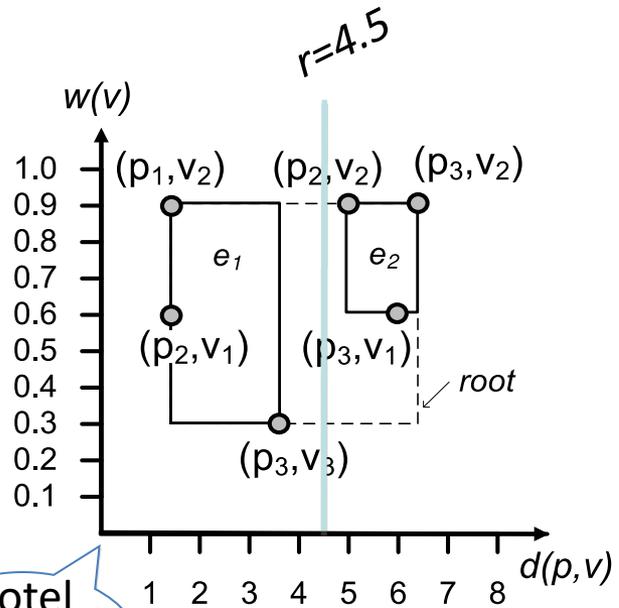
Beispiel (range, $r=4.5$)



hotel
x
restaurant



$p_3(0.8)$



hotel
x
bar



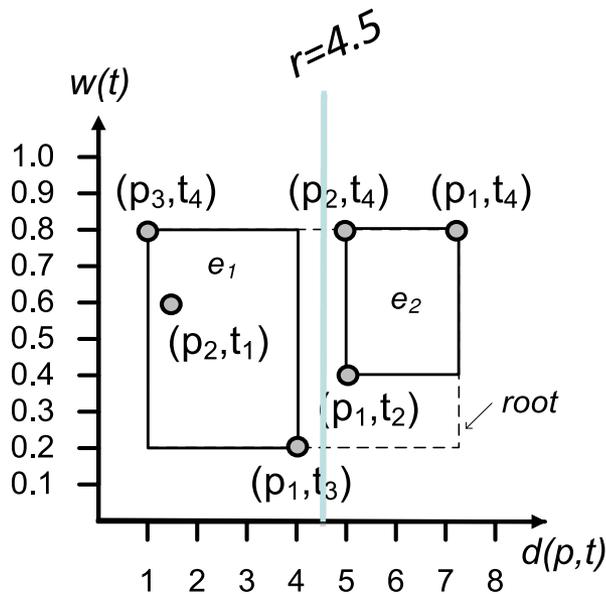
$p_1(0.9)$

+

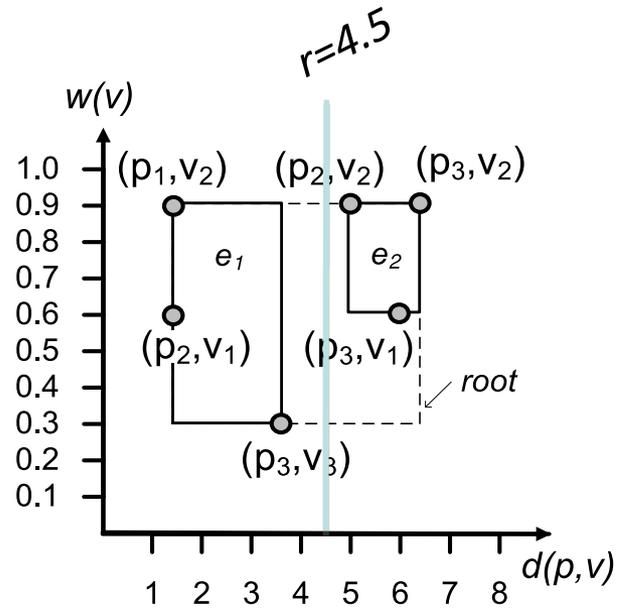
= 1.7

Object	R_1	R_2	Score	Upper-bound
p_3	0.8	-	0.8	1.7
p_1	-	0.9	0.9	1.7

Beispiel (range, $r=4.5$)



$p_2(0.6)$



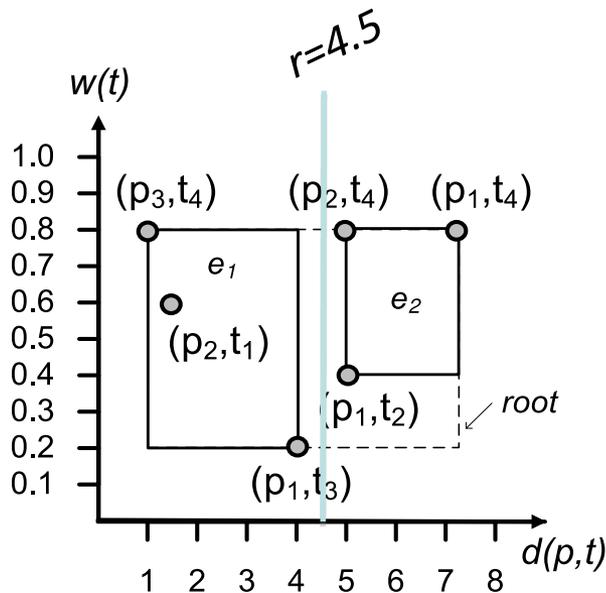
$p_2(0.6)$

+

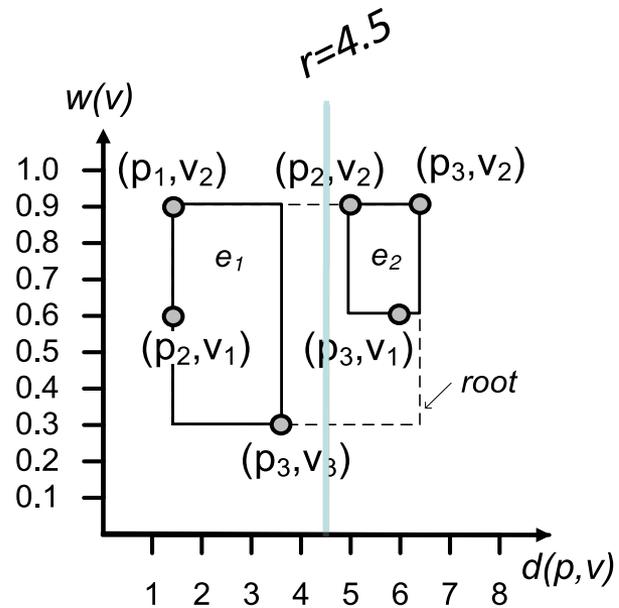
= 1.2

Object	R_1	R_2	Score	Upper-bound
p_3	0.8	-	0.8	1.4
p_1	-	0.9	0.9	1.5
p_2	0.6	0.6	1.2	1.2

Beispiel (range, $r=4.5$)



$p_1(0.2)$



$p_3(0.3)$

+

= 0.5

Object	R_1	R_2	Score	Upper-bound
p_3	0.8	0.3	1.1	1.1
p_1	0.2	0.9	1.1	1.1
p_2	0.6	0.6	1.2	1.2

Top-1

Top-k-Anfrage-Verarbeitung

- Berechne top-k Datenobjekte fortschreitend, indem partielle Maße aggregiert werden, die von R_i stammen
 - Ähnlich wie bei Fagins-Algorithmus
- Algorithmus
 - Wenn ein Objekt p aus R_i geladen wird, hat jedes nicht betrachtete Object p' in R_i ein Maß $\text{score}(p') \leq \text{score}(p)$
 - Verwaltung untere und obere Rangmaß-Grenze für bislang nicht gesehene Objekte
 - Terminierung, wenn untere Grenze (Score) des k -ten Objekts besser als obere Grenze (Upper Bound) der übrigen Objekte

Non-Standard-Datenbanken und Data Mining

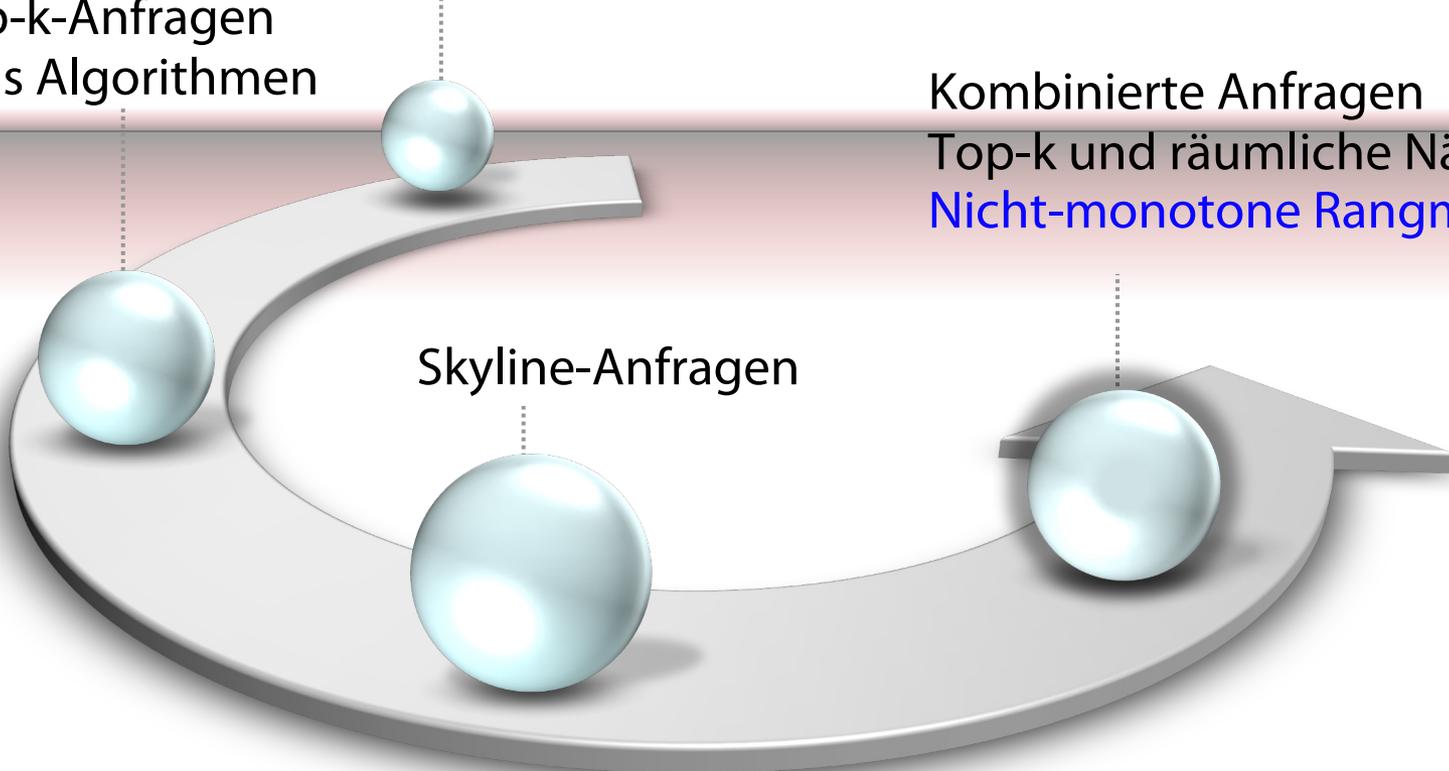
Von First-n- und Top-k-Anfragen zu Skyline-Anfrage

Top-k-Anfragen
Fagins Algorithmen

Anwendungen, First-n-Anfragen, Optimierung

Kombinierte Anfragen
Top-k und räumliche Nähe
Nicht-monotone Rangmaße

Skyline-Anfragen



Monotonie der Bewertungsfunktion

- BBS benötigt eine monotone Bewertungsfunktion
 - Monoton steigend oder monoton fallend
- Verwendet folgende Anfrage eine monotone Bewertungsfunktion?

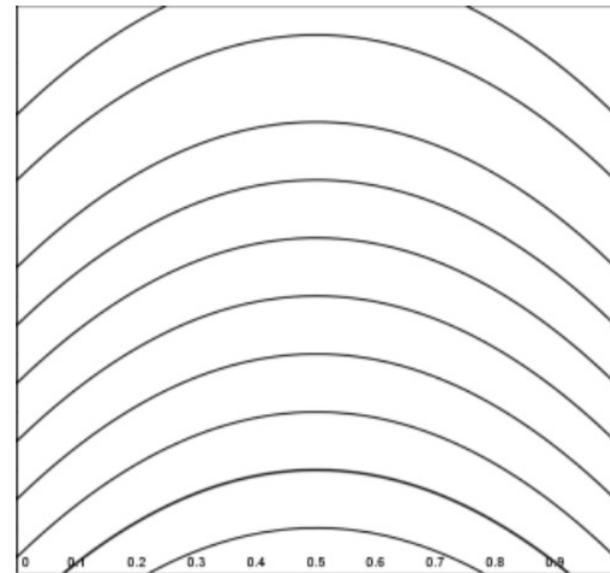
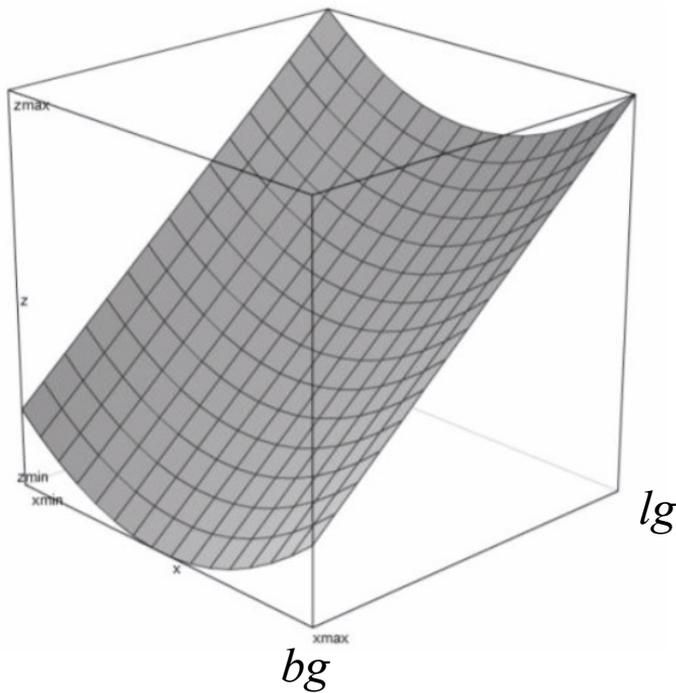
```
SELECT Top 3 *  
FROM Immobilien  
WHERE PLZ BETWEEN 80331 AND 81929  
ORDER BY (BG - 0.5)2 + LG DESC
```

- Bewertungsfunktion, die maximiert werden soll:

$$f(t) = (bg(t) - 0,5)^2 + lg(t)$$

- Algorithmus **Branch-and-Bound-Ranked-Search**

Graph und Höhenlinie von f



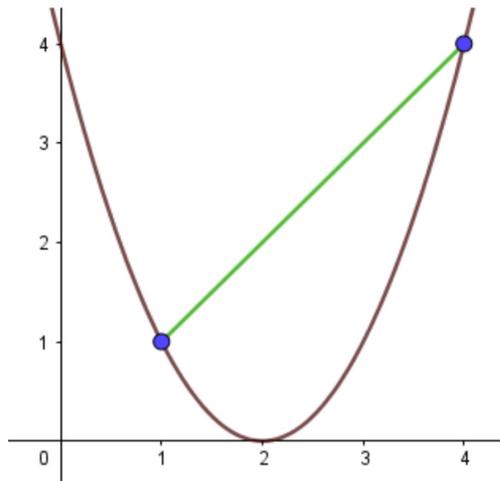
Konvexe und Quasi-Konvexe Funktionen

Eine reellwertige, multivariate Funktion $f: C \rightarrow \mathbb{R}$ auf einer konvexen Teilmenge $C \subseteq \mathbb{R}^n$ heißt *konvex*, wenn für je zwei Punkte $x, y \in C$ und alle $\lambda \in [0,1]$ gilt, dass

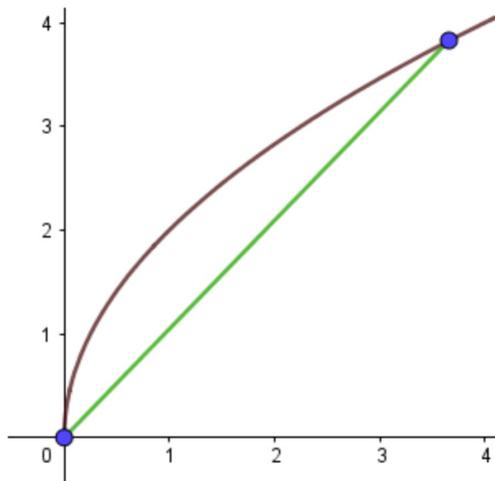
$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

Sie heißt *quasi-konvex*, wenn für jedes $x, y \in C$ und alle $\lambda \in [0,1]$ gilt, dass

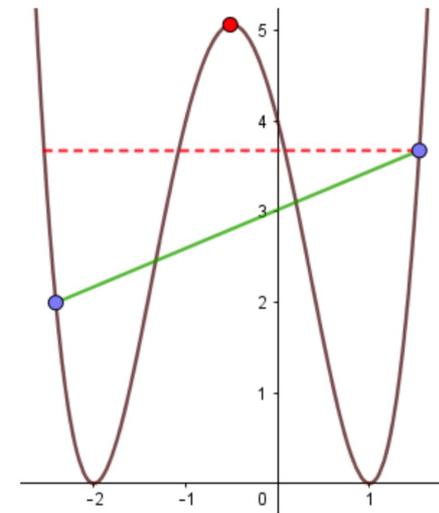
$$f(\lambda x + (1 - \lambda)y) \leq \max\{f(x), f(y)\}.$$



Konvexe Funktion



Quasi-konvexe Funktion



Nicht-konvexe Funktion

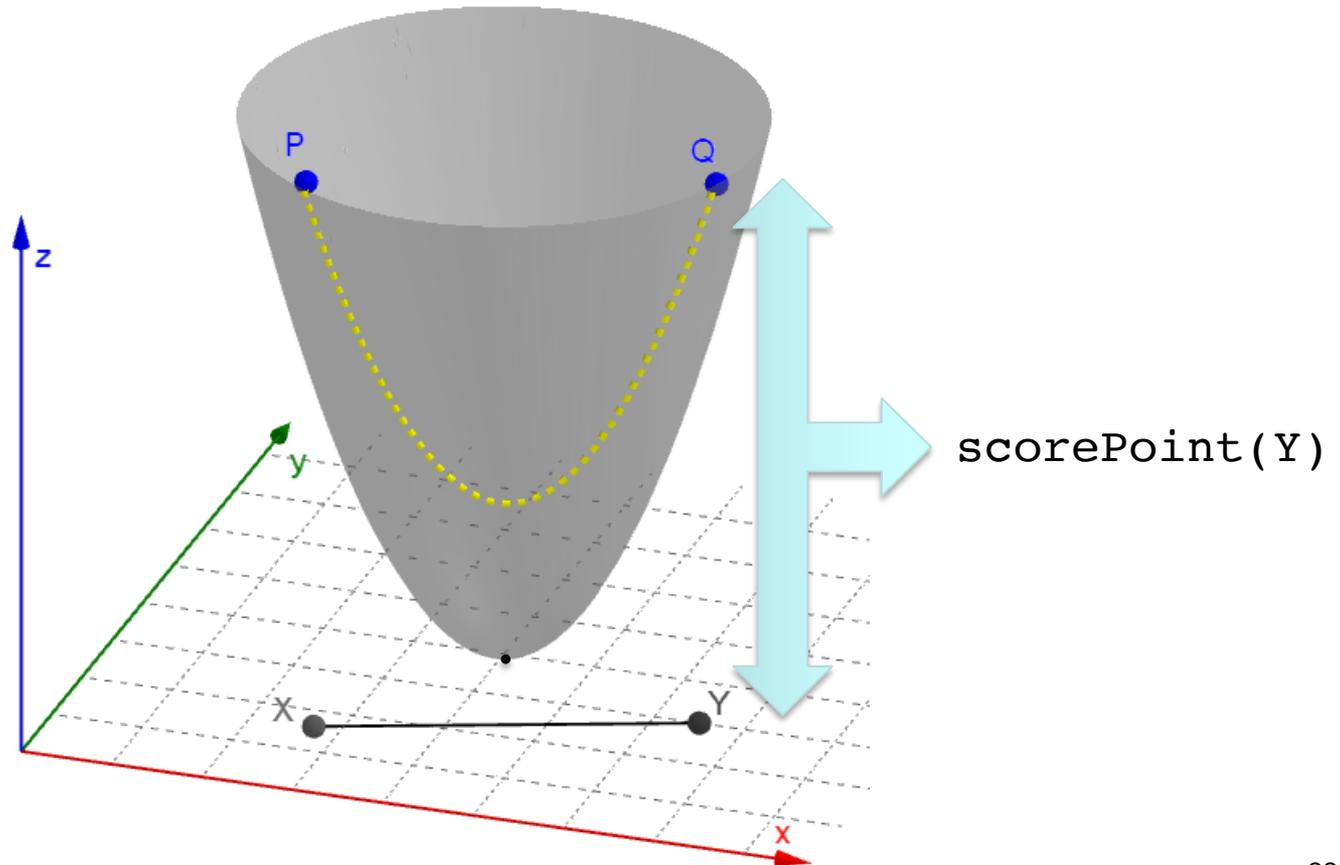
Branch-and-Bound-Ranked-Search (BRS)

```
algorithm BRS (rt, k, type, score_r, score_p)
// rt is an r-tree on the dataset
// k denotes the number of data points to return
// type can be either 'min' or 'max',
// score_r computes a score for an MBR
// score_p is a point scoring function
let pq = build_priority_queue(type, // pq with (obj, score) entries
    map(lambda(obj) return (obj, score_r(mbr(obj), score_p)),
        root(rt)))

    result = {}
    n = 0
    object
while n < k and not empty?(pq) do
    object := delete_next(pq)
    if point?(object) then
        result := result U {object}
        n := n + 1
    else
        if leaf?(object) then
            for p ∈ points(mbr(object)) do
                insert((p, score_p(p)), pq)
        else
            for e ∈ children(object) do
                insert((e, score_r(e, score_p)), pq)
return result
```

ScorePoint

```
SELECT Top 3 *  
FROM Immobilien  
WHERE Typ = 'Apartment' AND Standort = 'Berlin'  
ORDER BY (Wohnraum - 0,4)2 + (Kaufpreis - 0,5)2 DESC
```



Maximumprinzip Quasi-Konvexer Funktionen

Es sei $f: H \rightarrow \mathbb{R}$ eine quasi-konvexe Bewertungsfunktion, $H \subset \mathbb{R}^n$ ein beliebig gegebenes Hyperrechteck (MBR) mit seinen 2^n Ecken e_1, \dots, e_{2^n} , dann gilt:

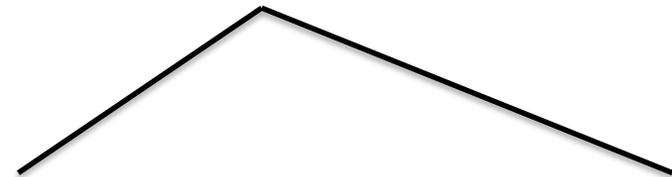
$$f_{\max}(H) = \max\{f(e_1), \dots, f(e_{2^n})\}.$$

Anders formuliert bedeutet diese Aussage, dass eine quasi-konvexe Funktion auf einem Hyperrechteck ihr Maximum immer auf einer, oder auf mehreren Ecken annimmt.

Damit können wir BRS wie folgt aufrufen:

```
let query_tuple_score =  
  lambda (t)  
    return (normalize(wohnraum(t)) - 0.4)2 + (normalize(kaufpreis(t)) - 0.5)2  
BRS(data_rtree, k, 'max',  
  lambda (mbr)  
    return reduce(max, map(query_tuple_score, vertices(mbr)), 0)  
  query_tuple_score)
```

Referenzpunkt $q = (0.4, 0.5)$



Vertices liefert die Eckpunkte eines MBR, normalize bildet Werte in Einheitswürfel ab.

Minimumprinzip?

- Auch beim Minimum nur die Eckpunkte betrachten?
- Leider so für quasi-konvexe Funktionen nicht anwendbar
- Finden wir ein entsprechendes Minimumprinzip?
- Für welche Funktionsklassen?

SELECT *Top k* *

FROM *R*

[WHERE...]

ORDER BY $\omega_1(a_1 - q_1)^2 + \omega_2(a_2 - q_2)^2$ ASC

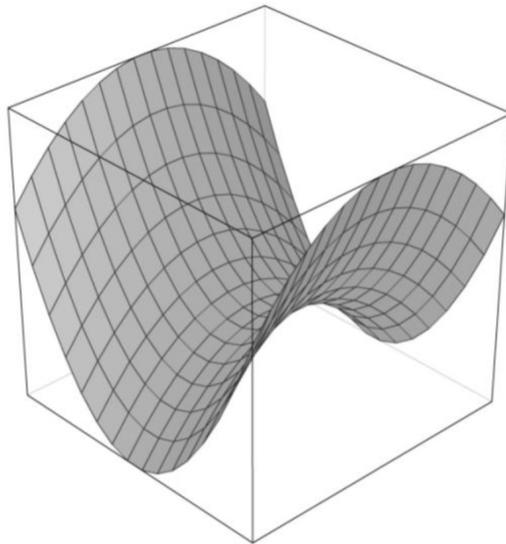
- Quadratische Funktionen
(genauer: Funktionen mit geradem Exponenten)

Quadratische Funktionen

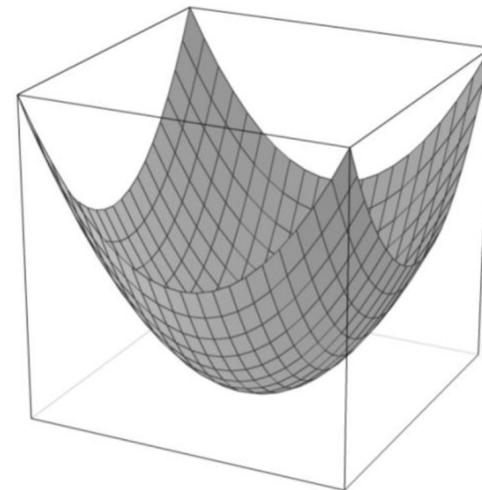
- Eine Abbildung $f: D \rightarrow \mathbb{R}, D \subseteq \mathbb{R}^n$ der Form

$$(x_1, \dots, x_n) \mapsto \sum_{i=1}^n \omega_i (x_i - q_i)^2$$

mit $\omega_i \in \mathbb{R}$ für jedes $i \in \{1, \dots, n\}$ heißt multivariate *quadratische* Funktion.

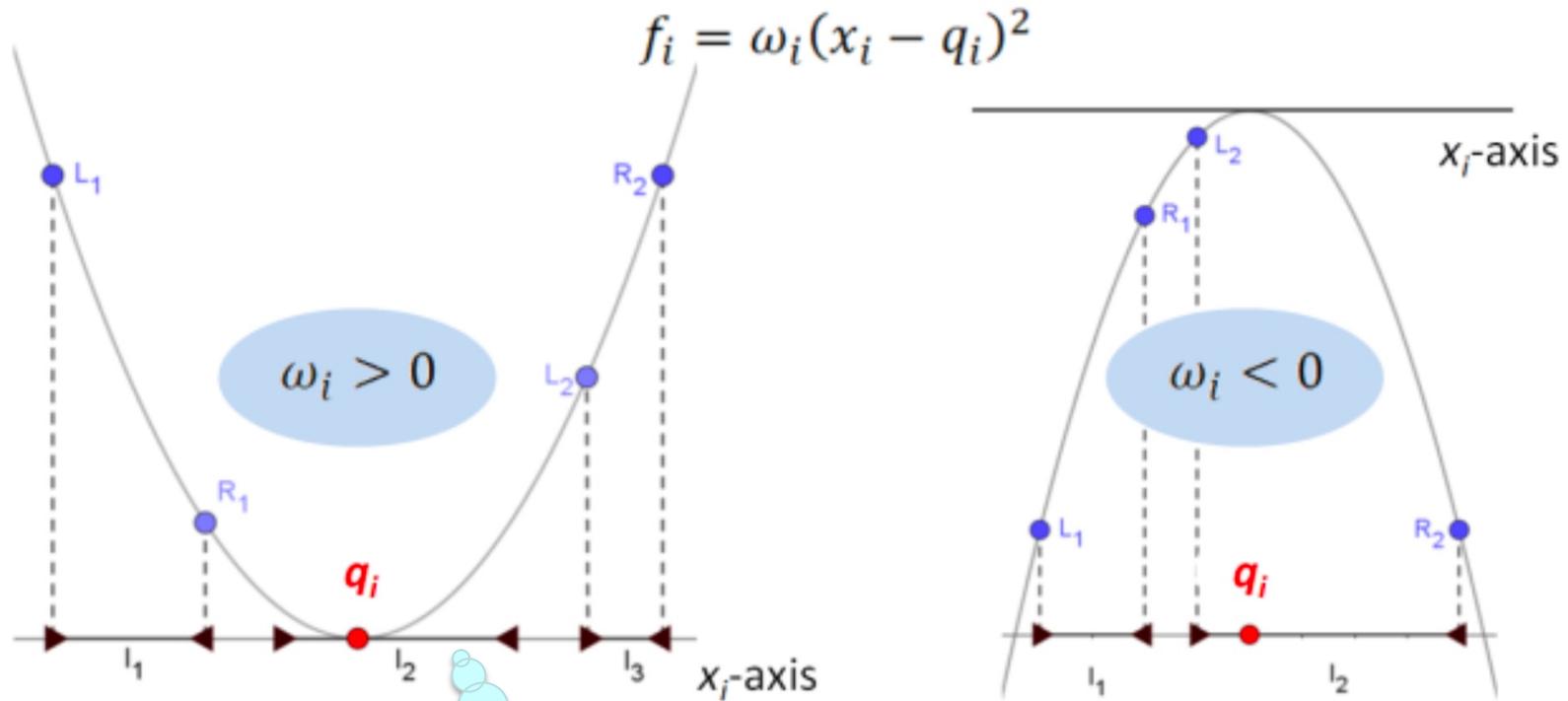


Hyperbolisches Paraboloid (ein $\omega_i < 0$)



Elliptisches Paraboloid (alle $\omega_i > 0$)

Rückführung auf eindimensionale Paraboloide



Mögliche Lage
der Seiten eines
MBR

Unterscheidung der Fälle

- Falls ω_i aus f positiv (linke Seite):
 - Minimaler Wert von f ist 0 gdw. das Intervall einer MBR-Seite die Koordinate des globalen Minimums enthält (I_2)
 - Sonst, wenn die Koordinate des globalen Minimums außerhalb ist (I_1 or I_3), ist das Minimum der kleinste Funktionswert der linken (I_1) oder rechten (I_3) Intervallgrenze
- Falls ω_i aus f negativ (rechte Seite):
 - Minimum ist immer der kleinste Funktionswert der linken oder rechten Intervallgrenze (I_1 oder I_2)

Zusammenfassung

- Branch-and-Bound-Skyline
- Top-k-Präferenzanfragen für räumliche Objekte
- Monotonie der Bewertungsfunktion
- Branch-and-Bound Ranked Search
 - Ausnutzung von R-Bäumen
 - Auch für bestimmte Klassen von nicht-monotonen Bewertungsfunktionen
 - Quasi-konvexe Bewertungsfunktionen: Max
 - Quadratische Funktionen (Paraboloide): Min