2.6 Skylineanfrage

- Gegeben:

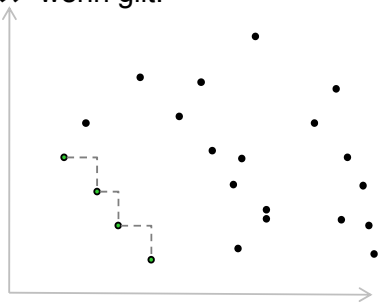
  Menge von Objekten mit positiven Attributwerten $a_i \in \Re^+$ ($1 \le i \le d$).
- Gesucht:

  Ergebnis enthält alle Objekte die von keinem anderen Objekt *dominiert* werden, d.h. kein anderes Objekt ist „besser“ bzgl. aller Attribute.
- Formal:

  Vektor $x=(x_1,..,x_d) \in \Re^d$ dominiert Vektor $y=(y_1,..,y_d) \in \Re^d$ wenn gilt:

  $dom(x,y) \Leftrightarrow (\forall\ 1 \le i \le d\ x_i \le y_i) \wedge (\exists\ 1 \le i \le d\ x_i < y_i)$

  SkyLine(*DB*) = $\{o \in DB \mid \forall p \in DB : \neg dom(p,o)\}$



Skyline Anfrage

- Basisalgorithmus (sequential scan):

```
Skyline-SeqScan(DB)
    result = ∅;
    FOR i=1 TO n DO
        o = getObject(i);
        IF o not-dominated-by(any other object in DB)THEN
            result := result ∪ o;
    RETURN result;
```

zusätzlicher sequentieller scan über DB

- Skyline-Anfrage Varianten: [Papadias et al., ToDS 2005]

  zum Beispiel:
  - » *Constrained Skyline*:

    Ausgabe aller Skyline-Objekte deren Attribute zusätzlich eine bestimmte Bedingung (z.B. Hotelkosten zwischen 50 und 80 Euro) erfüllen.
  - » *Ranked Skyline*:

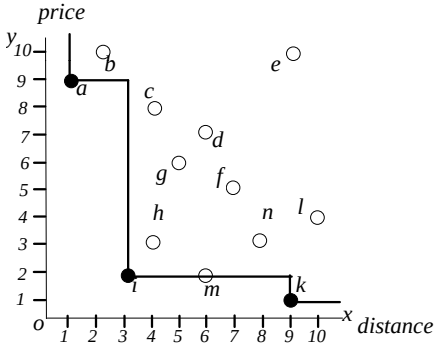
    Ausgabe der *k* ersten Skyline-Objekte, die bzgl. einer Präferenzfunktion (Kostenfunktion über alle Attribute) sortiert ausgegeben werden.
  - » weitere Varianten:

    Group-by Skyline, *Dynamic Skyline*, *K-dominating Queries*, etc.

2.6.2 Indexbasierte Skyline-Anfrage

– Anforderungen:

- Gegeben:
  - Menge von d-dimensionalen Punkte (Objekte)
  - Indexierung mittels R-Baum



- Gesucht:
  - Alle Objekte, die von keinem anderen Objekt dominiert werden.
- Ziele:
  - Wenig Seitenzugriffe
  - Wenig Dominanzüberprüfungen (Objektvergleiche)
  - Möglichst früh erste Ergebnisse ausgeben

- Grundsätzlich viele unterschiedliche Ansätze
  - Hauptspeicher-basiert ←→Sekundärspeicher-basiert
  - Iterative Berechnung ←→ Nicht-Iterative Berechnung

  Skyline-Anfrage Varianten:
  - Mit explizitem Anfrageobjekt(en) (dynamische Skyline)
  - zusätzliche Bedingungen
  - andere Skylinevarianten: z.B: Top-k-Dominanz, etc.

- Bekannteste Ansätze die auf Sekundärspeicher beruhen:

  (Zusammenfassung aus [Papadias et al., ToDS 2005])
  - Divide-and-Conquer, Block-Nested Loop [Borzsonyi et al., 2001]
  - Sort First Skyline [Chomicki et al., 2003]
  - Bitmap, Index [Tan et al., 2001]
  - Nearest-Neighbor [Kossmann et al., 2002][Papadias et al., ToDS 2005]

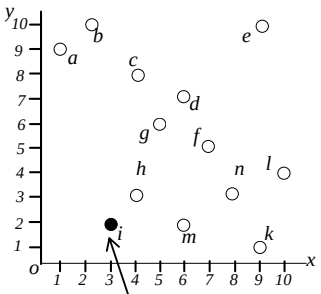
    Eigenschaften:
    - » Sekundärspeicherbasiert
    - » Erfüllen alle drei Ziele:
      - wenig Seitenzugriffe und Dominanzüberprüfung mittels Index (R-Baum).
      - Erste Ergebnisse werden frühzeitig ausgegeben durch iterative Verarbeitung.

– Nächste-Nachbarn-Skyline (NNS) Algorithmus:

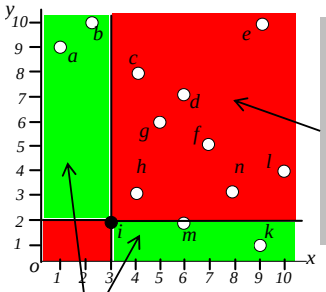
[Kossman et al., VLDB 2002]

Prinzip:

- Benutzt Nächste-Nachbarn-Suche zur (rekursiven) Partitionierung des Suchraums



Nächster Nachbar
(des Koordinaten-Ursprungs)
→ erstes Skyline-Ergebnis

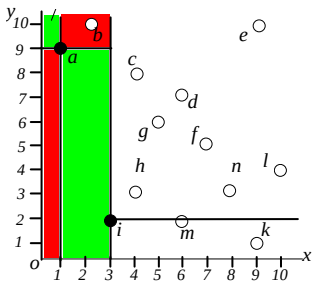


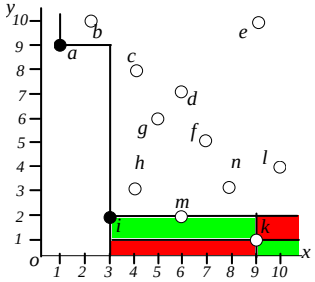
Raumpartition mit Objekten die von Objekt i dominiert werden
=> Objekte gehören nicht zum Ergebnis (*true drops*).

Raumpartitionen mit weiteren Skyline-Kandidaten

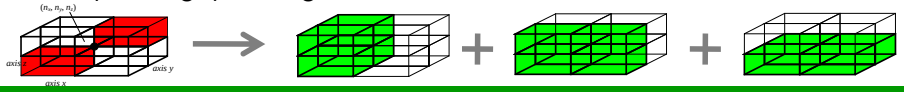
- Nächste-Nachbar-Suche kann durch R-Baum Index beschleunigt werden (z.B. Alg.: k-NN-Index-HS).

- Nächste-Nachbar-Suche wird zur weiteren Partitionierung **in jeder** Kandidaten-Suchregion rekursiv fortgesetzt.





- Vorteile:
  - Verwendung von effizienten Methoden zur NN-Suche.
  - Erste (relevante) Resultate können schnell ausgegeben werden.
- Nachteile:
  - Im d-dimensionalen Raum führt jedes gefundene Skyline-Objekt (Punkt) zu d weiteren Fensteranfragen.
    → viele redundante Anfragen → Duplikateliminierung
    → viele (unnötige) Anfragen auf leeren Raum



– Branch-and-Bound Skyline (BBS) Algorithm:

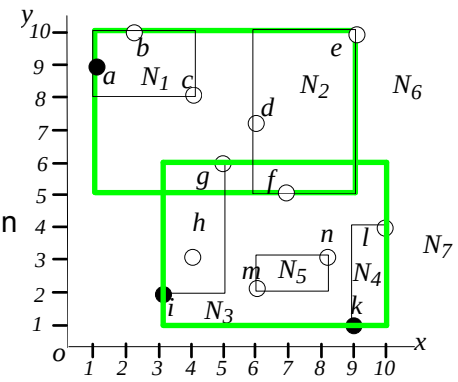
[Papadias et al., ToDS 2005]

Prinzip:

- Idee: Datenorientierte Suche statt Datenraumorientierte Suche
  - Vermeidung von Suche in “leeren” Datenraumpartitionen.
  - Kandidaten werden direkt über einen Index (R-Baum) ermittelt.
- Prioritäts-basierte Suche des nächsten Skyline-Objektes
  - Priorität entsprechend Manhattan-Distanz zum Koordinaten-Ursprung
  - Iterative Verfeinerung des Index (R-Baum) mittels Prioritätsliste (vgl. k-NN-Index-HS, Folie 63)

→ Verwendung eines Heaps aufsteigend sortiert über MINDIST(e,(0,0)), e := Seitenregion oder Objekt (Punkt)

→ Verwendung einer Liste mit bereits gefundenen Skylineobjekten zum Prunen von anderen Seitenregionen / Objekten

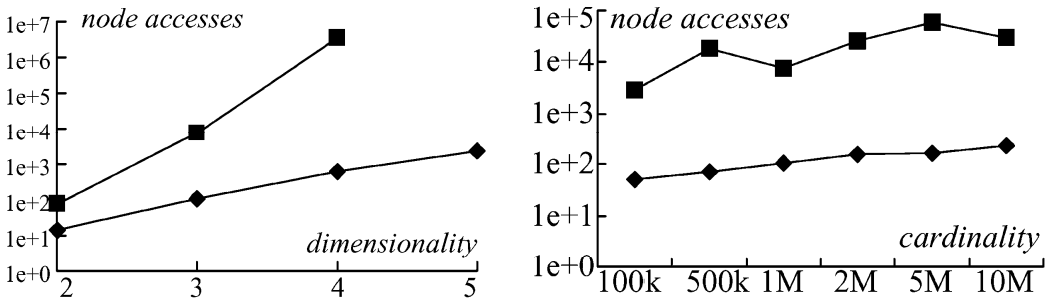


- Algorithmus:

Algorithm BBS (R-tree R)
S=Ø
Füge alle Einträge der Wurzel R in den Heap ein
Solange Heap nicht leer:
  entferne ersten Eintrag e
  wenn e von einem Punkt in S dominiert wird, verwerfe e
  sonst (e ist nicht dominiert)
    wenn e kein Datenpunkt ist
      für jedes Kind e_i von e
        falls e_i nicht von einem Punkt in S dominiert wird, füge e_i in den Heap ein
    sonst (e ist ein Datenpunkt)
        füge e in S ein

- Vorteile:
  - Vorzeitige Ausgabe von ersten Resultaten
  - Keine unnötige Partitionierung des Datenraums → geeignet auch für Suchraumdimensionen > 3 (im Gegensatz zu NNS)
  - BBS ist optimal bzgl. der Seitenzugriffe im R-Baum (I/O-optimal)

- Experimenteller Vergleich: NNS ←→ BBS
  - Datensatz: 1 Mio. Objekte gleichmäßig verteilt



  - Fazit: BBS schlägt NNS bzgl. I/O-Kosten über mehrere Größenordnungen

2.6 Bewertung von Methoden zur Ähnlichkeitssuche

– Fragestellung
- Anfragebearbeitung in metrischen Räumen oder Vektorräumen
- Gesucht: Feature-Transformation zur Umwandlung komplexer STMM-Objekten in metrische Objekte/Featurevektoren
- Wie gut drückt die Feature-Transformation die Ähnlichkeit der realen Objekte aus, d.h. wie gut approximiert die Distanz im Feature-Raum die Distanz im Objektraum?
- Bewertung von Methoden zur Ähnlichkeitssuche („Ähnlichkeitsmodelle“)
  - Testset von Objekten
  - Stelle für alle Objekte des Testsets Ähnlichkeitsanfragen (typischerweise *k*-NN-Queries)
  - Evaluiere das Ergebnis dieser Anfragen

– Objekte mit bekannten Kategorien
  - Objekte sind in Kategorien eingeteilt und entsprechend markiert (z.B. „Schrauben“, „Nägel“, „Bolzen“, …), d.h. Ergebnis der Anfragen ist bekannt
  - Übersicht

	erwünscht	**unerwünscht**
gefunden	richtig positive (rp)	falsch positive (fp)
nicht gefunden	falsch negative (fn)	richtig negative (rn)

  - Recall (Sensitivität): Wie viele der erwünschten Objekte wurden gefunden?

$$\frac{rp}{rp+fn} = \frac{\text{gefundene erwünschte Objekte}}{\text{alle erwünschten Objekte}}$$

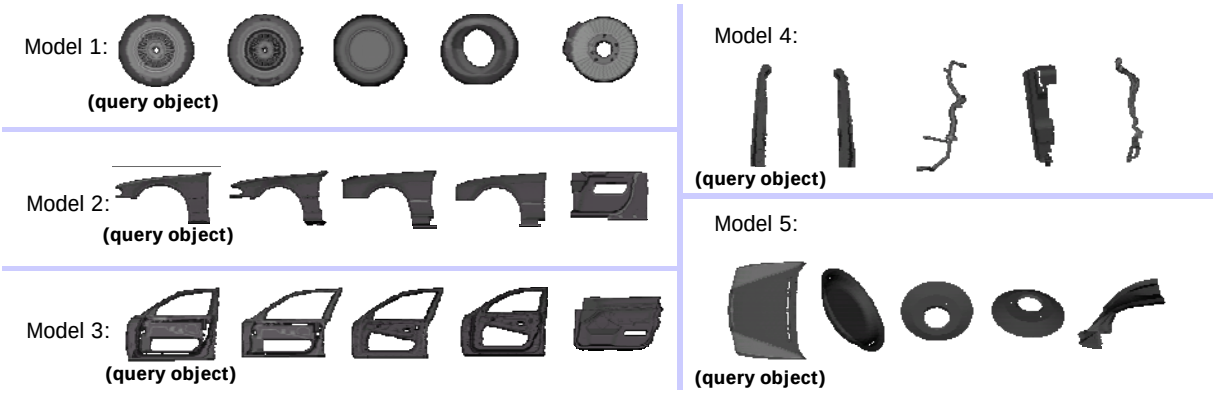
  - Precision: Wie viele der gefundenen Objekte sind erwünscht?

$$\frac{rp}{rp+fp} = \frac{\text{gefundene erwünschte Objekte}}{\text{alle gefundenen Objekte}}$$

  - Spezifität: WS, dass Test für unerwünschtes Obj. negativ verläuft

$$\frac{rn}{rn+fp} = \frac{\text{richtig negativ}}{\text{alle unerwünschten Objekte}}$$

– Objekte mit unbekannten Kategorien
  - Ergebnis der Anfragen ist unbekannt
  - Manuelle Evaluation weniger zufälligen *k*-NN-Queries
  - Problem: Qualität des Modells hängt ab von
    – einer geringen Anzahl von Query-Objekten
      » Besser: möglichst alle Objekte der DB spielen eine Rolle bei der Evaluation
    – der Wahl dieser Query-Objekte
      » Schlechtes Anfrageergebnis für gegebenes *q* bedingt nicht schlechtes Modell
      » Gutes Anfrageergebnis für gegebenes *q* bedingt nicht gutes Modell



- BOSS (Browsing OPTICS-plots for Similarity Search)

  [Brecheisen, Kriegel, Kröger, Pfeifle. Proc. SIAM Int. Conf. Data Mining (SDM), 2004]

  – Idee: benutze Data Mining Methoden
  – Clustering
    » Fasse Objekte in Gruppen zusammen, sodass die Objekte in einer Gruppe (Cluster) ähnlich, Objekte aus verschiedenen Clustern unähnlich sind
    » Hierarchisches Clustering: erstelle eine Hierarchie von ähnlichen Objekten
  – Clustererkennung und Clusterrepräsentation
    » Erkenne automatisch geeignete Cluster in der Hierarchie
    » Stelle jeden Cluster durch einen geeigneten Repräsentanten dar
  – Evaluation/Retrieval
    » Hierarchie von Clusterrepräsentanten ist navigierbar
    » Evaluation der Cluster um Ähnlichkeitsmodell zu evaluieren
    » Ähnlichkeits-basierte Suche nach Objekten ohne konkretes Anfrageobjekt angeben zu müssen