2.5 Reverse nächste Nachbarn Anfragen

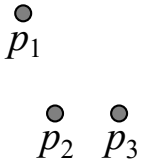
2.5 Reverse Nächste Nachbarn Anfragen

2.5.1 Allgemeines

- Eigenschaften
  - Benutzer gibt Anfrageobjekt *q* vor
  - Ergebnis enthält alle Objekte, die *q* als nächsten Nachbarn haben
  - Analog: Reverse *k*-nächste Nachbarn
  - Mehrdeutigkeiten (bei NN) entsprechend behandeln
- Formal
  - Reverse nächste Nachbarn $RNN(q) = \{o \in DB \mid q \in NN(o)\}$
  - Reverse *k*-nächste Nachbarn $RNN(q,k) = \{o \in DB \mid q \in NN(o,k)\}$
- Anwendungsbeispiel:
  - Standortsuche für neue Filiale (welche Kunden haben die neue Filiale als „nächsten Nachbarn“)

2.5 Reverse nächste Nachbarn Anfragen

- Zusammenhang zwischen NN und RNN
  - NN ist keine symmetrische Relation
    - $y \in NN(x) \not\Leftrightarrow x \in NN(y)$
    - $y \in NN(x) \not\Rightarrow y \in RNN(x)$
  - RNN ist ein „eigenständiges Problem



	NN	RNN
p_1	$\{p_2\}$	$\{\}$
p_2	$\{p_3\}$	$\{p_3, p_1\}$
p_3	$\{p_2\}$	$\{p_2\}$

- Basisalgorithmus (sequential scan): nichtdeterministisch

```
RNN-SeqScan(DB, q, k)
    resultSet = ∅;
  FOR i=1 TO n DO
    neighbors = NN-SequScan(DB, DB.getObject(i), k);
      IF q ∈ neighbors THEN
        resultSet.add(DB.getObject(i));
  RETURN resultSet;
```

  - Offensichtliche Verbesserung
    - Statt NN-SequScan einen besseren NN-Algorithmus verwenden

2.5 Reverse nächste Nachbarn Anfragen

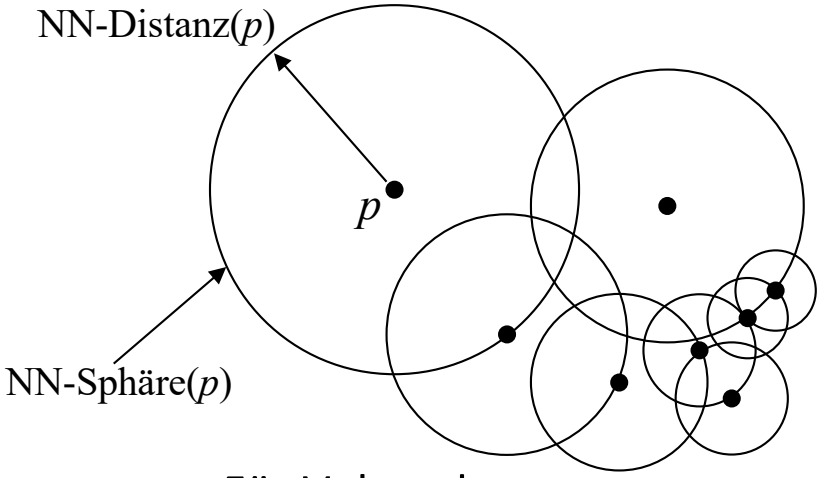
- Index-basierte Methoden für die RkNN-Suche
  - Annahme:
    - Daten/Objekte in einer Baumartigen Indexstruktur, z.B. R-tree, organisiert
    - Suche erfordert hierarchischen Durchlauf der Directory-Seiten
  - Ziel
    - Bei der Suche möglichst früh Seiten auf höheren Index-Level ausschließen (d.h. effektive Pruning-Strategien)

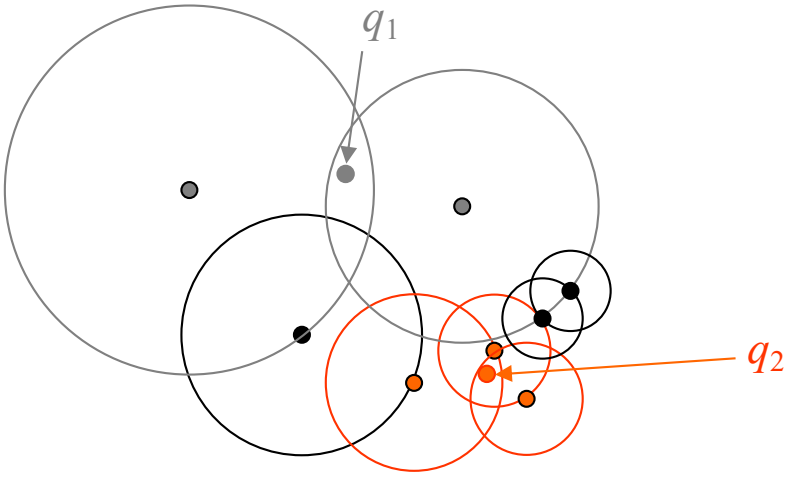
      ==> möglichst starke Einschränkung des Suchraums
  - Pruning-Strategien
    - Generell gibt es zwei Index-basierte Pruning-Strategien für die RkNN-Suche
    - Self-Pruning Strategien:
      - Punkte/Seiten schließen sich selbst aus
      - Basieren auf k-NN-dist-Abschätzungen angewandt auf Punkte/Seitenregionen
      - Punkte/Seiten, deren k-NN-dist-Bereich den Anfragepunkt nicht enthalten können ausgeschlossen werden
    - Mutual-Pruning Strategien:
      - Punkte/Seiten schließen sich gegenseitig aus
      - Basieren auf Voronoi-Hyperebenen

2.5 Reverse nächste Nachbarn Anfragen

2.5.2 Self-Pruning Strategien

- $p \in \text{RNN}(q) \Leftrightarrow \text{dist}(p, q) \leq \text{NN-Distanz}(p)$
- Materialisiere für alle Objekte die NN-Distanz
- Prüfe, ob $\text{dist}(p, q) \leq \text{NN-Distanz}(p)$ statt während der Anfrage eine NN-Query für alle Objekte zu berechnen

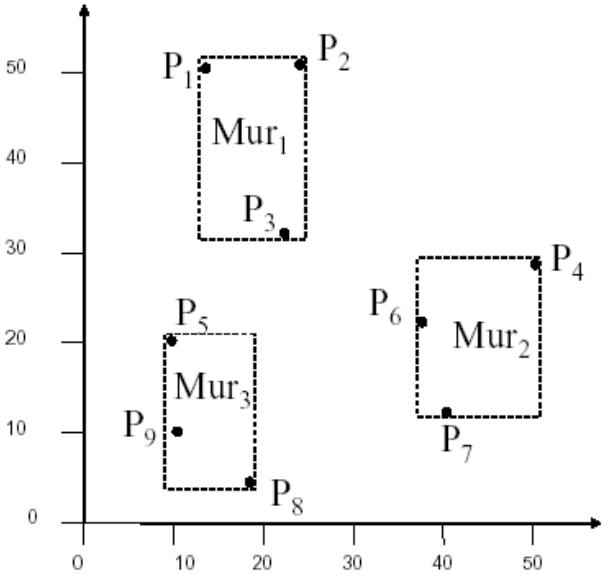




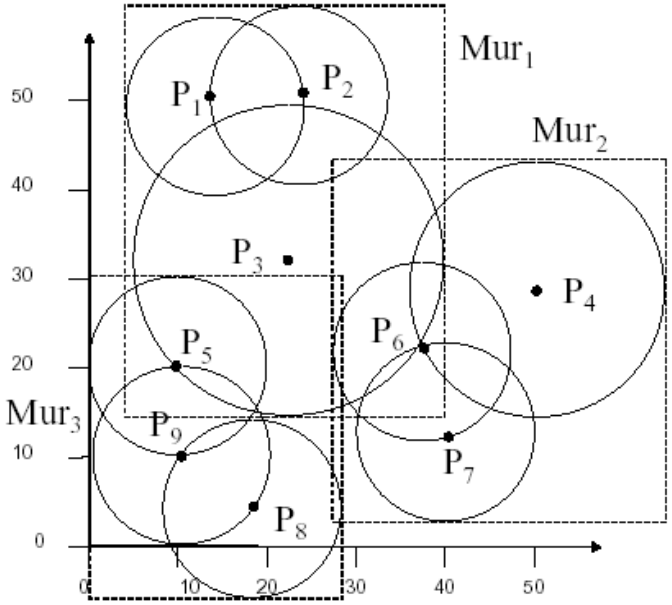
- Für Vektordaten
  - RNN-Query ist Punktanfrage bzgl. der NN-Sphären der Punkte (vgl. Voronoi-Ansatz zur NN-Query)
  - Speichere NN-Sphären in einem Index für ausgedehnte Objekte (z.B. R-Baum)

2.5 Reverse nächste Nachbarn Anfragen

- RNN-Baum [Korn, Muthukrishnan. ACM Int. Conf. Management of Data (SIGMOD), 2000]
  - RNN-Queries für Vektordaten
  - Berechne NN-Distanz für alle DB-Punkte
  - Speichere statt Punkte alle NN-Sphären der Punkte in R-Baum



Normaler R-Baum



RNN-Baum

2.5 Reverse nächste Nachbarn Anfragen

- Algorithmus zur NN-Suche
  - Datenseiten enthalten Kreise, d.h. Objekte der Form (Punkt, Radius)
    - Punkt = DB-Objekt (Mittelpunkt)
    - Radius = NN-Distanz(Punkt)

```
RNN-Tree-Search(pa, q,)                    // pa = Diskadress z.B. der Wurzel des Indexes
    result = ∅;
    p := pa.loadPage();
    IF p.isDataPage() THEN
        FOR i=0 TO p.size() DO
            IF dist(q, p.getObject(i).Point) ≤ p.getObject(i).Radius THEN
                result := result ∪ getObject(i).Point;
        ELSE                                // p ist Directoryseite
        FOR i=0 TO p.size() DO
            IF MINDIST(q, p.getRegion(i)) = 0 THEN
                result := result ∪ RNN-Tree-Search(p.childPage(i), q);
        RETURN result;
```

2.5 Reverse nächste Nachbarn Anfragen

- Vorteil
  - Sehr gute Performanz (Pruning-Power) bei RNN-Anfragen
- Nachteile
  - *k* muss fest vorgegeben sein
  - Nur für Vektordaten
  - Hohe Überlappungen der Seitenregionen im Directory führt zu schlechter Performanz bei normalen NN-Anfragen

    Lösung: speichere einen RNN-Baum und einen konventionellen R-Baum
  - Schlechte Performanz bei Einfügungen und Löschungen

    Beispiel: *Einfügen* von *p*

„Normaler“ Index
- Bestimme NN(*p*) und füge Kreis (*p*, NN-Distanz(*p*)) in Index ein

Zusätzlich für RNN-Baum
- Bestimme RNN(*p*)
- Erneuere NN-Sphären aller $o \in$ RNN(*p*)
- Erneuere Seitenregionen (von *p* und allen *o*) betroffener Datenseiten
- Erneuere rekursiv Seitenregionen der Vaterseiten

- Varianten:
  - Voronoi-Zellen statt NN-Sphären
  - Andere Verfeinerungsreihenfolge (siehe NN-Algorithmen)