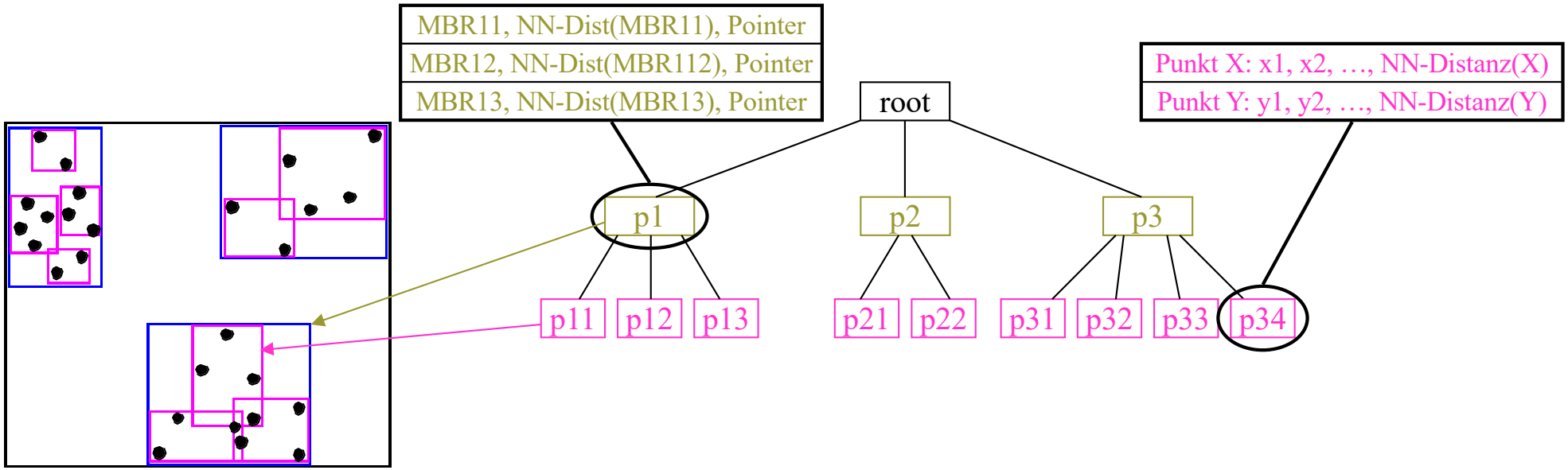
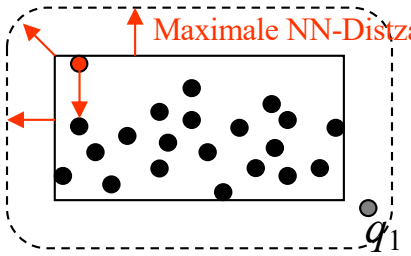
2.5 Reverse nächste Nachbarn Anfragen

- RdNN-Baum [Yang, Lin. IEEE Int. Conf. Data Engineering (ICDE), 2001]
  - Prinzip: Idee des RNN-Baum, ABER
    - Speichere die DB-Objekte statt NN-Sphären
    - Speichere zu jedem Punkt in der DB seine NN-Distanz
      - Zu jedem Punkt zusätzlich einen Wert abspeichern
      - Zu jeder Seitenregion *s* zusätzlich noch den Wert $\max_{child \in s} child.getNNDist()$ abspeichern (Maximum aller NN-Distanzen in den Kinderseiten)



2.5 Reverse nächste Nachbarn Anfragen

- Ausschluss von Directory-Seiten, wenn MINDIST(q,Seitenregion) größer ist, als die aggregierte NN-Distanz der Seitenregion



Query q_2: Seite kann ausgeschlossen werden; kein Punkt der Seite kann q_2 als NN haben

Query q_1: Seite kann nicht ausgeschlossen werden

- Algorithmus zur RNN-Suche

```
RdNN-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)) ≤ p.getNNDist(i) THEN
                result := result ∪ getObject(i);

        ELSE                         // p ist Directoryseite
        FOR i=0 TO p.size() DO
            IF MINDIST(q, p.getRegion(i)) ≤ p.getNNDist(i) THEN
                result := result ∪ RdNN-Tree-Search(p.childPage(i), q);
        RETURN result;
```

2.5 Reverse nächste Nachbarn Anfragen

- Auch hier wieder alle möglichen anderen algorithmischen Lösungen zur NN-Suche anwendbar (Prioritätssuche, etc.)
- Vorteil
  - Sehr gute Selektivität, damit gute Performanz bei Anfragen
  - Seitenüberlappung wie bei „normalem“ R-Baum, daher kein extra Index für NN-/ RQ-Anfragen nötig
  - Leicht erweiterbar auf allg. metrische Daten (z.B. M-Tree)
- Nachteil
  - *k* muss fest vorgegeben sein
  - Nur für Vektordaten (aber: Konzept sehr leicht für M-tree erweiterbar)
  - Weiterhin schlechte Performanz bei Einfügungen und Löschungen

2.5 Reverse nächste Nachbarn Anfragen

- MRkNNCoP-Baum (**M**etric **RkNN** w. **Co**nservative A**p**proximation)

  [Achtert, Böhm, Kröger, Kunath, Pryakhin, Renz. ACM Int. Conf. Management of Data (SIGMOD), 2006]

  - Vergleich bisheriger Verfahren
    - RNN-Tree/RdNN-Tree: Vorberechnung der NN-Distanz
      - Wert für *k* ist fix und vorher bekannt
      - Update-Problematik
      - Dafür: erweiterbar auf metrische Daten (z.B. M-Tree)
  - Idee:
    - Benutze die gute Selektivität der vorberechneten NN-Distanzen
    - Berechne für mehrere (am besten alle) Werte für *k* die *k*-NN-Distanzen vor
    - Problem: Speicherung aller Distanzen zu aufwendig
      - Pro Objekt alle *k*-NN-Distanzen
        => Index wäre sehr hoch => hohe Kosten bei der Anfrage
  - Lösung: Approximiere die *k*-NN-Distanzen mit günstigen Approximationen
    - Approximation sollte obere Schranke (UB) der *k*-NN-Distanz sein => true drops, wir können Objekte (Seiten) frühzeitig ausschließen
    - Zusätzliche Approximation als untere Schranke (LB) => true hits

  Bemerkung: Bisher (bei RQ und (k)NN Anfragen) wurden LB-Distanzabschätungen für die Ermittlung von *true drops* verwendet (analog UB-Dist. für *true hits*), was ist hier anders?

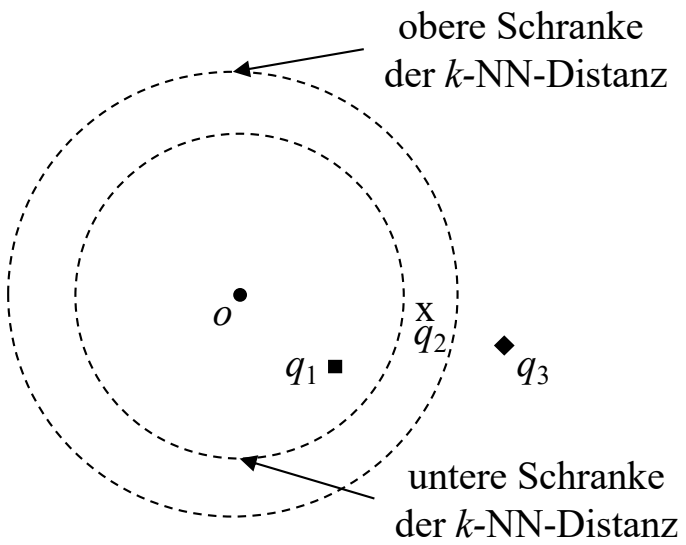
2.5 Reverse nächste Nachbarn Anfragen

- Bewertung von Objekt *o* (analog: Seiten) mit UB- und LB-Approximationen
  - $dist(o,q) \leq LB_{k\text{-NN-Dist}}(o)$

    => *o* true hit, d.h. $o \in RNN(q, k)$
    - Beispiel: $q = q_1$
  - $dist(o,q) \geq UB_{k\text{-NN-Dist}}(o)$

    => *o* true drop, d.h. $o \notin RNN(q, k)$
    - Beispiel: $q = q_3$
  - $UB_{k\text{-NN-Dist}}(o) \leq dist(o,q) \leq LB_{k\text{-NN-Dist}}(o)$

    => *o* Kandidat
    - Beispiel: $q = q_2$



- Gegeben: für jedes Objekt *o* eine Sequenz der *k*-NN-Distanzen, <1-NN-Dist(*o*), 2-NN-Dist(*o*), …, k_{max}-NN-Dist(*o*)> für ein hinreichend großes k_{max}
- Frage: wie kann ich UB- und LB-Approximation dieser *k*-NN-Distanzen berechnen und kompakt speichern?

2.5 Reverse nächste Nachbarn Anfragen

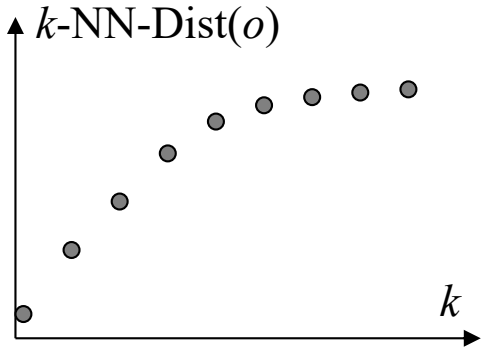
- Lösung aus der Theorie der Selbstähnlichkeit
  - Potentzgesetz gilt für Verhältnis zwischen
    - dem Radius einer Hyperkugel $encl(\varepsilon) \propto \varepsilon^{d_f}$
    - der Anzahl an Objekten innerhalb der Hyperkugel

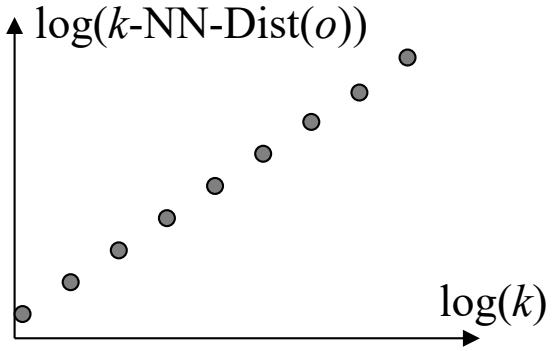
      wobei *encl*(ε) = #Objekte innerhalb der Kugel

      d_f = „Fraktale Dimension“
  - Übertragung auf k-NN-Sphäre:
    - ε = *k*-NN-Distanz
    - *encl*(ε) = *k*

$$\log(k-\mathrm{NN}-\mathrm{Dist}(o)) \propto \frac{\log(k)}{d_f}$$

  - Im log-log-Raum:





2.5 Reverse nächste Nachbarn Anfragen

- In der Realität verhalten sich die Distanzen nicht wie perfekte Linien im log-log-Raum
- Trotzdem: im log-log-Raum können die *k*-NN-Distanzen mit einer Linie approximiert werden
- Das ist erheblich billiger als alle *k*-NN-Distanzen zu speichern, oder andere Funktionen höherer Ordnung zu verwenden, um die Distanzen im normalen *k* / *k*-NN-Dist – Raum zu approximieren

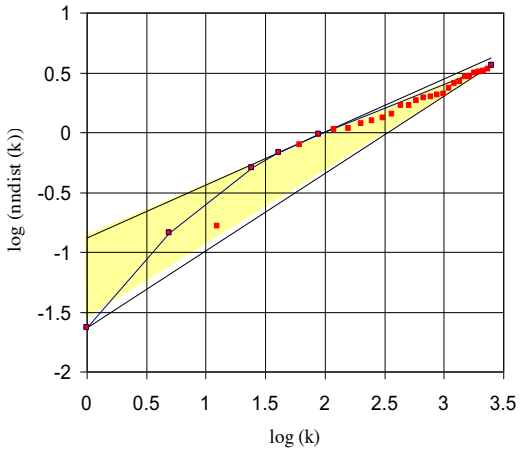
LB- und UB-Approximationen

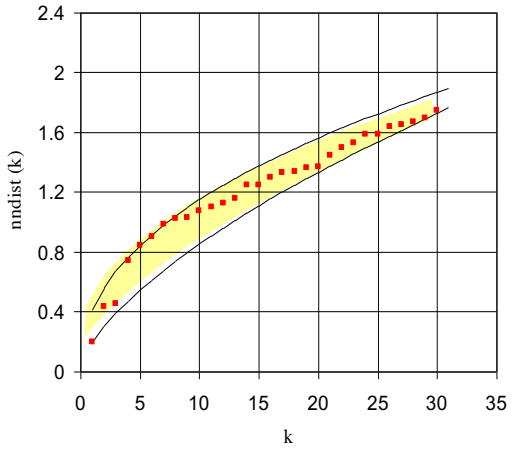
- UB-Approximation ist eine Linie im log(k)-log(dist)-Raum, sodass

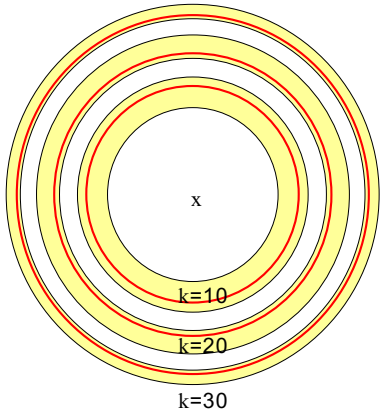
$$\forall k \le k_{\max} : k - \mathrm{NN} - \mathrm{Dist}(o) \le \mathrm{UB}_{k-\mathrm{NN}-\mathrm{Dist}}(o)$$

- LB-Approximation ist eine Linie im log(k)-log(dist)-Raum, sodass

$$\forall k \le k_{\max} : k - \mathrm{NN} - \mathrm{Dist}(o) \ge \mathrm{LB}_{k-\mathrm{NN}-\mathrm{Dist}}(o)$$







2.5 Reverse nächste Nachbarn Anfragen

- Jedem Objekt wird zugeordnet
  - Eine LB-Approximation der *k*-NN-Distanzen
  - Eine UB-Approximation der *k*-NN-Distanzen
- Jeder Seite im Index wird zugeordnet
  - Eine UB-Approximation der UB-Approximationen der Kindseiten
  - LB-Approximation wird nicht gespeichert, da zu wenig selektiv, d.h. zu wenig Kandidaten könnten aufgrund der zu ungenauen Approximation auf höheren Indexebenen als sichere Treffer erkannt werden, der damit verbundene Aufwand (Speicherung der Approximation + Anwendung bei Anfrage) ist zu hoch.
- Vorteil:
  - Beliebiges *k*
  - Für allgemein metrische Daten (M-Tree) oder Vektordaten (z.B. R-Tree)
- Nachteil
  - Updateproblematik (inhärent in der Klasse der self-pruning-Strategien)
  - k_{max} muss bekannt sein (ABER i.d.R. kein Problem)
  - Teurer Verfeinerungsschritt nötig, da Ergebnis nur Ergebnis-Kandidaten liefert (ABER i.d.R. deutlich weniger Kandidaten)

2.5 Reverse nächste Nachbarn Anfragen

- ***Filter***-Algorithmus für allgemein metrische Daten (M-tree)

  Knoten Node = (RoutingObj, CovRadius)

  MINDIST(q,Node) = max{dist(q,Node.RoutingObj) – CovRadius, 0}

```
MRkNNCoP-Tree-Search(DB, q)                    // DB als MRkNNCoP-Tree organisiert
    result = ∅;
    candidates = ∅;
    queue = LIST OF (dist:Real, obj:Object) ORDERED BY dist ASCENDING;
    queue = [(0.0, DB.root)];
    WHILE NOT queue.isEmpty() DO
        p = queue.first().Object;
        IF p.isDataPage() THEN
            FOR i=0 TO p.size() DO
                IF dist(q, p.getObject(i)) ≤ LB_k-NN-Dist(p.getObject(i)) THEN
                    result := result ∪ getObject(i);
                ELSE IF dist(q, p.getObject(i)) ≤ UB_k-NN-Dist(p.getObject(i)) THEN
                    candidates := candidates ∪ getObject(i);
            ELSE                                // p ist Directoryseite
            FOR i=0 TO p.size() DO
                IF MINDIST(q, p.getRegion(i)) ≤ UB_k-NN-Dist(p.getRegion(i)) THEN
                    queue.insert((MINDIST(q, p.getRegion(i)), p.childPage(i)));
```