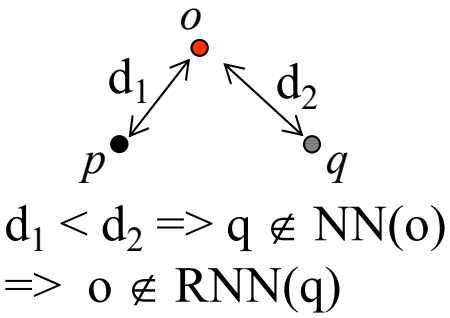
2.5 Reverse nächste Nachbarn Anfragen

2.5.3 Mutual Pruning Strategien

Idee:

- Ausschluss von Seiten/Objekten mittels Distanzabschätzungen (ohne Vorberechnung, zur Laufzeit der Anfrage ermittelt)
- Für mindestens *k* Objekte ist die Distanz zu Objekt o kleiner als die Distanz zwischen Anfrageobjekt q und o

  => Objekt o $\notin$ RkNN(q)



- Distanzabschätzung zwischen o und p nicht über vorberechneten (k)NN-dist(o), sondern erst zur Laufzeit (der Anfrage)
- Distanzabschätzungen ebenfalls über Indexseitenregionen

Vorteile:

- Keine Vorberechnung notwendig → keine Update-Problematik mehr !!!
- Parameter *k* zur Anfragezeit frei wählbar

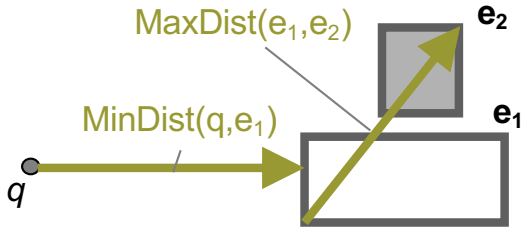
2.5 Reverse nächste Nachbarn Anfragen

- Min/Max-Distanz-Ansatz (Distanzabschätzung über Seitenregionen)

  [Achtert, Kröger, Kriegel, Renz, Züfle, EDBT, 2009]

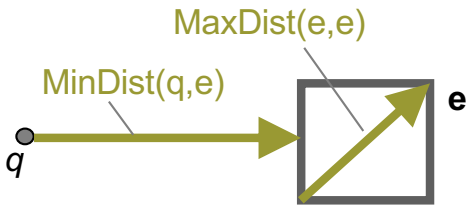
  - Vorraussetzung: Indexseite speichert Anzahl der Objekte |e| in Seite e

Ausschluß von Seite e_1 durch Seite e_2 ($k \geq 1$):



$MaxDist(e_1,e_2) < MinDist(q,e_1)$
$\Rightarrow e_1$ kann keine RkNN(q)-Ergebnisse enthalten wenn $|e_2| \geq k$

Seite e schließt sich selbst aus ($k \geq 1$):



$MaxDist(e,e) < MinDist(q,e)$
$\Rightarrow$ e kann keine RkNN(q)-Ergebnisse enthalten wenn $|e|-1 \geq k$

$MinDist(e_1,e_2)$: Distanz zwischen den beiden voneinander am nächsten liegenden Punkten $p_1 \in e_1$ und $p_2 \in e_2$

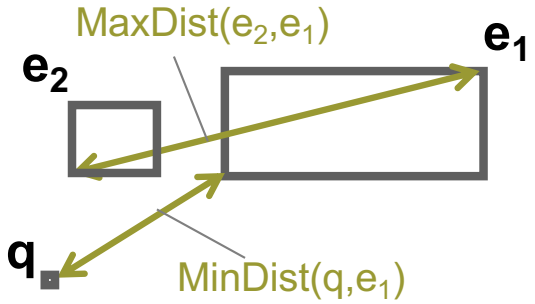
$MaxDist(e_1,e_2)$: Distanz zwischen den beiden voneinander entferntesten Punkten $p_1 \in e_1$ und $p_2 \in e_2$

  - Vorteile:
    - Keine Vorberechnung mehr (keine Update-Problematik)
    - Parameter k zur Anfragezeit frei wählbar
    - Pruningkonzept auf allg. metrische Daten anwendbar (z.B. M-tree-Seitenregionen)

2.5 Reverse nächste Nachbarn Anfragen

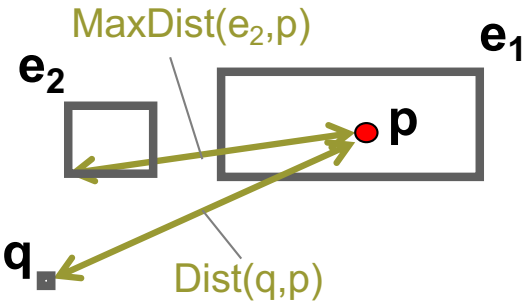
- Problem bei Distanzabschätzung über Seitenregionen:
  - Abhängigkeiten zwischen Distanzen unberücksichtigt

Min/Max-Dist Pruning:



MinDist(q,e_1) < MaxDist(e_2,e_1)

„Optimales“ Pruning:



$\forall p \in e_1$: Dist(q,p) > MaxDist(e_2,p)

  - Die Distanz MinDist(q,e_1) schätzt die Distanz dist(q,p) zwischen q und einem Objekt p in Seitenregion e_1 (untere Distanzabschätzung)
  - Die Distanz MaxDist(e_2,e_1) schätzt die Distanz dist(o,p) zwischen einem Objekt o in Seitenregion e_2 und einem Objekt p in Seitenregion e_1 ab (obere Distanzabschätzung)
  - Beide Distanzen dist(q,p) und dist(o,p) hängen von der Lage des Objektes p in Region e_1 ab => dist(q,p) abhängig von dist(o,p)
  - Diese (Abhängigkeits-) Information geht bei der Verwendung von MaxDist(e_2,e_1) und MinDist(q,e_1) verloren

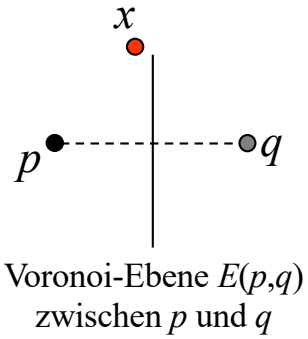
→ geringeres Pruningpotential

2.5 Reverse nächste Nachbarn Anfragen

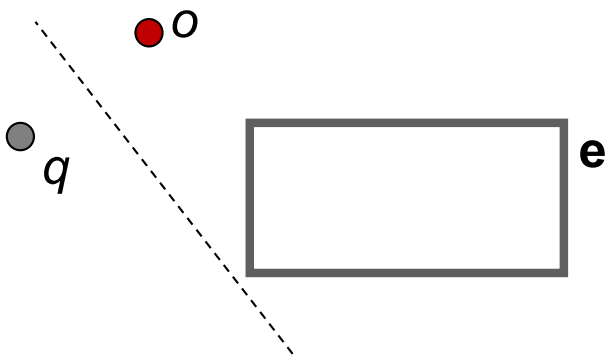
- Geometrische RNN-Suche (Filter/Verfeinerung)

  [Tao, Papadias, Lian. Int. Conf. Very Large Databases (VLDB), 2004]

  - Idee
    - Gegeben: Voronoi-Ebene zwischen q und beliebigen Punkt p
    - Liegt ein Punkt x auf der Seite von p dieser Voronoi-Ebene, kann q nicht NN von x sein und damit $x \notin RNN(q)$
    - Voronoi-Ebene $E(p,q)$: für alle Punkte $e \in E$ gilt: $dist(q,e) = dist(p,e)$
  - „Geometrisches“ Pruning ist optimal bzgl. der Pruning-Stärke

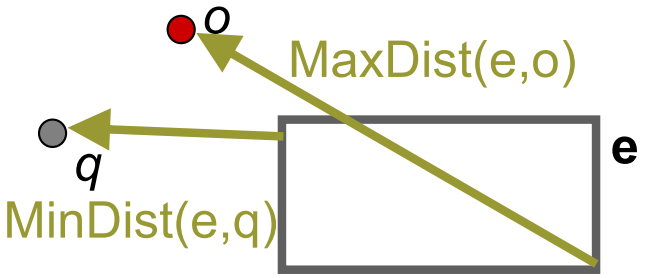


geometrisches Pruning:



e hinter der Hyperebene
=> $\forall p \in e: p \notin RNN(q)$ → prune e

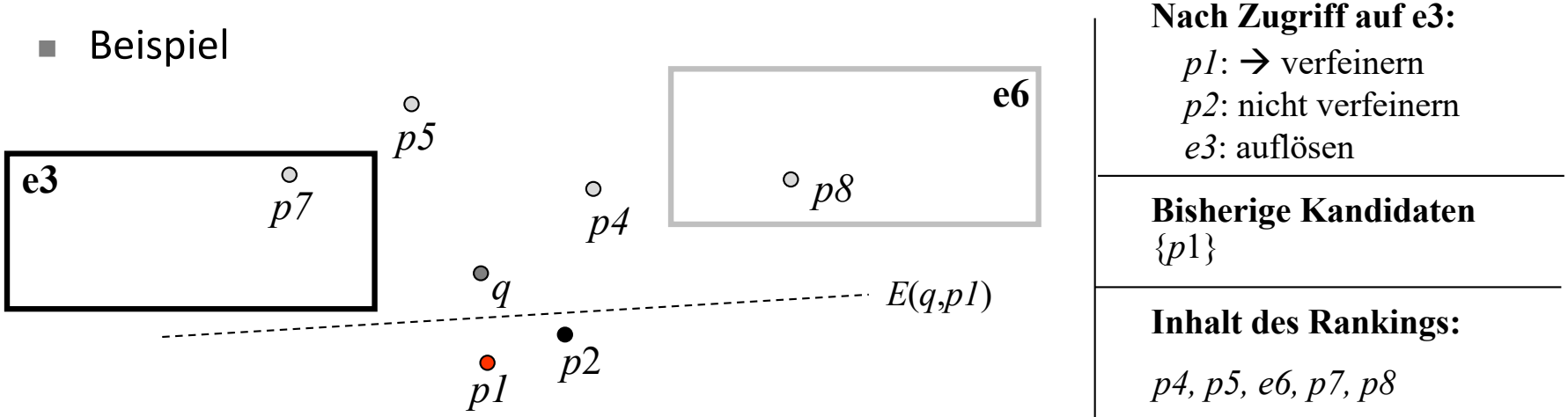
Min/Max-dist Pruning:



MaxDist(e,o) > MinDist(e,q)
=> Kein Ausschluss von e !

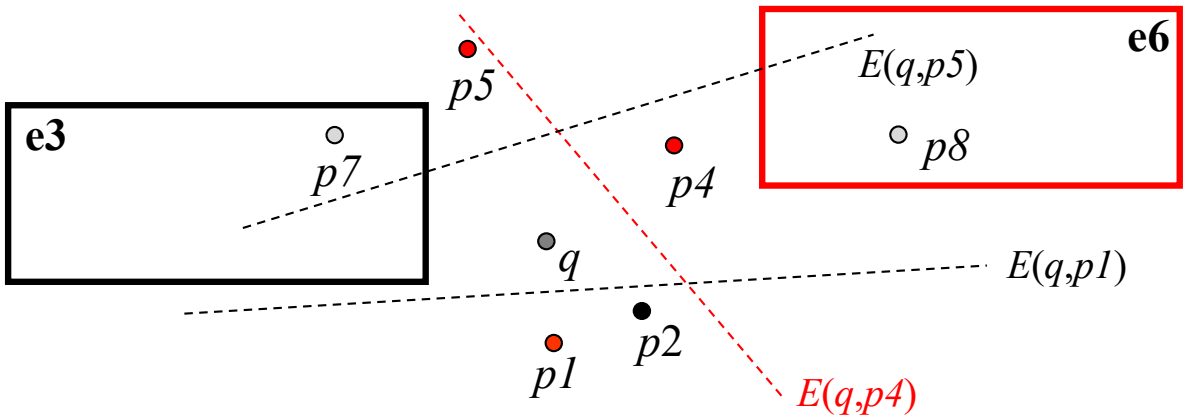
2.5 Reverse nächste Nachbarn Anfragen

- Algorithmus: Filter-Schritt (Skizze)
  - Berechne ein NN-Ranking der DB bzgl. Anfrageobjekt q
  - Solange noch Objekte im Ranking sind:
    - Rufe getNext() auf
    - Wenn aktueller Punkt *p* nicht „hinter“ einer Voronoi-Ebene liegt, konstruiere neue Voronoi-Ebene *E*(*p*,*q*); *p* wird zur Kandidatenmenge hinzugefügt
    - Punkte/Directoryseiten, die „hinter“ einer der Voronoi-Ebenen E(p,q) liegen (außer Punkt p selbst), können aus dem Ranking/Kandidatenmenge gelöscht werden
  - Punkte, die die Ebenen bestimmen, müssen verfeinert werden, d.h. für diese Punkte muss jeweils eine NN-Anfrage im Verfeinerungsschritt berechnet werden
- Beispiel



2.5 Reverse nächste Nachbarn Anfragen

- Zugriff auf *p4*
  - Streiche *e6* und *p8* aus Kandidatenliste (liegt nun hinter *E*(*q*,*p4*))



Nach Zugriff auf *p5*:

p4: → verfeinern
p5: → verfeinern
e6: nicht auflösen → prune

Bisherige Kandidaten
{*p*1, p4, p5}

Inhalt des Rankings:

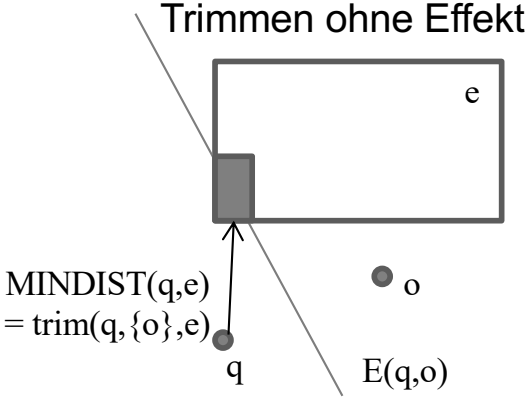
--

Nach Zugriff auf p5: Streiche *p7* aus dem Ranking → Ranking leer → verfeinere p1, p4, p5
Nach Verfeinerung von p5: Anfrageergebnis = {*p1*, *p4*}, *p5* = true drop (wegen NN(*p5*)=*p7*)

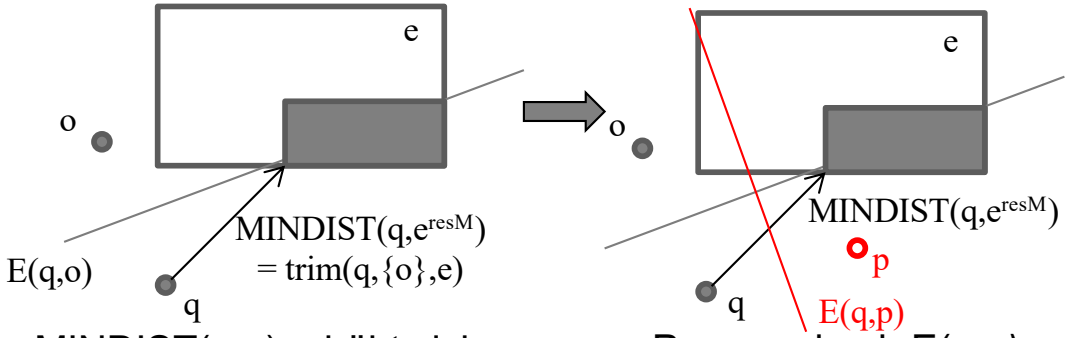
- Vorteil
  - *k* kann beliebig sein (Ausschlusskriterium: Objekt/Seite muss hinter *k* Ebenen liegen
  - Keine vorberechneten Distanzen, daher keine Update-Problematik und bessere Speicherkomplexität
- Nachteil
  - Nur für Vektordaten
  - Teurer Verfeinerungsschritt (eine NN-Anfrage pro Kandidat)
  - Teilweise komplexe Ebenenverwaltung

- Trimmen
  - Partielles abschneiden (trimmen) von Seitenregionen (Rechtecken) bzgl. einer Pruningebene

Fall 1:
Trimmen ohne Effekt



Fall 2:
Trimmen mit Effekt



MINDIST(q,e) erhöht sich

Prune *e* durch E(*q*,*p*)

  - Anpassung der MINDIST(q,e) nach dem Trimmen
    - ➔ führt eventuell zur Erhöhung der MINDIST(q,e)
    - ➔ erhöht die Chance, dass Seitenregion e früher ausgefiltert (geprunt) werden kann.

```
trim(q,S,e)
/* q = query point, S=set of DB points,
e = rectangle */
   e^resM = e
   for each p ∈ S do
      e^resM = clipping(e^resM, E(q,p))
      if e^resM = ∅ then return ∞
   return mindist(q, e^resM)
```

- Algorithmus (Filter):

Algorithm TPL-filter(q) /* q is the query point */
1. initialize a min-heap H accepting entries of the form (e, key)
2. initialize sets $S_{cnd}=\emptyset$, $S_{rfn}=\emptyset$
3. insert (R-tree root, 0) to H
4. while H is not empty
5. (e, key)=de-heap H
6. if (**trim**(q, S_{cnd}, e)=∞) then $S_{rfn}=S_{rfn}\cup\{e\}$
7. else // entry may be or contain a candidate
8. if e is data point p
9. $S_{cnd}=S_{cnd}\cup\{p\}$
10. else if e points to a leaf node N
11. for each point p in N (sorted on $dist(p,q)$)
12. if (**trim**(q,S_{cand},p)$\neq\infty$) then insert (p,$dist(p,q)$) in H
13. else $S_{rfn}=S_{rfn}\cup\{p\}$
14. else // e points to an intermediate node N
15. for each entry N_i in N
16. $mindist(N_i^{resM}, q)$=**trim**(q, S_{cnd}, N_i)
17. if ($mindist(N_i^{resM}, q)=\infty$) then $S_{rfn}=S_{rfn}\cup\{N_i\}$
18. else insert (N_i, $mindist(N_i^{resM}, q)$) in H

End TPL-filter

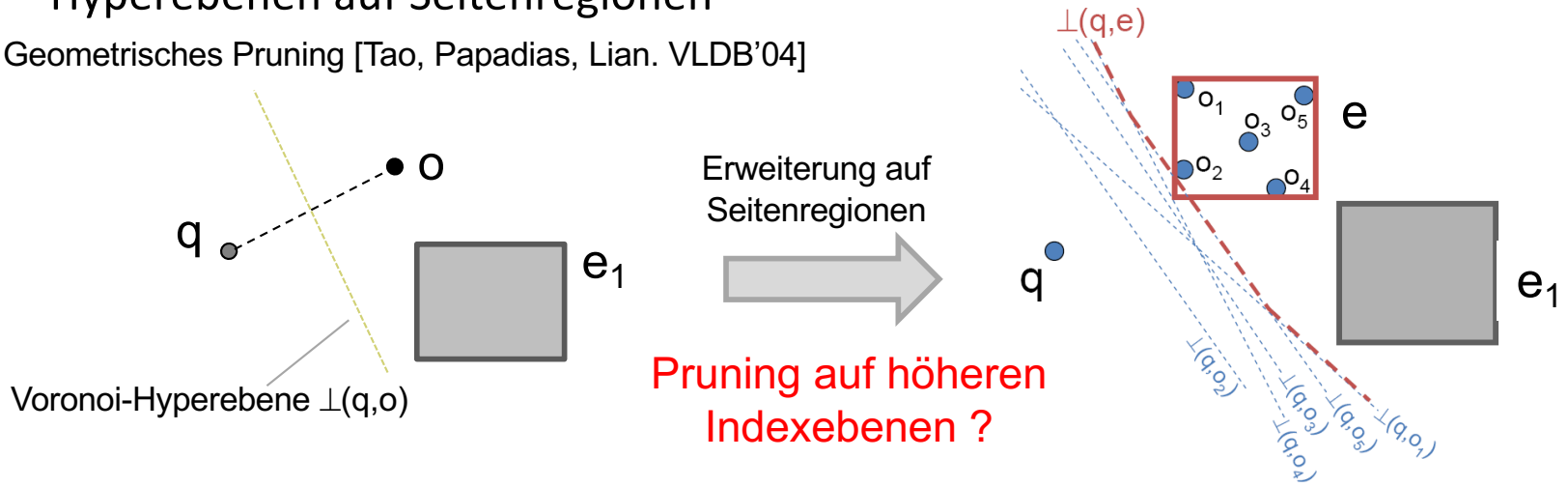
falls e hinter einer bestehenden Pruningebene

Quelle: [Tao, Papadias, Lian. Int. Conf. Very Large Databases (VLDB), 2004]

2.5 Reverse nächste Nachbarn Anfragen

- Weitere Eigenschaft der Geometrischen RNN-Suche
  - Ausschluß (Pruning) von Punkten/Seiten nur aufgrund anderer Punkte
  - Ausschlußkriterium könnte auch vollständig auf Directory-Ebene angewandt werden, aber wie? (Auflösung folgt …)
- Erweiterung des geometrischen Pruning-Konzepts auf Basis von Voronoi-Hyperebenen auf Seitenregionen

Geometrisches Pruning [Tao, Papadias, Lian. VLDB'04]



- Frage: Effiziente Repräsentation von Voronoi-Hyperebenen zwischen einem Punkt und einer Seitenregion ?
- Idee: Konservative Approximation der Hyperebenen
  Eigenschaft: $\forall p \in \perp(q,e)$: dist(q,p) = MaxDist(e,p)