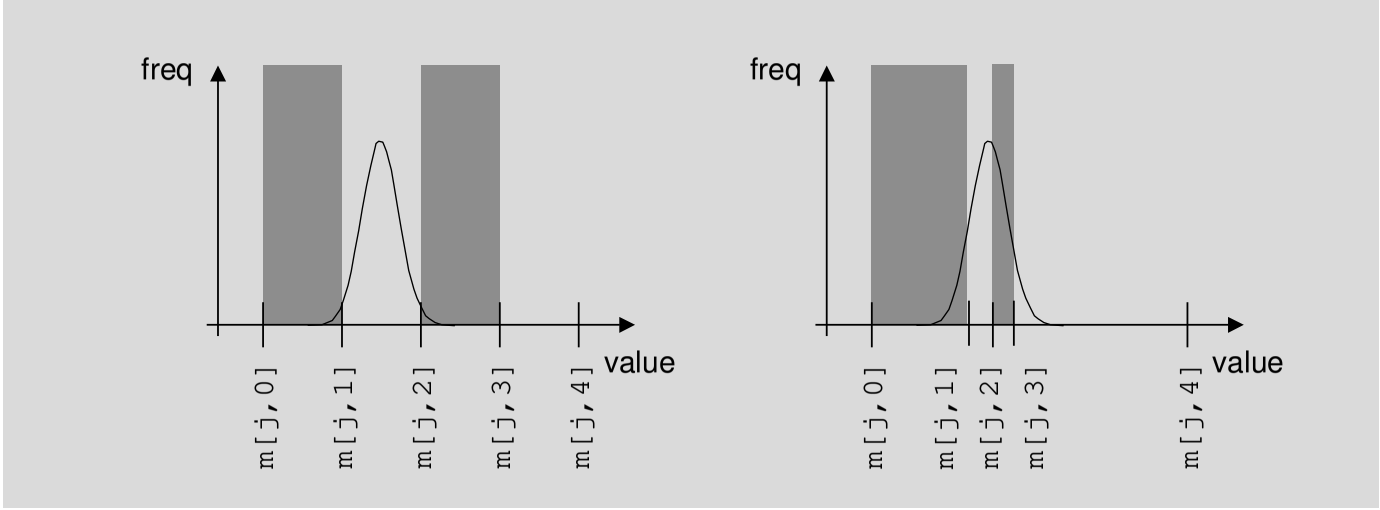
– **Bestimmung der Markierungen:** Entlang jeder Achse werden die Markierungspunkte so gewählt, dass jede Scheibe die gleiche Anzahl Punkte enthält. Zu diesem Zweck berechnet man zuerst die Verteilungsfunktion entlang der Achsen (z.B. durch Sampling einiger Punkte) und bestimmt die Markierungspunkte so, dass das Integral über die Verteilungsfunktion für alle Scheiben gleich gross ist (das Integral entspricht dabei der Anzahl Punkte in den Scheiben). Die untenstehende Figur zeigt eine typische Verteilungsfunktion der Komponentenwerte. Links wurden die Scheiben gleichförmig gewählt, rechts so, dass die Anz. Punkte pro Scheibe (ungefähr) gleich ist. Das linke Schema ist ungeeignet, da praktsich alle Punkte in der dritten Scheibe liegen.



– **Bestimmung der Anzahl Bits:** Grundsätzlich sollte die Anzahl Bits so gewählt werden, dass die Approximationen möglichst klein sind, aber dennoch eine effiziente Suche ermöglichen. Es gibt einige Modelle, wie man die Bits verteilen sollte und wie viele Bits total verwendet werden sollen. In der Regel genügen aber 4 oder 8 Bits pro Dimension, um eine gute Suchperformanz zu erreichen.

– **Berechnung der Approximation:** Wie erwähnt, setzt sich die Approximation eines Datenpunktes p aus den Bitstrings der den Punkt p enthaltenden Scheiben entlang aller Achsen zusammen. Dabei sei $a(p,j)$ der Bitstring (resp. die Nummer) der Scheibe in der j-ten Dimension, welche den Punkt enthält. Offensichtlich muss gelten

$$m[j, a(p,j)] \leq pj \leq m[j, a(p,j)+1]$$

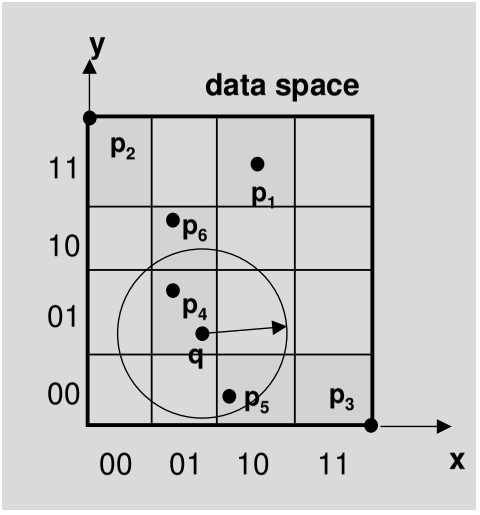
Die Approximation des Punktes p ist die Konkatenation aller Bitstrings $a(p,j)$.

- **Was nützen nun die Approximationen?**
  – Die Approximation eines Datenpunktes, d.h. der ihn umgebende rechteckige Bereich, hat ein recht kleines Volumen. Falls der Datenraum $[0,1]^d$, $b_i=b$ für alle Dimensionen und alle Markierungen gleichförmig verteilt sind, dann ist das Volumen einer Partition 2^{-bd}. Bei grossem b und d also beinahe 0. D.h. die Approximation bedeckt nur unwesentlich mehr Volumen als der Punkt selber (im Vergleich zu den Partitionen in einem R-Baum oder einem Grid-File).

– **Betrachte die Figur:** Sei q ein Anfragepunkt und der Kreis repräsentiere eine Bereichsanfrage für Euklidische Distanz. Die Approximationen der Punkte, repräsentiert durch die umgebenden Partitionen, erlauben die Aussage, dass

- p_1, p_2 und p_3 bestimmt nicht relevant für die Bereichsanfrage sind
- p_4 sicher relevant für die Bereichsanfrage ist
- p_5 und p_6 möglicherweise relevant für die Bereichsanfrage sind

Zur Beantwortung der Anfrage müssen also alle Approximationen der Datenpunkte gelesen werden und die Vektoren von p_5 und p_6, um das richtige Resultat $\{p_4, p_6\}$ zu bilden. Im hoch-dim. Raum haben wir bereits gesehen, dass jede hierarchische Struktur wie der seq. Scan alle Vektoren lesen muss, um eine nicht-leere Bereichsanfrage zu beantworten (NN-Suche ist eine nicht-leere Bereichsanfrage!). Mit dem VA-File brauchen wir aber nur noch die Approximationen (4-8 mal weniger Daten) und einige Datenpunkte zu lesen. Natürlich „funktioniert" dies nur, wenn nicht allzu viele Vektoren gelesen werden müssen, d.h. höchstens 20-50 Vektoren. Andernfalls sind die Kosten für das Lesen der Vektoren zu hoch (random access)



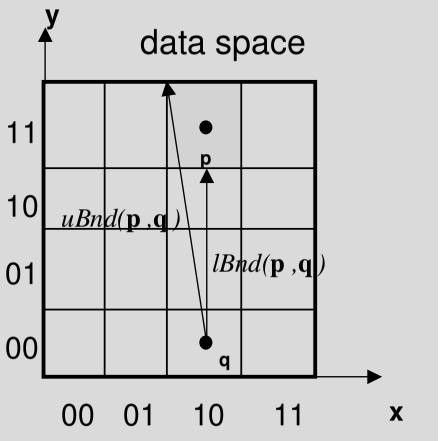
– **Berechnung von Schranken:** Die Approximation eines Datenpunktes p erlaubt es, eine obere (=*uBnd(p,q)*) und untere (=*lBnd(p,q)*) Schranke für die Distanz zwischen dem Datenpunkt p und einem Anfragepunkt q zu berechnen (siehe Grafik). Die untere (resp. obere) Schranke ist gegeben als minimale (maximale) Distanz zwischen Anfragepunkt und Partition, welche zur Approximation des Punktes gehört. Die Berechnung der Schranken ist z.B. für die Euklidische Distanz wie folgt (für andere Normen analog):

- Sei $l_j(p) = m[j,a(p,j)]$ die untere Schranke der Partition von p für die j-te Dim.
- Sei $u_j(p) = m[j,a(p,j)+1]$ die obere Schranke der Partition von p für die j-te Dim.

Dann gilt $lBnd(p,q) \le \delta(p,q) \le uBnd(p,q)$ mit

$$lBnd(p,q) = \sqrt{\sum_{j=0}^{d-1} \begin{cases} (l_j(p) - q_j)^2 & q_j < l_j(p) \\ (q_j - u_j(p))^2 & q_j > u_j(p) \\ 0 & l_j(p) \le q_j \le u_j(p) \end{cases}}$$

$$uBnd(p,q) = \sqrt{\sum_{j=0}^{d-1} \begin{cases} (u_j(p) - q_j)^2 & q_j \le \frac{l_j(p)+u_j(p)}{2} \\ (q_j - l_j(p))^2 & q_j > \frac{l_j(p)+u_j(p)}{2} \end{cases}}$$



– **Filter Bedingungen:** Die folgenden Bedingungen helfen, aufgrund der Approximationen Punkte von der Suche auszuschliessen.
  - Falls r ein Suchradius ist (Bereichsanfrage) für einen Anfragepunkt q, dann ist

    $uBnd(p,q) \leq r \Rightarrow$ „p in Antwort“
    „p in Antwort“ $\Rightarrow lBnd(p,q) \leq r$
    $lBnd(p,q) > r \Rightarrow$ „p nicht in Antwort“

    Dasselbe gilt auch für eine NN-Suche. In diesem Fall ist r durch die NN-Distanz von q gegeben.
  - Falls der NN gesucht wird, kann man zusätzlich noch folgende Bedingung aufstellen. Seien p und p' Datenpunkte und q der Anfragepunkt, dann:

    $uBnd(p',q) < lBnd(p,q) \Rightarrow$ „p nicht in Antwort“

– **Suchalgorithmen:**
  - Die folgenden Algorithmen beschreiben wie man eine Bereichsanfrage resp. eine NN-Suche mit dem VA-File evaluiert. Dabei beschreiben wir zwei Lösungen für die NN-Suche: die erste Methode ist optimiert bzgl. CPU-Kosten (NNSearchSSA), die zweite bzgl. IO-Kosten (NNSearchNOA)
  - Die Algorithmen benutzen stets zwei Phasen: in der ersten Phase werden die Approximationen gelesen und Punkte aufgrund obiger Bedingungen ausgeschlossen. In der zweiten Phase werden die exakten Vektoren gelesen und das Resultat berechnet.

– Benötigte Funktionen/Variablen:
  - **`ReadApxEntry(i,apx):`** Lesen der i-ten Approximation und speichern in **`apx`**
    `ReadVecEntry(i,vec):` Lesen des i-ten Vektors und speichern in **`vec`**
  - **`Q:`** Struktur, welche die Anfrage beschreibt; **`Q.query.q:`** Anfragepunkt; **`Q.query.r:`** Suchradius, **`Q.query.k:`** Anzahl NN.
  - **`GetLowerBound(apx,Q):`** Berechnet untere Schranke zur Anfrage für aktuelles **`apx`**
    `GetUpperBound(apx,Q):` Berechnet obere Schranke zur Anfrage für aktuelles **`apx`**
    `GetDist(vec,Q):` Berechnet die Distanz zwischen **`vec`** und Anfrage

– **Range Search:**

```
FUNCTION RangeSearch(Q: RangeSearch): Answer;
 VAR i:    INTEGER;          ans: Answer;
     dist: REAL;             apx: BitString;
     vec:  Vector;
 BEGIN
   FOR i:=0 TO maxPos DO
     IF ReadApxEntry(i,apx) THEN
       /*  first phase: filtering step */
       IF GetLowerBound(apx,Q)<Q.query.r THEN
         /* second phase: refinement step */
         ReadVecEntry(i,vec);
         dist:=GetDist(vec,Q);
         IF dist<Q.query.r THEN ans.Add(vec,dist) END;
       END;
     END;
   RETURN ans;
 END RangeSearch;
```

– **NN-Suche (CPU optimiert)**

- `NNSearchSSA` (SSA: simple search algorithm) ist ein einfacher Algorithmus, um die nächsten k Nachbarn für einen Punkt zu finden.
- Es werden alle Approximationen gelesen. Zu jedem Zeitpunkt enthalten die Arrays `v[]` und `dst[]`, die k besten bisher gesehenen Punkte und deren Distanzen zum Anfragepunkt (Kandidatenliste).
- Falls die aktuelle Approximation eine untere Schranke hat, welche kleiner ist als die grösste in `dst[]` gespeicherte Distanz (=`max`), dann könnte der aktuelle Punkt näher liegen als der entsprechende Punkt in `v[]`. In diesem Fall muss der Vektor gelesen (`ReadVecEntry`) und die exakte Distanz (`GetDist`) berechnet werden. Die Funktion `Candidate` überprüft dann, ob der aktuelle Punkt neu zu den besten k Punkten gehört und fügt in gegebenenfalls in die Kandidatenliste ein.
- NNSearchSSA benutzt nur die Filterbedingung

  $lBnd(p,q) > r \quad \Rightarrow$ „p nicht in Antwort“

  wobei r der maximalen Distanz der bisher k besten Punkte entspricht. Die Suche ist korrekt (der tatsächliche NN wird gefunden), da r stets grösser gleich der endgültigen NN-Distanz ist. Damit werden also nur Punkte ausgeschlossen, deren untere Schranke grösser als die NN-Distanz ist. Für alle anderen Punkte wird die exakte Distanz bestimmt und mit Hilfe der Kandidatenliste werden die besten k Punkte ermittelt.


```
FUNCTION NNSearchSSA(Q: NNSearch): Answer;
  VAR i:   INTEGER;      v:    ARRAY OF Vector;   dst:  ARRAY OF REAL;
      apx: BitString;    ans:  Answer;            vec:  Vector;
      max: REAL;         dist: REAL;

   /* initializes candidate set; returns MAXREAL as k-th smallest distance */
   FUNCTION InitCandidate(): REAL; BEGIN
     FOR i:=1 TO Q.query.k DO dst[i]:=MAXREAL; v[i].OID:=-1 END;
     RETURN MAXREAL;
   END InitCadidate;

   /* adds a point to the candidate set; returns k-th smallest distance */
   FUNCTION Candidate(p: Vector; dist: REAL): REAL; BEGIN
     IF dist<dst[Q.query.k-1] THEN
       dst[Q.query.k-1]:=dist; v[Q.query.k-1]:=p;
       SortOnDst(v,dst,Q.query.k);
     END;
     RETURN dst[Q.query.k-1];
   END Candidate;

 BEGIN
   max:=InitCandidate();
   FOR i:=0 TO maxPos DO
     IF ReadApxEntry(i,apx) THEN
       /* first phase: filtering step */
       IF GetLowerBound(apx,Q)<max THEN
         /* second phase: refinement step */
         ReadVecEntry(i,vec);
         dist:=GetDist(vec,Q);
         max:=Candidate(vec,dist);
       END;
     END;
   /* build answer set, compute scores */
   FOR i:=0 TO Q.query.k-1 DO ans.Add(v[i],dst[i]);
   RETURN ans;
 END NNSearchSSA;
```

– **NN-Suche (IO optimiert)**

- `NNSearchNOA` (NOA: near optimal algorithm) ist eine Erweiterung von `NNSearchSSA`, welche nur gerade jene exakten Vektoren liest, welche auch wirklich notwendig sind (ausser in einer degenerierten Konstellation)
- Im Gegensatz zu `NNSearchSSA`, `NNSearchNOA` trennt konsequent zwischen erster und zweiter Phase. In der ersten Phase wird die Filterbedingung

  $uBnd(p',q) < lBnd(p,q) \quad \Rightarrow$ „p nicht in Antwort“

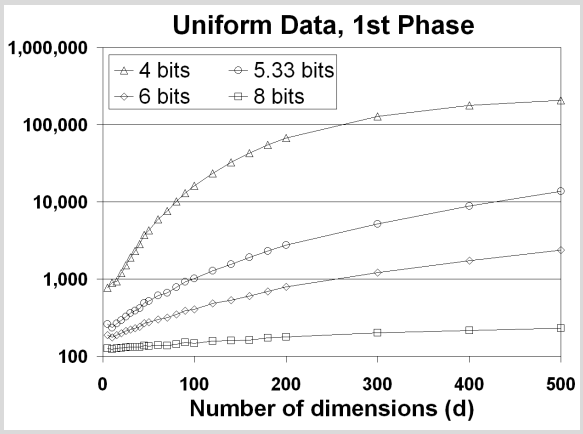
  angewendet. Ein Punkt ist ein Kandidat, wenn seine untere Schranke zum Zeitpunkt des Einlesens der Approximation kleiner war als die k besten, bis dahin gesehenen oberen Schranken. D.h. nach der ersten Phase ist es möglich, dass Punkte die obige Bedingung erfüllen (d.h. sicher nicht in der Antwort liegen).
- In der zweiten Phase werden die exakten Vektoren in aufsteigender Reihenfolge der unteren Schranken gelesen. Für jeden Vektor wird die exakte Distanz berechnet und es werden, analog zu `NNSearchSSA`, mittels der `Candidate` -Funktion die besten k Punkte behalten. Die zweite Phase kann gestoppt werden, falls die aktuelle untere Schranke grösser ist, als die k besten bisher gesehenen Distanzen. Alle verbleibenden Punkte kommen als Antwort nicht in Frage wegen der Filterbedingung

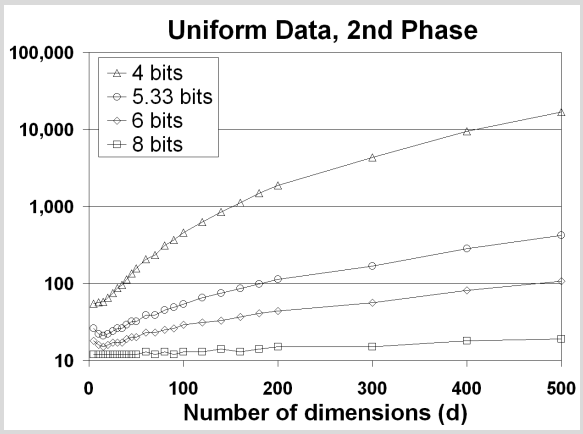
  $lBnd(p,q) > r \quad \Rightarrow$ „p nicht in Antwort“

  wobei r der maximalen Distanz der bisher k besten Punkte entspricht. Die Suche ist korrekt und optimal, da nur Punkte betrachtet werden, deren untere Schranke kleiner als die endgültige NN-Distanz ist.

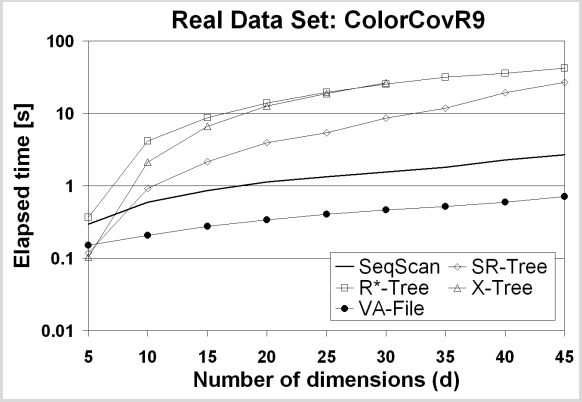

```
FUNCTION NNSearchNOA(Q: NNSearch): Answer;
  VAR i:         INTEGER;     v:    ARRAY OF Vector;       dist:     REAL;
      apx:       BitString;   dst:  ARRAY OF REAL;         heap:     Heap;
      ans:       Answer;      vec:  Vector;                 max,lBnd: REAL;

   /* as in Function NNSearchSSA */
   FUNCTION InitCandidate(): REAL;
   FUNCTION Candidate(p: Vector; dist: REAL): REAL;
 BEGIN
   /* first phase: filtering step */
   max:=InitCandidate();
   FOR i:=0 TO maxPos DO
     IF ReadApxEntry(i,apx) THEN
       lBnd:=GetLowerBound(apx,Q);
       IF lBnd<max THEN
         /* Candidate maintains only the k-th best upper bound seen so far */
         max:=Candidate(NullVec,GetUpperBound(apx,Q));
         heap.Push(i,lBnd);
       END;
     END;
   /* second phase: refinement step */
   max:=InitCandidate();lBnd=0;
   WHILE heap.IsNotEmpty() AND lBnd<max DO
     heap.Pop(i,lBnd);      /* read out position and lower bound */
     IF lBnd<max THEN
       ReadVecEntry(i,vec);
       max:=Candidate(vec,GetDist(vec,Q));
     END
   END;
   FOR i:=0 TO Q.query.k-1 DO ans.Add(v[i],dst[i]);
   RETURN ans;
 END NNSearchNOA;
```

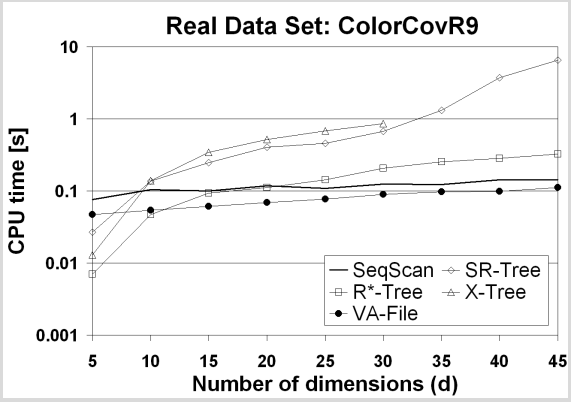
- **Vergleich der Performanz von NN-Suche**
  - NN-Suche mit dem VA-File ist nur effizient, falls die Anzahl gelesener Vektoren in der zweiten Phase klein ist. Andernfalls hat man den Vorteil der ersten Phase (seq. Lesen einer kleinen Approximationsdatei) schnell verspielt (siehe Bäume: random access auf Disk ist recht teuer im Vergleich zu sequentiellem Lesen).
  - Die Performanz von `NNSearchNOA` und `NNSearchSSA` ist vergleichbar gut. Im folgenden betrachten wir nur noch `NNSearchNOA`.
  - Die folgende Grafiken zeigen die Anzahl Punkte, welche nach der ersten und zweiten Phase übrig bleiben (hier für uniforme Daten; andere Datensätze liefern vergleichbare Resultate). Die Datenbank enthält ***230'000*** Punkte. ***k=10***.





  - Wählt man also z.B. ***8*** Bits pro Dimension, dann bleiben nach der ersten Phase einer ***10***-NN-Suche noch etwa ***300*** Punkte übrig und nach der zweiten Phase noch etwa ***15-20*** Punkte. Mit anderen Worten, von den ***230'000*** Punkten in der Datenbank müssen neben den ***10*** Punkten der Antwort nur noch ***5-10*** zusätzliche Punkte gelesen werden. D.h. die Selektivität der Filterschritte ist enorm gut. Damit fallen die Kosten der zweiten Phase kaum ins Gewicht.
  - Die Grafiken zeigen ein repräsentativen Vergleich mit den Merkmalsvektoren unser Bilddatenbank (***N=230'000, k=10***). Sowohl CPU als auch totale Suchzeiten (Elapsed time) des VA-Files sind deutlich kleiner als jene der anderen Such-algorithmen (logarithmische Skala). Nur bei kleiner Dimensionalität können Bäume mithalten; oberhalb ***d=10*** sind Bäume auch schlechter als der seq. Scan.





3.5 Auswertung komplexer Anfragen

- Wir haben in Kapitel 3.3 gesehen, wie man für komplexe Anfragen einen Ähnlichkeitswert (score) berechnen kann. Einfache Anfragen (ein Referenzobjekt, ein Merkmal) können mit Hilfe einer der in Kapitel 3.4 vorgestellten Indexstrukturen beantwortet werden. Für komplexere Anfragen brauchen wir aber zusätzliche Algorithmen und/oder Indexstrukturen.
- Die folgenden Vorschläge betrachten jeweils eine spezielle komplexe Anfrage; die Zusammensetzung könnte aufgrund der Vereinigung der Vorschläge umgesetzt werden. Zuletzt wollen wir kurz noch den Vorschlag in der Bilddatenbank der DBS Gruppe besprechen.
  - **1. Vorschlag [Ciaccia:1998]:** Ermöglicht die Auswertung komplexer Anfragen mit mehreren Referenzobjekten in beliebigen Indexstrukturen.
  - **2. Vorschlag [Fagin:1996]:** Middleware Algorithmus für die Auswertung komplexer Anfragen in verschiedenen Subsystemen. Damit sind grundsätzlich alle komplexen Anfragen beantwortbar. Im Speziellen ist der Vorschlag geeignet für Anfragen über mehrere Merkmale (auch Text & Merkmal).
  - **3. Vorschlag [Chaudhuri:1996]:** Löst komplexe Anfragen mit Prädikat- und Ähnlichkeitsbestandteil.
  - **4. Vorschlag [Weber:2001]:** Beliebige komplexe Anfragen basierend auf dem VA-File.

1. Vorschlag: Ciaccia:1998

- Die Anfrage setze sich aus mehreren Referenzobjekten zusammen. Alle Referenzobjekte bedienen sich desselben Merkmals zur Beschreibung der Ähnlichkeit.
- Die Idee von Ciaccia et al. wurde zwar für den M-Tree entwickelt, doch ist er allgemein verwendbar (also auch in allen anderen Bäumen oder seq. Strukturen). Die Grundüberlegung ist die folgende:
  - Die Distanzverknüpfungsfunktion muss monoton sein, d.h.

    $x1>x1'$ und $x2>x2'$ => $D^w(x1,x2)>D^w(x1',x2')$

    Diese Eigenschaft ist keine Einschränkung sondern eine sehr natürliche Anforderung an Distanzverknüpfungsfunktionen: falls die Distanzen der Teilanfragen kleiner sind, so muss auch die Gesamtdistanz kleiner sein.
  - Bei der NN-Suche verwendet man alle Referenzobjekte und rechnet mit Gesamtdistanzen. Mit dem HS-Algorithmus in Bäumen (analog mit dem VA-File) müssen Gesamtdistanzen zwischen den Referenzobjekten und den Datenpunkten sowie den MBRs berechnet werden. Gesamtdistanzen zu den Datenpunkten sind kein Problem. Für die Gesamtdistanzen zwischen den Referenzobjekten und den MBRs muss man garantieren können, dass die Gesamtdistanzen zu den Punkten innerhalb der MBRs grösser ist als jene zu den MBRs (d.h. die Distanz zu den MBRs ist eine untere Schranke für die Distanzen zu den Punkten). Wegen der Monotonie-Eigenschaft der Distanzverknüpfungsfunktion und weil die Forderung für ein Referenzobjekt gilt, trifft sie auch für mehrere Referenzobjekte zu.

- Damit ändern sich die Suchmethoden wie folgt:
  - Die NN-Suche wird mit mehreren Referenzobjekten aufgerufen.
  - Die Berechnung von Gesamtdistanzen erfolgt wie in Kapitel 3.3 beschrieben.
  - Falls unter (oder obere) Schranken auf diese Distanzen berechnet werden müssen, so berechnet man erst die Schranken für die einzelnen Ref.objekte und kombiniert sie zu einer Schranke auf die Gesamtdistanz (Bem.: die minimale Distanz zu einer MBR ist eine untere Schranke für die Distanz der von der MBR enthaltenen Punkte).

- **Vorteile:**
  - Einfache Implementierung. Die existierenden Suchalgorithmen müssen nur minimal angepasst werden
  - Kann für jede beliebige Suchstrukturen angewendet werden
  - Die Indexstruktur wird nur einmal für alle Ref.objekte traversiert

- **Nachteile:**
  - Die Suche ist nicht mehr so selektiv wie mit einem Ref.objekt. In Bäumen müssen mehr Blätter gelesen werden, im VA-File mehr exakte Vektoren in der 2. Phase

2. Vorschlag: Fagin:1996

- Grundsätzlich könnte die Anfrage beliebig komplex sein. Da die Vorschläge 1 und 3 aber besser geeignet sind für die jeweilige Spezialform, betrachten wir hier nur Anfragen mit mehreren Merkmalen aber nur einem Referenzobjekt.
- Der Vorschlag von Fagin operiert in seiner ursprünglichen Form mit „grades“ und „scores“. Grades und Scores sind Ähnlichkeitswerte, wie sie in Kapitel 3.3 jeweils ganz am Schluss berechnet wurden (Anwendung der h-Funktion auf die Gesamtdistanz). Fagin geht davon aus, dass die Teilanfragen bereits Ähnlichkeitswerte liefern und dass daraus eine Gesamtähnlichkeit berechnet werden muss. Mit anderen Worten, für die Distanzen der Teilanfragen würde erst die h-Funktion angewendet werden; diese Werte werden grades genannt. Die grades werden dann mit scoring-Funktionen (ähnlich der Distanzverknüpfungsfunktionen) zu scores verrechnet. Wir können den Algorithmus aber entsprechend modifizieren, so dass anstelle von grades und scores Distanzen verwendet werden können.
- Wiederum ist die Monotonie-Eigenschaft der Distanzverknüpfungsfunktionen eine wichtige Voraussetzung

  $x1>x1'$ und $x2>x2'$ => $D^w(x1,x2)>D^w(x1',x2')$
- Ferner geht der Algorithmus davon aus, dass ein Subsystem i die i-te Teilanfrage auswertet. In unserem Falle entspricht dies der NN-Suche über das i-te Merkmal. Eine zusätzliche Komponente (middleware) liest die Antworten der Subsysteme, kombiniert sie und produziert das endgültige Resultat.

- Der Algorithmus operiert in drei Phasen
  - **1. Phase (sorted access):** Jedes Subsystem berechnet seine Teilanfrage und liefert einen Antwortstrom mit den Objekten und ihren Distanzen. Der Strom ist aufsteigend sortiert, d.h. zuerst die ähnlichsten Objekte, danach die weniger ähnlichen. Die middleware Komponente liest von jedem Strom der Subsysteme eine bestimmte Anzahl von Elementen. Sei A die Menge der Objekte, welche von *mindestens einem* der Subsysteme gelesen wurde, und sei L die Menge der Objekte, welche von *allen* Subsystemen gelesen wurden. Die erste Phase bricht ab, bis L k Elemente enthält, d.h. bis k Objekte von allen Subsystemen geliefert wurde.
  - **2. Phase (random access):** Bestimme für alle Objekte in A die Menge der noch unbekannten Distanzen für alle Subsysteme.
  - **3. Phase (computation):** Berechne die Gesamtdistanzen für alle Objekte in A und wähle die k Besten aus.

- **Korrektheitsbeweis (Idee):** Sei $k=1$
  - In der ersten Phase werden von jedem Subsystem so viele Einträge gelesen, bis ein Objekt x von allen Strömen geliefert wurde. Mit diesem Objekt und mit Hilfe der Monotonie-Eigenschaft können Objekte von der Suche ausgeschlossen werden: alle Objekte, welche nicht in A sind, haben eine Distanz, welche bzgl. aller Teilanfragen grösser sein muss als diejenige von x (andernfalls wären sie in mind. einem Strom vor x gelegen). Aufgrund der Monotonie muss also auch die Gesamtdistanz dieser Objekte grösser sein als jene von x.
  - Für die Objekte in A können wir nicht sicher sagen, ob sie besser oder schlechter sind als x (aber: siehe QuickCombine). Daher müssen für alle Punkte Gesamtdistanzen berechnet werden.
- **Beispiel:** k=1, 3 Merkmale

Farbe	Form	Textur
a, 0.4	a, 0.6	d, 0.1
b, 0.5	d, 0.7	b, 0.2
c, 0.7	b, 0.8	e, 0.7
d, 0.9	z, 0.9	z, 0.8
...	...	...

1. Phase: lesen der ersten drei Einträge pro Strom; b wurde von allen Strömen geliefert
L= {b}, A={a,b,c,d,e}

2. Phase: Bestimmung der fehlenden Distanzen der Objekte in A.

3. Phase: Berechnung der Gesamtdistanzen und Bestimmung des besten Objektes.

- **Bemerkungen:**
  - Die 2. Phase (random access) kann recht teuer werden, falls die Anzahl Objekte in A gross ist. Zudem muss für jedes Objekt in A und jedes Merkmal eine Distanz berechnet werden. In der Praxis hat sich gezeigt, dass gerade diese Phase zu teuer ist. Z.B. hatten wir in unserer Bilddatenbanken Anfragezeiten von 10-20 Minuten erhalten. Hauptgrund für die schlecht Performanz: in der 2. Phase mussten für mehr als 10'000 Objekte Distanzen für die Teilanfragen berechnet werden.
  - QuickCombine [Güntzer:2000] ist eine Erweiterung von A0, welche vor allem die Anzahl random accesses in der 2. Phase reduziert. Einerseits nützt QuickCombine mehr Möglichkeiten für das Pruning aus, andererseits werden die Objekte mit einer optimierten Strategie von den Subsystemen angefordert (bei Fagin geschieht dies mehr oder weniger zufällig resp. uniform). In dem Beispiel auf der vorherigen Seite kann man weitere, nützliche Aussagen über die Objekte in A machen.
    - Annahme: Distanzen werden mit max verknüpft. Dann hat b die Gesamtdistanz $max\ \{0.5, 0.8, 0.2\}=0.8$. Objekt e wurde nur in der Teilanfrage „Textur" gelesen. Für die anderen Teilanfragen kennt man die exakte Distanz nicht. Man weiss aber, dass sie jeweils grösser sein müssen, als die grösste von diesen Strömen gelesene Distanz. Also: für Farbe muss die Distanz für e grösser als 0.7 sein, für Form grösser als 0.8. Damit muss die Gesamtdistanz von e grösser als 0.8 sein. Somit kann e unmöglich besser als b sein (höchstens gleich gut).
    - Objekt d kann analog behandelt werden. Allerdings erhält man hier eine untere Schranke von 0.7. Also könnte d besser sein als b.

    - Diese und weitere Überlegungen haben zu einem Algorithmus geführt, welcher bis zu 100-mal schneller ist als der A0-Algorithmus.

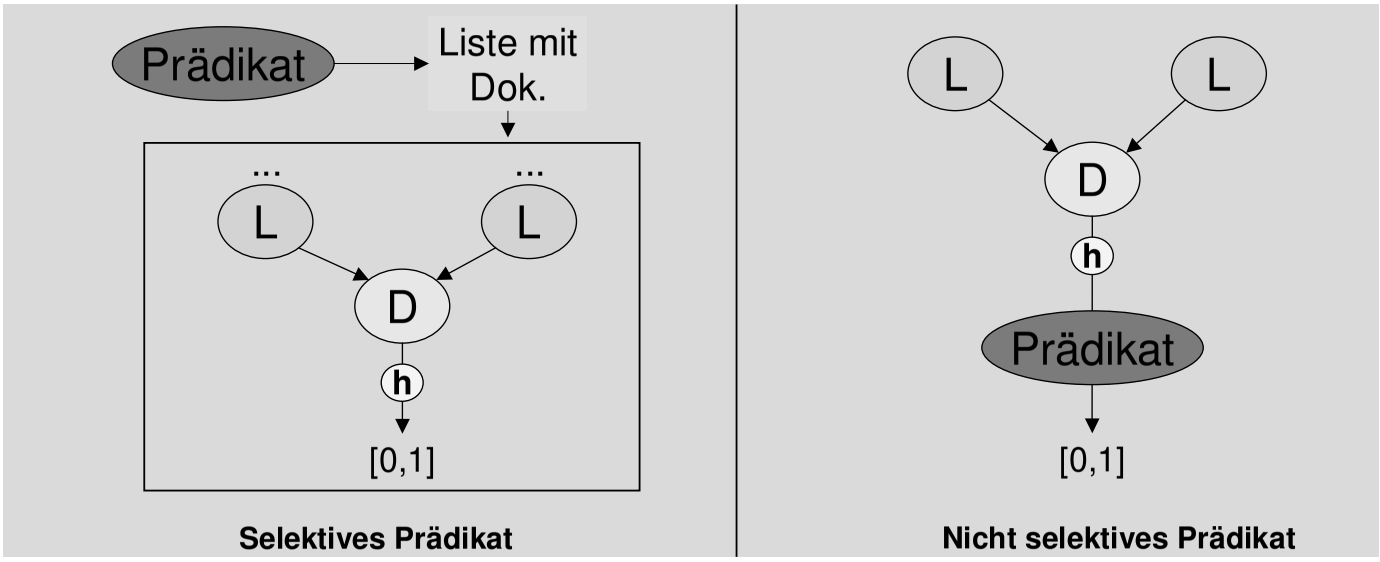
- **Vorteile:**
  - Beliebig komplexe Anfragen können evaluiert werden
  - Falls Bäume als Index-Strukturen verwendet werden, dann ist dies die einzige bisher bekannte Möglichkeit für komplexe Anfragen mit mehreren Merkmalen
  - Relativ einfache Implementierung möglich

- **Nachteile:**
  - Suchzeiten können enorm ansteigen. Selbst Fagin et al. mussten zugeben, dass die Effizienz des Algorithmus zu wünschen übrig lässt. Ein naiver Ansatz (lesen aller Objekte von allen Subsystemen) ist in vielen Fällen effizienter, wenn auch komplexitätsmässig nicht ganz so gut (der A0-Algorithmus hat sublineare Komplexität)
  - Viele Indexstrukturen oder Subsysteme erlauben keine „random accesses". Z.B. in einem R-Baum ist es nicht möglich, nach der OID zu suchen (sondern nur nach der Punktdarstellung, welche aber in diesem Fall nicht bekannt ist). Konsequenterweise müsste man neben dem R-Baum auch einen B-Baum für die „random access" Phase zur Verfügung stellen.

3. Vorschlag: Chaudhuri:1996

- Chaudhuri et al. haben eine Reihe von Algorithmen entwickelt, welche Ähnlichkeitsanfragen mit Prädikatsanfragen verknüpfen. Z.T. sind die Fragestellungen noch komplexer als in unserem Anfragemodell in Kapitel 3.3 (wobei die zusätzliche Komplexität meistens gar nicht benötigt wird)
- Zusammenfassend kann man die Algorithmen für die gegebene Anfrageart wie folgt beschreiben:
  - Ein Subsystem wertet Ähnlichkeitsanfragen aus (z.B. eine Methode aus Kapitel 3.4), ein zweites System die Prädikatanfragen (z.B. eine Datenbank).
  - Je nach Selektivität des Prädikates wird eine der folgenden Strategien gewählt
    - falls das Prädikat selektiv ist (d.h. wenige Objekte erfüllen das Prädikat), so wird zuerst die Menge der Objekte bestimmt, welche das Prädikat erfüllen. Für jedes dieser Objekte wird in einem zweiten Schritt der Ähnlichkeitswert berechnet und das Resultat bestimmt.
    - falls das Prädikat nicht selektiv ist, so wird zuerst eine Ähnlichkeitsanfrage gestartet. Aus der Antwort werden dann all jene Objekte gestrichen, welche das Prädikat nicht erfüllen. Evtl. müssen noch weitere Objekte vom Subsystem für Ähnlichkeitsanfragen nachgefordert werden, damit die Antwort auch wirklich k Elemente enthält

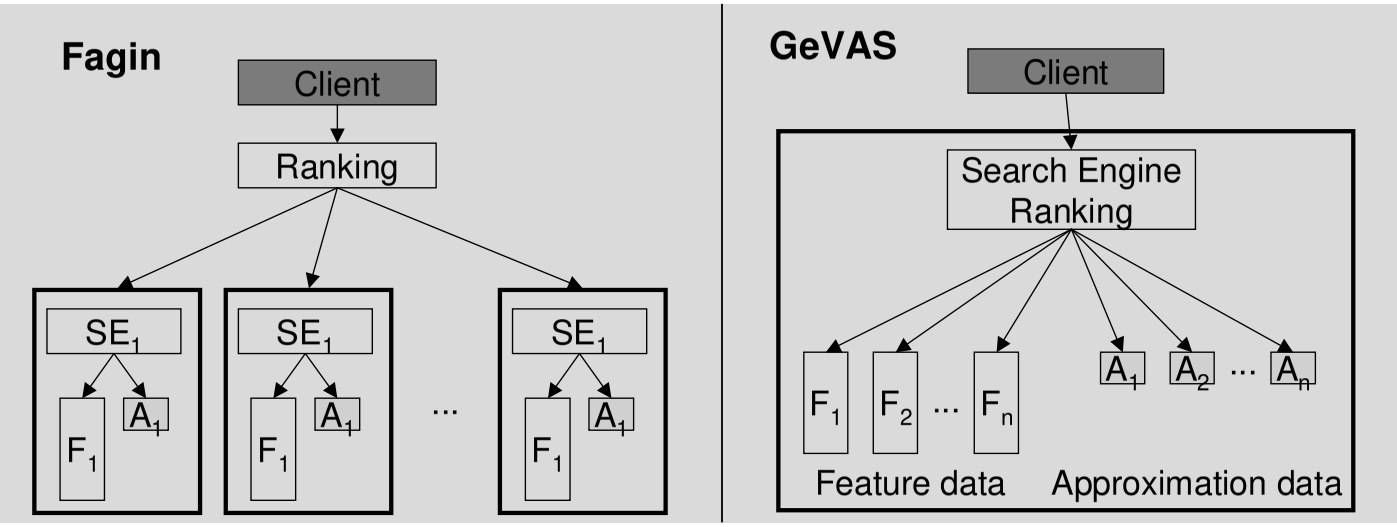
- **Vorteile:**
  - Falls das Prädikat sehr selektiv ist, oder falls das Prädikat nicht selektiv ist, dann ist der Algorithmus sehr effizient
  - Einfache Implementierung möglich
- **Nachteile:**
  - Falls die Selektivität „irgendwo“ zwischen selektiv und nicht selektiv ist, dann nützt uns weder die eine, noch die andere Strategie etwas.
- **Darstellung:** Die beiden Strategien können wie folgt visualisiert werden



Selektives Prädikat **Nicht selektives Prädikat**

4. Vorschlag: Weber:2001

- Die Idee von Ciaccia wurde für allgemeinere Anfragearten erweitert. Anstatt das Problem auf einer höheren Ebene zu lösen (middleware wie in Fagin, Chaudhuri), werden die Indexstrukturen selber „fit gemacht“ für eine Parallelisierung auf der unteren Stufe. Diese neuen Algorithmen heissen GeVAS (generalized va-file search)
- Die untenstehende Grafik zeigt den Unterschied zwischen Fagin und GeVAS für Anfragen mit mehreren Merkmalen. Anstelle die Ranglisten auf höherer Ebene zu kombinieren (Fagin), werden bei GeVAS die Approximations- und Vektordaten parallel gelesen (d.h. virtuell entsteht ein langer kombinierter Vektor)



- Da in der Praxis Prädikate mit sehr starker Selektivität resp. sehr schwacher Selektivität eher selten sind, bildet GeVAS die Prädikate zu einem N-dimensionalen Array ab, welches analog zu den Merkmalen behandelt werden kann. Enthält die Komponente des Arrays eine 1, so erfüllt das i-te Objekt das Prädikat. Die Suchalgorithmen müssen nur geringfügig angepasst werden: in der ersten Phase wird zuerst geprüft, ob das Prädikat erfüllt ist. Falls ja, dann werden Schranken berechnet usw., falls nein, dann wird das Objekt sofort gestrichen. Um die Suchzeiten weiter zu reduzieren, kann die Auswertung des Prädikates (auf Datenbank) parallel zum Lesen der Approximationen erfolgen. Natürlich müssen die beiden Aufgaben synchronisiert werden, d.h. wenn der Suchalgorithmus den i-ten Eintrag im Prädikatsarray benötigt, dann muss dieser Wert bereits berechnet worden sein. Andernfalls muss der Suchalgorithmus warten.
- GeVAS erlaubt es, Anfragen mit der vollen Komplexität des in Kapitel 3.3 vorgestellten Modells effizient zu beantworten:
  - Textanfragen werden wie folgt behandelt:
    - Boolesche Anfragen können als Prädikate betrachtet werden
    - Signaturen werden wie Approximationen behandelt
    - Vektorraumretrieval und Probabilistisches Retrieval verwenden anstelle eines booleschen Arrays bei Prädikaten ein Array mit RSV-Werten
  - Mehrere Referenzobjekte werden wie bei Ciaccia behandelt
  - Anfragen über mehrere Merkmale und mit Prädiakten: wie oben besprochen

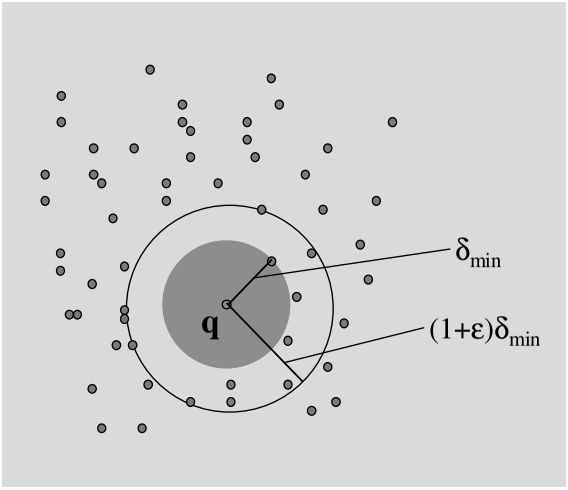
- **Vorteile:**
  - Bis zu 100-mal schneller als Vorschlag von Ciaccia (wegen Dominanz von VA-File gegenüber von M-Tree) und über 1000-mal schneller als Fagin (wegen hoher Kosten des random acces in Fagin).
  - Einfache Implementierung
- **Nachteil:**
  - Aufwand steigt linear zur Datenbankgrösse und der Komplexität der Anfrage

3.6 Approximierte und parallele Ähnlichkeitssuche

- In Kapitel 3.4 und 3.5 haben wir gesehen, dass die Kosten für die Ähnlichkeitssuche linear mit der Grösse der Datenbank wächst. Um diese ungünstige Effizienz zu verbessern, wurden zwei Arten von Vorschlägen (in verschiedenen Varianten) gemacht:
  - **Approximierte Ähnlichkeitssuche:** Die Idee besteht darin, nicht das exakte Resultat zu berechnen, sondern ein approximiertes Resultat, welches aber deutlich weniger Kosten verursacht.
  - **Parallele Ähnlichkeitssuche:** Anstatt die Suche nur auf einem Rechner durchzuführen, werden mehrere Rechner für die Ähnlichkeitssuche eingespannt.

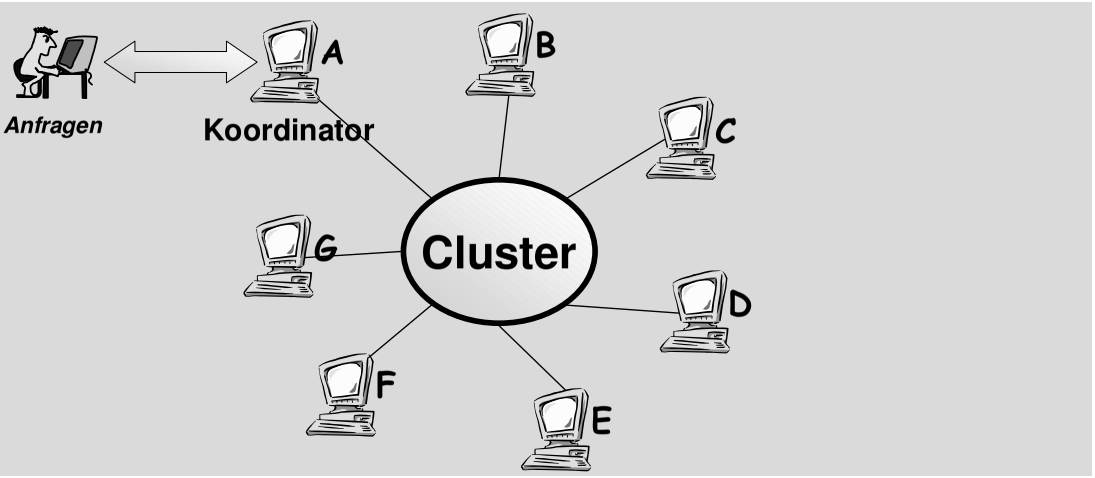
Approximierte Ähnlichkeitssuche

- Gegeben sei ein beliebiger Anfragepunkt q (siehe Grafik). Nehmen wir an, wir hätten den exakten NN zu q gefunden. Die Distanz zwischen q und seinem NN sei δ_{min}. Falls wir die NN-Sphäre um q um einen Faktor $(1+\varepsilon)$ vergrössern, mit kleinem ε, so enthält die neue Sphäre jene Punkte, welche „ungefähr“ gleich weit von q entfernt sind, wie der exakte NN. Jeder dieser Punkte käme als Approximation des NNs in Frage.
- Anpassung der Algorithmen
  - **Allgemein:** Ein Datenpunkt p kommt als Approximation des NNs in Frage, falls seine Distanz δ zum Anfragepunkt kleiner als $(1+\varepsilon)$*NN-Dist ist. D.h. falls die Distanz eines anderen Punktes nicht besser als $\delta/(1+\varepsilon)$ ist, können wir p als Approximation des NNs behalten.
  - **HS-Algorithmus:** Datenpunkte werden nicht mit deren exakten Distanz δ in die Queue eingefügt, sondern mit $\delta/(1+\varepsilon)$. Falls nun ein Datenpunkt zuoberst in der Queue liegt, bedeutet dies, dass kein anderer Datenpunkt näher als $\delta/(1+\varepsilon)$ zu q liegen kann. D.h. der HS-Algorithmus kann früher abgebrochen werden.
  - **VA-File:** Analog zur Anpassung des HS-Algorithmus werden die exakten Distanzen um den Faktor $1/(1+\varepsilon)$ angepasst. Damit kann die 2. Phase früher abgebrochen werden.

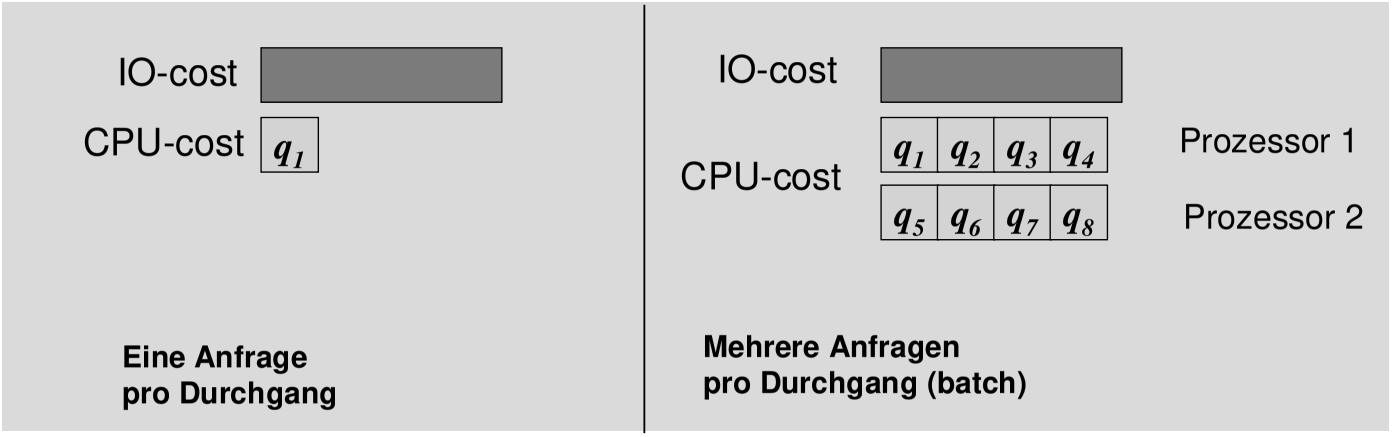


Parallele Ähnlichkeitssuche

- Falls die NN-Suche sequentiell implementiert ist (z.B. VA-File), so bietet die Verwendung eines „Cluster of Workstations“ einen nahezu linearen speedup in der Suche. Der Aufbau eines solchen Clusters ist wie folgt (siehe Grafik)
  - Jede Komponente enthält alle Indexdaten, d.h. volle Replikation. Damit kann sicher gestellt werden, dass ein Ausfall einer Komponente nicht zum Totalausfall des Systems führt.
  - Ein Koordinator empfängt die Anfragen, teilt diese auf, und verteilt sie an die Komponenten im Cluster. „Aufteilen“ bedeutet hier, dass den Komponenten disjunkte Suchbereiche im VA-File zugeordnet werden. Die Wahl der Komponenten kann aufgrund von Lastinformation geschehen.



- Eine andere Möglichkeit der Paralleliserung besteht darin, Anfragen in Gruppen auszuwerten. Wie schon mehrfach erwähnt ist die NN-Suche vor allem ein IO-Problem. Die CPU wird fast nicht benötigt (siehe Grafik, linke Seite)
  - Anstatt nur einer Anfrage, werden gleich mehrere Anfragen in einem Durchgang beantwortet (query batching). Falls die Anfragen dieselben Merkmalsdaten benötigen, so müssen die Daten nur einmal geladen werden. Die CPU wird dafür stärker belastet. Damit sinken zwar die Antwortzeiten der Anfragen nicht, aber es können mehr Anfragen pro Zeiteinheit evaluiert werden (Erhöhung des throughputs). Der throughput kann weiter erhöht werden, falls mehrere Prozessoren eingesetzt werden.



Literatur / Links (1)

- **Allgemeine Bücher zum Bildtretrieval**
  - Christos Faloutsos, **Searching Multimedia Databases by Content**, Kluwer Academic Publishers, 1996
  - Alberto Del Bimbo, **Visual Information Retrieval**, Morgan Kaufmann Publishers, 1999.
  - Michael S. Lew (Ed.), **Principles of Visual Information Retrieval**, Springer Verlagr, 2001.

- **Merkmalsextraktion**
  - Farbräume
    - Charles Poynton, **Color technology**, http://www.inforamp.net/~poynton/Poynton-color.html
  - Primitive Merkmale
    - Y. Rui, T. Huang, and S.-F. Chang, **Content-Based Image Retrieval: A Survey**, Journal of Visual Communication and Image Representation, Academic Press, March 1999.
    - M. Flickner, H. S. Sawhney, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, **Query by Image and Video Content: The QBIC System.** IEEE Computer, 28(9):23-32, Sept. 1995.
    - M. Stricker and A. Dimai. **Color Indexing with Weak Spatial Constraints.** In Storage and Retrieval for Image and Video Databases (SPIE), volume 2670 of SPIE Proceedings, pages 29-40, San Diego/La Jolla, CA, USA, Feb. 1996.
    - A. Dimai, **Invariant Scene Representations for Preattentive Similarity Assessment**, PhD. Thesis, ETH Zurich, 1999.
    - B.S. Manjunath and W.Y. Ma, **Texture Features for Browsing and Retrieval of Image Data**, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18:8, p. 837-842, 1996.
    - A. Vailaya, **Shape-Based Image Retrieval**, Master Thesis, Michigan State University, 1996.
    - S. Berchtold, D. A. Keim, and H.-P. Kriegel. **Section Coding: Ein Verfahren zur Ähnlichkeitssuche in CAD-Datenbanken.** In Datenbanksysteme in Büro, Technik und Wissenschaft, pages 152-172, Ulm, Germany, Mar. 1997. Springer.

Literatur / Links (2)

- **Merkmalsextraktion**
  - Logische Merkmale
    - David Forsyth, http://www.cs.berkeley.edu/~daf/
    - The Face Recognition Home Page: `http://www.cs.rug.nl/~peterkr/FACE/frhp.html`
    - Henry A. Rowley, Shumeet Baluja, and TakeoKanade, **Neural Network-BasedFace Detection**, IEEE Transactions on Pattern Analysis and MachineIntelligence, volume 20, number 1, pages 23-38, January 1998, http://www.cs.cmu.edu/~har/faces.html.
    - Henry A. Rowley, Shumeet Baluja, and Takeo Kanade, **Rotation Invariant Neural Network-Based Face Detection**, accepted for oral presentation at Computer Vision and Pattern Recognition, 1998,pages 38-44, http://www.cs.cmu.edu/~har/faces.html.

- **Sinn und Unsinn von NN-Suche in hoch-dimensionalen Räumen**
  - S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. **A Cost Model For Nearest Neighbour Search**. In Proceedings of the ACM Symposium on Principles of Database Systems (PODS), pages 78-86, Tucson, Arizona, USA, May 1997. ACM Press.
  - R. Weber, H.-J. Schek, and S. Blott. **A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces**. In Proceedings of the International Conference on Very Large Databases (VLDB), New York, USA, August 1998.
  - K. Beyer, J.Goldstein, R. Ramakrishnan, and U. Shaft. **When is ``nearest neighbour'' meaningful?** In Proc. 7th Int. Conf. Data Theory, ICDT, number 1540 in Lecture Notes in Computer Science, LNCS, pages 217-235. Springer-Verlag, 10-12 January 1999.
  - A. Hinneburg, C. C. Aggarwal, and D. A. Keim. **What Is the Nearest Neighbor in High Dimensional Spaces?** In Proceedings of the International Conference on Very Large Databases (VLDB), pages 506-515, Cairo, Egypt, Sept. 2000. Morgan Kaufmann.
  - C. C. Aggarwal, A. Hinneburg, and D. A. Keim. **On the surprising behavior of distance metrics in high dimensional spaces**. In Proceedings of the International Conference on Database Theory (ICDT), volume 1973 of Lecture Notes in Computer Science, pages 420-434, London, UK, Jan. 2001. Springer.

Literatur / Links (3)

- **Indexstrukturen für NN-Suche**
  - R.A. Finkel and J.L. Bentley. **Quad-trees: A data structure for retrieval on composite keys**. ACTA Informatica, 4(1):1-9, 1974
  - J.T. Robinson. **The k-d-b-tree: A search structure for large multidimensional dynamic indexes**. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 10-18, 1981.
  - J. Nievergelt, H. Hinterberger, and K.C. Sevcik. **The grid file: An adaptable symmetric multikey file structure**. ACM Transactions on Database Systems, 9(1):38-71, March 1984.
  - J. A. Orenstein and T. H. Merrett. **A Class of Data Structures for Associative Searching**. In Proceedings of the ACM Symposium on Principles of Database Systems (PODS), pages 181--190, Waterloo, Ontario, Canada, Apr. 1984. ACM.
  - A.Guttman. **R-Trees: A dynamic index structure for spatial searching**. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 47-57, Boston, MA, June 1984.
  - T. Sellis, N. Roussopoulos, and C. Faloustos. **The R+-tree: A dynamic index for multi-dimensional objects**. In Proceedings of the International Conference on Very Large Databases (VLDB), pages 507-518, Brighton, England, 1987.
  - N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. **The R*-tree: An efficient and robust access method for points and rectangles**. In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, pages 322-331, Atlantic City, NJ, 23-25 May 1990.
  - H. V. Jagadish. **Spatial Search with Polyhedra**. In Proceedings of the International Conference on Data Engineering (ICDE), pages 311--319, Los Angeles, California, USA, Feb. 1990. IEEE Computer Society.
  - K.-I. Lin, H.V. Jagadish, and C. Faloutsos. **The TV-tree: An index structure for high-dimensional data**. The VLDB Journal: The International Journal on Very Large Data Bases, 3(4):517-549, October 1994.
  - Tzi-cker Chiueh. **Content-Based Image Indexing (vp-Tree)**. In Proceedings of the Twentieth International Conference on Very Large Databases, pages 582--593, Santiago, Chile, 1994.
  - S. Berchtold, D.A. Keim, and H.-P. Kriegel. **The X-tree: An index structure for high-dimensional data**. In Proceedings of the International Conference on Very Large Databases (VLDB), pages 28-39, 1996.
  - David A. White, Ramesh Jain: **Similarity Indexing with the SS-tree.** ICDE 1996: 516-523

Literatur / Links (4)

- **Indexstrukturen für NN-Suche**
  - R. Kurniawati, J. S. Jin, and J. Shepherd. **SS+-Tree: An Improved Index Structure for Similarity Searches in a High-Dimensional Feature Space**. In Storage and Retrieval for Image and Video Databases (SPIE), volume 3022 of SPIE Proceedings, pages 110--120, San Jose, CA, USA, Feb. 1997.
  - N. Katayama and S. Satoh. **The SR-tree: An index structure for high-dimensional nearest neighbor queries**. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 369-380, Tucson, Arizon USA, 1997.
  - P. Ciaccia, M. Patella, and P. Zezula. **M-tree: An efficient access method for similarity search in metric spaces**. In Proceedings of the International Conference on Very Large Databases (VLDB), Greece, 1997.
  - C. Böhm and H.-P. Kriegel. **Dynamically Optimizing High-Dimensional Index Structures**. In Proceedings of the International Conference on Extending Database Technology, volume 1777 of Lecture Notes in Computer Science, pages 36--50, Konstanz, Germany, Mar. 2000. Springer-Verlag.
  - J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. **Generalized Search Trees for Database Systems (GiST)**. In Proceedings of the International Conference on Very Large Databases (VLDB), pages 562-573, Zurich, Switzerland, Sept. 1995. Morgan Kaufmann.
  - Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegel. **The pyramid-technique: Towards breaking the curse of dimensionality**. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 142-153, Seattle, WA, USA, 1998
  - J. Goldstein and R. Ramakrishnan. **Contrast Plots and P-Sphere Trees: Space vs. Time in NN Searches**. In Proceedings of the International Conference on Very Large Databases (VLDB), pages 429--440, Cairo, Egypt, Sept. 2000. Morgan Kaufmann.

Literatur / Links (5)

- **VA-File**
  - Roger Weber, Hans-J. Schek and Stephen Blott. **A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces**. In Proceedings of the International Conference on Very Large Databases (VLDB), New York, USA, August 1998.
  - Roger Weber, Klemens Böhm. **Trading Quality for Time with Nearest-Neighbor Search**. In VII. Conference on Extending Database Technology (EDBT'2000), Konstanz, Germany, March 27-31 2000.
  - Roger Weber, Klemens Böhm, and Hans-J. Schek. **Interactive-Time Similarity Search for Large Image Collections Using Parallel VA-Files**. In 16th International Conference on Data Engineering (ICDE'2000), San Diego, CA, USA, February 29 - March 3, 2000.
  - Roger Weber, **Similarity Search in High-Dimensional Vector Spaces**, PhD. Thesis, ETH Zurich, 1999.

- **Komplexe Anfragen**
  - P. Ciaccia, M. Patella, and P. Zezula. Processing Complex Similarity Queries with Distance-Based Access Methods. In Proceedings of the International Conference on Extending Database Technology, volume 1377 of Lecture Notes in Computer Science, pages 9-23, Valencia, Spain, Mar. 1998. Springer.
  - R. Fagin. Combining Fuzzy Information from Multiple Systems. In Proceedings of the ACM Symposium on Principles of Database Systems (PODS), pages 216-226, Montreal, Canada, June 1996. ACM Press.
  - Surajit Chaudhuri, Luis Gravano. **Optimizing Queries over Multimedia Repositories**, SIGMOD Conf. 1996, pages 91-102.
  - Surajit Chaudhuri,Kyuseok Shim. **Optimization of Queries with User-defined Predicates**, VLDB 1996, pages 87-98.
  - M. Mlivoncic, K. Böhm, H.-J. Schek, R. Weber: **Fast Evaluation Techniques for Complex Similarity Queries**. 27th Int. Conf. on Very Large Databases (VLDB), Roma, Italy, September 2001.

Literatur / Links (6)

- **Bildsuchsysteme (ausgewählte)**
  - Übersichtsseite
    - http://cui.unige.ch/~viper/other_systems.html
  - Altavista
    - http://www.altavista.com/cgi-bin/query?pg=q&stype=simage
  - WebSeek, Columiba University
    - http://www.ctr.columbia.edu/webseek/
  - Chariot, ETH Zürich
    - http://simulant.ethz.ch/Chariot/
  - BlobWorld, Berkley University
    - http://www.cs.berkeley.edu/~carson/blobworld/
    - http://dlp.cs.berkeley.edu/photos/blobworld/animals.html
  - Semantics-sensitive Integrated Matching for Picture Libraries, Stanford University
    - http://wang.ist.psu.edu/cgi-bin/zwang/regionsearch_show.cgi
  - Face Detection Demo, Carnegie Mellon University
    - http://www.vasc.ri.cmu.edu/IUS/usrp0/har/FaceDemo/
  - QBIC, IBM
    - Briefmarken: http://wwwqbic.almaden.ibm.com/cgi-bin/stamps-demo/
    - QBIC-Museum: http://www.hermitagemuseum.org/fcgi-bin/db2www/qbicSearch.mac/qbic?selLang=English