George Mason University | Computational and Data Sciences

CSI 695: Scientific Databases

Fall Term 2017

Lecture 6: Similarity Search Algorithms

Lectures: Prof. Dr. Matthias Renz

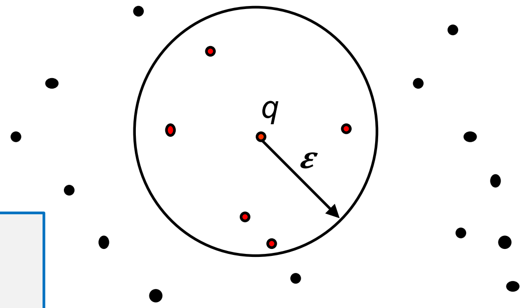
Exercises: TBA

Similarity Search Algorithms: Outline

- Range Query Algorithms
- (k)-Nearest Neighbor Query Algorithms
- Reverse (k)-Nearest Neighbor Query Algorithms
- Skyline Query Algorithms
- Evaluation of Similarity Search Methods

Similarity Search Algorithms: Range Query

- Range Query (RQ)
  - Definition
    - Properties:
      - User defines query object q and a distance range $\varepsilon \in \mathbb{R}_0^+$
      - The result of a range query RQ(DB) contains all objects in database DB, having a distance to q of at most ε.
    - Formal:
      - $RQ(DB,q,\varepsilon) = \{o \in DB \mid dist(q,o) \leq \varepsilon\}$

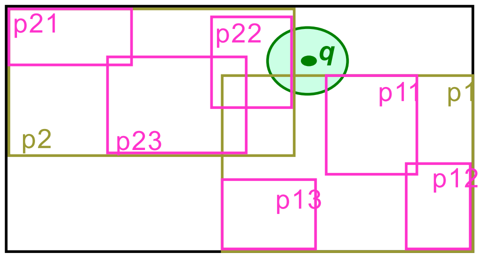


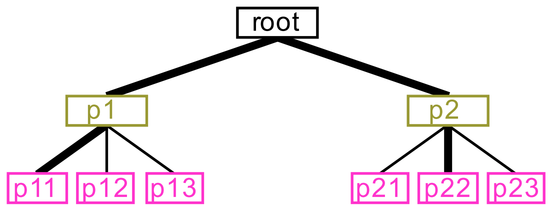
  - Basic Algorithm (sequential scan)

```
RQ-SeqScan(DB,q,ε)
        result = ∅;
        FOR i=1 TO n DO
                IF dist(q,DB.getObject(i)) ≦ ε THEN
                        result := result ∪ getObject(i);
        RETURN result;
```

Similarity Search Algorithms: Range Query

- Range Query (RQ) (cont.)
  - Index-based Algorithm
    - How can we support the search in an efficient way using a spatial index (e.g. R-tree)?
    - Observation and basic idea:
      - Using spatial keys (page regions) to guide the search.
      - Every region that intersect the query range could contain candidates
      - Start at the root, and access all children where the corresponding regions intersect the query range
      - Repeat the last step for all accessed nodes recursively

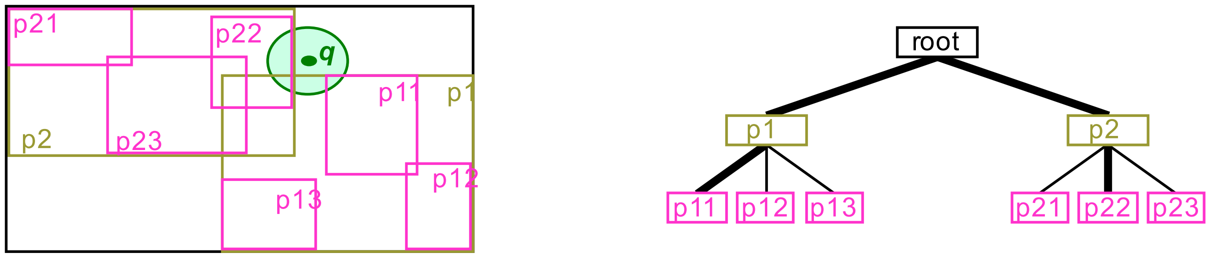




Similarity Search Algorithms: Range Query

- Range Query (RQ) (cont.)
  - Index-based Algorithm: Depth-first-search

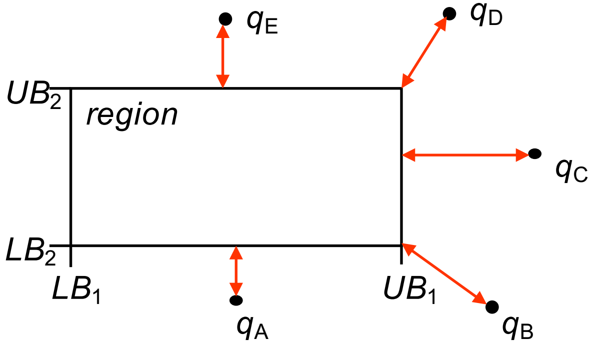
```
RQ-Index(pa,q,ε)   // pa := Disk address e.g. root of index (R-tree)
   result = ∅;
   p := pa.loadPage();
   IF p.isDataPage() THEN
      FOR i=0 TO p.size() DO
         IF dist(q, p.getObject(i)) ≦ ε THEN
            result := result ∪ getObject(i);
   ELSE     // p is index directory page
      FOR i=0 TO p.size() DO
         IF MINDIST(q,p.getRegion(i)) ≦ ε THEN
            result := result ∪ RQ-Index(p.childPage(i),q,ε);
   RETURN result;
```



Similarity Search Algorithms: Range Query

- Range Query (RQ) (cont.)
  - What is the MINDIST(..) function?
    - Used to test if an index page region intersects with the query range.
    - MINDIST() is the minimal distance between the query object and all objects covered by the rectangular page region (lower bound distance).

$$\mathrm{MINDIST}(region,q) = \sqrt{\sum_{0<i\le d} \begin{cases} (region.LB_i - q_i)^2 & \text{if} \quad q_i \le region.LB_i \\ 0 & \text{if} \quad region.LB_i \le q_i \le region.UB_i \\ (q_i - region.UB_i)^2 & \text{if} \quad region.UB_i \le q_i \end{cases}}$$



    - In other words: For all o ∈ region: MINDIST(region,q) ≦ dist(o,q)
    - Consequence: For a RQ(DB,q,ε): if MINDIST(region,q)>ε
      => For all o ∈ region: dist(o,q)>ε
      => there is no candidate in *region* !

Similarity Search Algorithms: Range Query

- Range Query (RQ) (cont.)
  - Multi-Step Query Processing Algorithm
    - How can we support the search in an efficient way using a filter-refinement strategy?
    - Observation and basic idea:
      - Assume we can compute lower-bounding (LB) and upper-bounding (UB) filter distances between objects in an efficient way, s.t.: For all q,o $\in$ DB: LB-dist(q,o) $\leqq$ dist(q,o) $\leqq$ UB-dist(q,o) holds.
      - Basic idea is to scan the database by applying the two filter distances (LB-dist and UB-dist) to filter out results (hits) and non-results (drops).
      - Identify a drop (non result) by LB-dist: If LB-dist(q,o) > ε, then o can't be a result => drop o.
      - Identify a hit (result) by UB-dist: If UB-dist(q,o) $\leqq$ ε, then o is definitely a result => report o as part of the result.
      - All remaining candidates have to be refined, i.e. compute the exact distance dist(q,o) and check against ε to finalize the result.

Similarity Search Algorithms: Range Query

- Range Query (RQ) (cont.)
  - Multi-Step Query Processing Algorithm (cont.)
    - Lower bounding filter distance LB-dist (LB), Upper bounding filter distance UB-dist (UB)

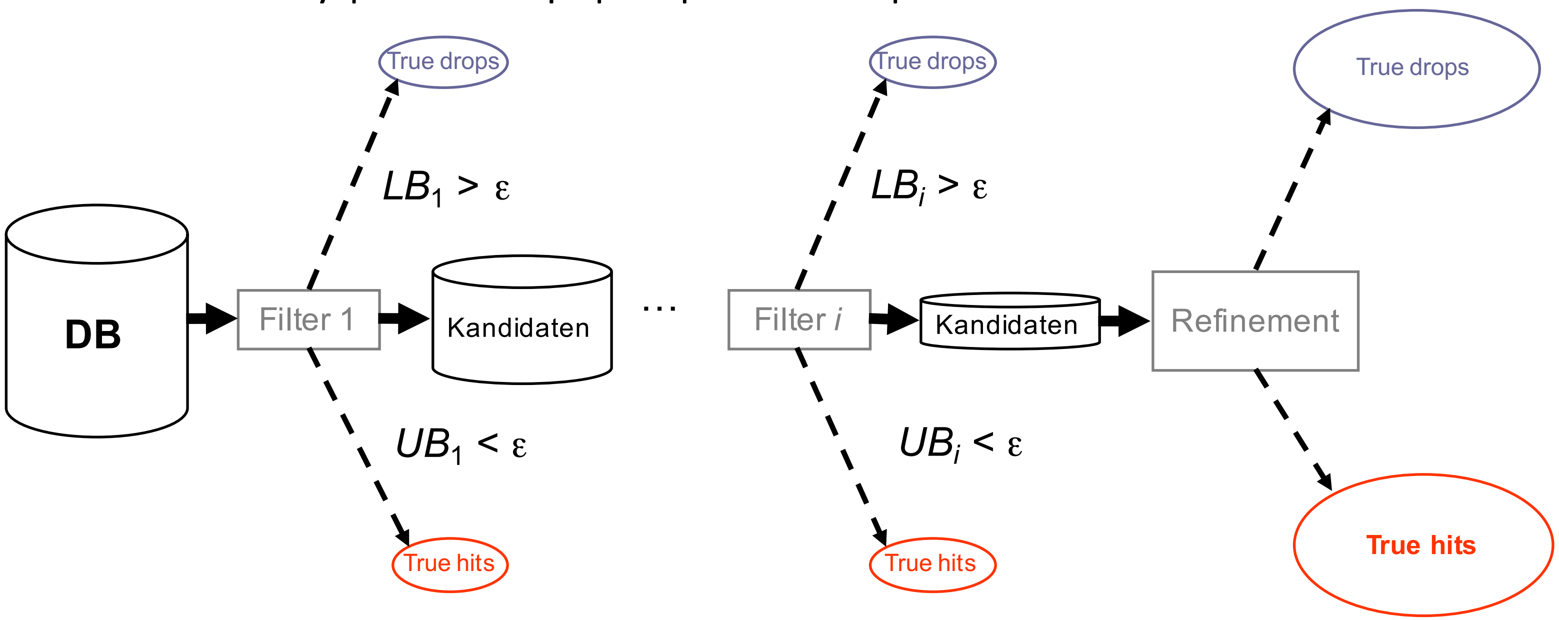
```
RQ-MultiStep(DB,q,ε)
   result = ∅;
   candidates = ∅;

   // Filter
   FOR i=1 TO n DO
      IF UB(q,DB.getObject(i)) ≦ ε THEN
         result := result ∪ getObject(i);
      ELSE IF LB(q, DB.getObject(i)) ≦ ε THEN
         candidates := candidates ∪ getObject(i);

   // Refinement
   FOR i=1 TO candiates.size() DO
      IF dist(q, candidates.getObject(i)) ≦ ε THEN
         result := result ∪ candidates.getObject(i);
   RETURN result;
```

Similarity Search Algorithms: Range Query

- Range Query (RQ) (cont.)
  - Multi-Step Query Processing Algorithm (cont.)
    - Often only lower bounding filter distance (LB-dist) is used, because usually |# true drops| >> |# true hits|



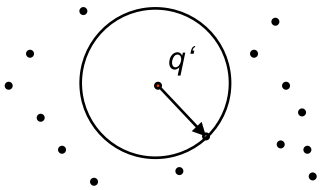
Similarity Search Algorithms: Outline

- Range Query Algorithms
- (k)-Nearest Neighbor Query Algorithms
- Reverse (k)-Nearest Neighbor Query Algorithms
- Skyline Query Algorithms
- Evaluation of Similarity Search Methods

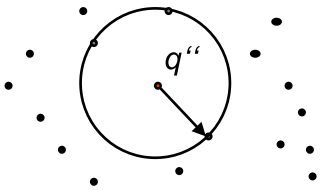
Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ)
  - Definition
    - Properties:
      - User defines query object q
      - The result is the object (or objects) in database DB, having the smallest distance to q.
      - Ambiguities have been resolved approriately.
  - Formal:

$$NN(q) = \{o \in DB \mid \forall x \in DB : dist(q,o) \le dist(q,x)\}$$



unique result



ambiguous result

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Basic Algorithm (sequential scan): non-deterministic

```
NN-SeqScan(DB,q)
   result = ∅;
   stopdist = +∞;
   FOR i=1 TO n DO
      IF dist(q,DB.getObject(i)) ≦ stopdist THEN
         result := getObject(i);
         stopdist = dist(q,DB.getObject(i));
   RETURN result;
```

- Deterministic- vs. non-deterministic NNQ
  - **Deterministic**: Query produces always the same result regardless of the order the objects are accessed.
  - **Non-deterministic**: Query produces a correct result, but the result depends on the order the objects are accessed.

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Algorithm with spatial index: simple depth-first search
    - Difference to Range Query (RQ):
      - Nearest neighbor can be arbitrarily far away from the query object.
      - Shape of the query region unknown.
      - A (single) page region does not suffice to make decisions about potential coverage of a candidate.
      - The need to access a page depends on the content of other pages or objects.
      - As soon as the distance to the nearest neighbor (NN-dist) of query object q is known, the query can be processed as range query.
      - The distance from query object q to any object $o \in DB$ can be used to upper bound the NN-dist.
      - If more distances between q to other objects are known, the smallest can be used as better NN-dist approximation.

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Algorithm with spatial index: simple depth-first search
    - NNQ-Algorithm: Reformulation of the RQ-Algorithm
      - Idea: Use smallest found distance to any object o∈DB as distance range ε.

```
global variable: stopdist = +∞;
NN-Index-Simple-DS(pa,q) // pa := Disk address e.g. root of index (R-tree)
   result = ∅;
   p := pa.loadPage();
   IF p.isDataPage() THEN
      FOR i=0 TO p.size() DO
         IF dist(q,p.getObject(i)) ≦ stopdist THEN
            result := getObject(i);
            stopdist = dist(q,p.getObject(i));
   ELSE      // p is directory page
      FOR i=0 TO p.size() DO
         IF MINDIST(q,p.getRegion(i)) ≦ stopdist THEN
            result := NN-Index-Simple-TS(p.childPage(i),q);
   RETURN result;
```

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Algorithm with spatial index: simple depth-first search
  - Weakness of simple depth-first search algorithm:
    - Initialization: stopdist = $+\infty$
    - Search starts with arbitrary path in the index tree
    - First accessed object(s) can be very far away from the query object
      => filter by stopdist is not selective
    - Better approach:
      - Use initial search path that is close to the query point
      - Access pages having a high probability that they contain the nearest neighbor to q
      - Instead of depth-first tree traversal, allow to switch to more promising search pathes during the tree traversal

    => Traversing index by best-first search

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Algorithm with spatial index: Priority-Based Search [Hjaltason, Samet, SSD 1995]
    - Instead of a recursive traversal an active page list (APL) is managed
    - A page (node) P is active if
      - P is not yet visited
      - A parent page of P has been visited
      - MINDIST(q, P) ≤ stopdist
    - APL is initialized with the root of the index tree
    - Pages in APL are sorted by increasing MINDIST to the query
    - At each step, the first entry in APL (page with smallest MINDIST) is processed

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Algorithm with spatial index: Priority-Based Search [Hjaltason, Samet, SSD 1995]
    - Leaf nodes: update the value of stopdist and keep potential hits.
    - Directory nodes: child nodes with MINDIST ≤ stopdist are inserted into APL.
    - If the value of stopdist is updated (decreased), pages with MINDIST > stopdist in APL can be ignored.

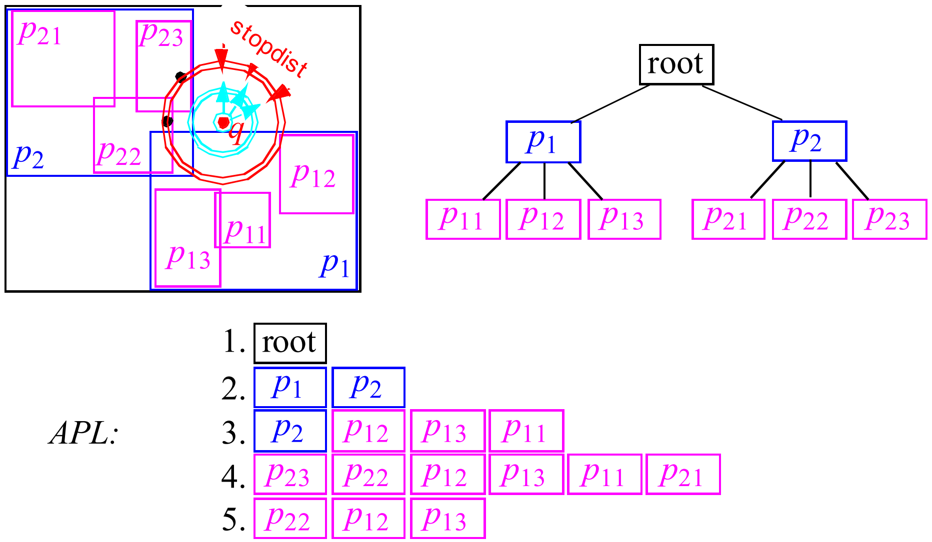
Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Algorithm with spatial index: Priority-Based Search [Hjaltason, Samet, SSD 1995]

```
Global variable: stopdist = +∞;
NN-Index-HS(pa,q)        // pa = Disk adress e.g. the root of the index
   result = ∅;
   apl = LIST OF (dist:Real, da:DiskAdress) ORDERED BY dist ASCENDING
   apl = [(0.0, pa)]
   WHILE NOT apl.isEmpty() AND apl.first().dist ≦ stopdist DO
      p := apl.getFirst().da.loadPage();
      apl.deleteFirst();
      IF p.isDataPage() THEN
           (* processed as in NN-Index-Simple-DS(pa,q) *)
      ELSE      // p is directory page
         FOR i=0 TO p.size() DO
            IF MINDIST(q,p.getRegion(i)) ≦ stopdist THEN
               apl.insert(MINDIST(q, p.getRegion(i)), p.childPage(i));
   RETURN result;
```

Similarity Search Algorithms: Nearest Neighbor Query

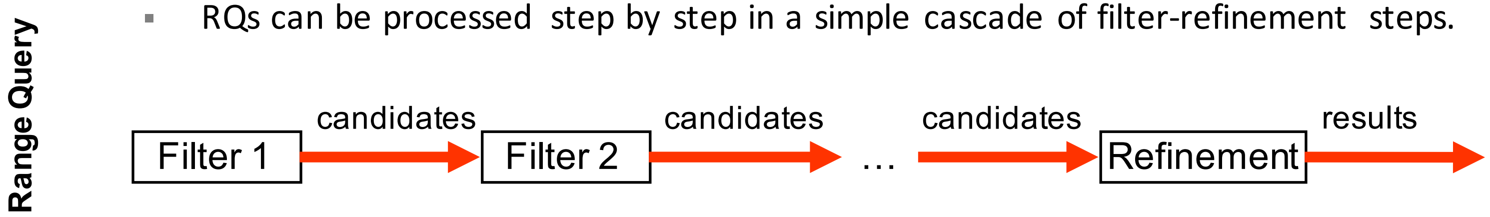
- Nearest Neighbor Queries (NNQ) (cont.)
  - Algorithm with spatial index: Priority-Based Search [Hjaltason, Samet, SSD 1995]
  - Example



  - The priority-based NN-Index-HS algorithm is optimal in the number of page accesses.

Similarity Search Algorithms: Nearest Neighbor Query

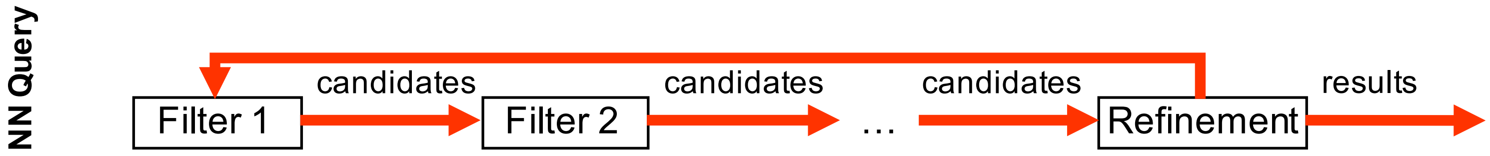
- Nearest Neighbor Queries (NNQ) (cont.)
  - Multi-step NNQ Algorithm
    - Principles:
      - Algorithms often only use lower-bounding (LB) filter
      - Using multiple filter steps: distLB1≦distLB2≦...
      - Difference to Range Queries (RQ):
        - RQs can be processed step by step in a simple cascade of filter-refinement steps.



        - Not possible with NNQs. For the first filter-step, NNQs needs feedback from the last step (refinement) to prune (reject) candidates while conserving the exact results.
        - With an appropriate filter distance, it is likely that the first candidates contain the exact nearest neighbor (NN).

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Multi-step NNQ Algorithm
    - Principles (cont.):
      - Filter-Refinement Feedback: Feedback with the refined distances from the refinement step to the first filter steps.
      - The filter-refinement cascade will be processed in a loop.



    - In the following:
      - Different query processing strategies
      - Here, we will consider just one filter step (easy transfer to multiple filter steps possible)

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Multi-step NNQ Algorithm
    - NNQ with Range Query
      [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. Proc. Int. Conf. Very Large Databases (VLDB), 1996]
      [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. TKDE 10(6), 1998]
    - Idea:
      - Refinement distance ε of an arbitrary object serves as upper bounding NN distance.
      - Consequence: Is object $p \in NNQ(q)$ => $dist(p,q) \leqq \varepsilon$ and $distLB(p,q) \leqq \varepsilon$, i.e. $p \in RQ(q,\varepsilon)$.
      - The smaller the initial distance ε, the better the query performance in the second filter-refinement round.
      - The nearest neighbor of q based on distLB usually provides a good distance ε.

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Multi-step NNQ Algorithm
    - NNQ with Range Query
      [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. Proc. Int. Conf. Very Large Databases (VLDB), 1996]
      [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. TKDE 10(6), 1998]
    - Principle:
      1. Perform an NN query based on the (lower bounding) filter distance
      2. The resulting object o will be refined and ε:= dist(q,o)
      3. Perform a range query RQLB(q,ε) based on the filter distance distLB(q,.)
      4. Refine the distances of all objects reported by RQLB(q,ε)
      5. Report the object with the smalles refined distance to q as result

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Multi-step NNQ Algorithm
    - Algorithm:

```
NN-MultiStep-RQ(DB,q)
   // (first) filter step
   r = NN-Query based on filter distanz;    // with or without index
   ε = dist(q,r);                           // refinement of object r
   candidates = RQ-MultiStep(DB,q,ε);

   // Refinement
   result = r ;
   stopdist = ε;
   FOR EACH p∈candidates DO
      IF dist(p,q) ≦ stopdist THEN
         stopdist = dist(q,p);
         result = p;
   RETURN result;
```

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Multi-step NNQ Algorithm
    - NNQ with Range Query (cont.)
      - Pros:
        - Very simple algorithm (simple implementation and integration)
      - Cons:
        - Performance highly depends on the filter selectivity:
          weak filter => large ε => large result set of RQLB => high refinement cost
      - Can we do better ?
        - Main problem is that the filter is based on the refinement result of just one object sample (first object retrieved by the first filter)
        - Basic idea: Use the result of each refined object to improve the filter step-by-step

          => multiple filter-refinement iterations

          => Apply filter after each refinement

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Multi-step NNQ Algorithm
    - Priority-based NNQ
      [Seidl, Kriegel. Proc. ACM Int. Conf. Management of Data (SIGMOD),1998]
      - Perform "Ranking Query" based on the filter distance distLB(q,.)
        - Function getNext() reports the first nearest neighbor to q with the first call, the second one with the second call, etc.
        - Start with the first call of getNext()
        - Refine each reported object immediately and setup stopdist (analog to ε value) with the smallest exact distance found so far.
        - Repeat getNext() calls + immediate refinement (see two steps above) as long as the filter distance distLB(q,o) of the reported object o is below or equal stopdist.
      - Priority-based NNQ is optimal w.r.t. the number of refinements.

Similarity Search Algorithms: Nearest Neighbor Query

- Nearest Neighbor Queries (NNQ) (cont.)
  - Multi-step NNQ Algorithm
    - Algorithm

```
NN-MultiStep-Optimal(DB,q)
   Ranking = initialize ranking query w.r.t. q based on distLB
   result = ∅;
   stopdist = +∞;
   REPEAT
      p = Ranking.getNext();
      filterdist = distLB(p,q);                // filter step
      IF filterdist ≦ stopdist THEN
         IF dist(q,p) ≦ stopdist THEN          // refinement step
            result = p;
            stopdist = dist(q,p);
   UNTIL filterdist > stopdist;
   RETURN result;
```

Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor Queries
  - Cost criterions:
    - Cost for accessing index pages (secondary storage accesses): I/O cost
    - Cost for computing exact distances (refinement): CPU cost
  - For multi-step query processing strategies, we are more interested in reducing the CPU cost.
  - Generally: The cost in a multi-step query processing approach mainly depends on the selectivity of the filter (filter steps)

    Higher filter selectivity → less candidates to be refined → less objects have to be refined (maybe also less I/O cost)
  - The more information of objects we use in the filter, the higher the selectivity of the filter, but also the higher the cost of the filter itself.

Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor Queries
  - Basic idea:
    - Use a little bit more (already available and cost wise easy to get) information for the filter step to reduce the candidates.
    - In addition to the lower bounding filter distance, use the upper bounding filter distance.
    - Constraints: Upper bounding filter distance can only be applied for k-NN queries with k>1. WHY?