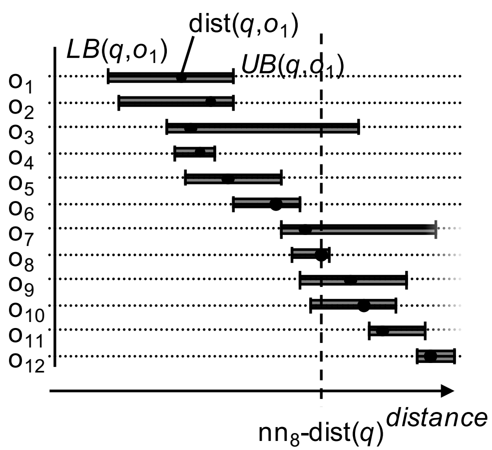
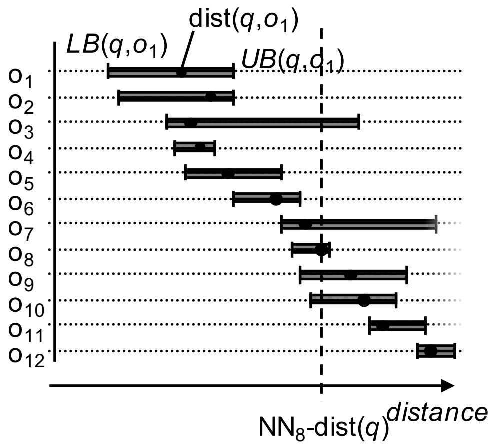
Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor Queries
  - How can the upper bounding filter distance be applied in the following example?
    - Given: 12 objects $o_1,..,o_{12}$, with lower/upper-bounding filter distances, respectively.
    - Query: 8-NN Query to be answered using multi-step query processing paradigm.
    - Which objects definitely have to be refined to answer the query?



Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor Queries
  - How can the upper bounding filter distance be applied in the following example?
    - Given: 12 objects $o_1,..,o_{12}$, with lower/upper-bounding filter distances (LB,UB), respectively.
    - Query: 8-NN Query to be answered using multi-step query processing paradigm.
    - Question: Which objects definitely have to be refined to answer the query?



Answer: All objects o_i have to be refined for which the following holds:

$LB(q,o_i) \leqq NN_8\text{-dist}(q) \leqq UB(q,o_i)$,

where $LB(q,o_i)<UB(q,o_i)$, and $NN_8\text{-dist}(q)$ is the distance between q and the 8^{th}-NN of q.

Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor (k-NN) Queries
  - Definition:
    A Multi-Step k-NN Query algorithm is called refinement optimal, if it only refines the set of objects $\{oi \in DB \mid LB(q,o_i) \leqq NN_k\text{-dist}(q) \leqq UB(q,o_i)\}$.
  - Problem: Distance NN_k-dist(q) not known before starting the refinement!!!
  - Can we find a refinement optimal algorithm without knowing NN_k-dist(q)? Yes we can!!!
  - For k=1, the existing algorithm NN-MultiStep-Optimal(DB,q) is already refinement optimal.
  - Now, let us explore the existing Multi-step NNQ Algorithm „NN-MultiStep-Optimal(DB,q)“ which we can easily adapt for k-NN queries (k>1).

Similarity Search Algorithms: Nearest Neighbor Query

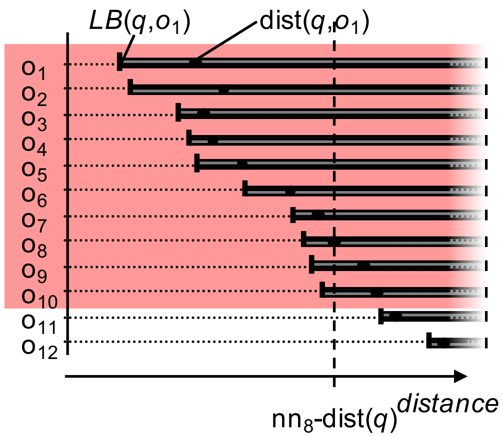
- Refinement Optimal Multi-Step (k)-Nearest Neighbor (k-NN) Queries
  - k-NN-MultiStep-Optimal(DB,q) adapted for k-NN queries (k>1):

```
k-NN-MultiStep(DB,q,k)
   Ranking = initialize ranking query w.r.t. q based on distLB
   result =  initilize heap of size k where o = result.first  is the
             element in the heap having the largest distance dist(q,o)  to q;
   stopdist = +∞;
   FOR i = 1..k DO
      p = Ranking.getNext();
      compute dist(q,p);
      result.add(p);
   REPEAT
      p = Ranking.getNext();
      filterdist = distLB(p,q);             // filter step
      IF filterdist ≦ stopdist THEN
         IF dist(q,p)  ≦ stopdist THEN      // refinement step
            remove o = result.first from result;
            result.add = p;
            stopdist = dist(q,result.first);
   UNTIL filterdist > stopdist;
   RETURN result;
```

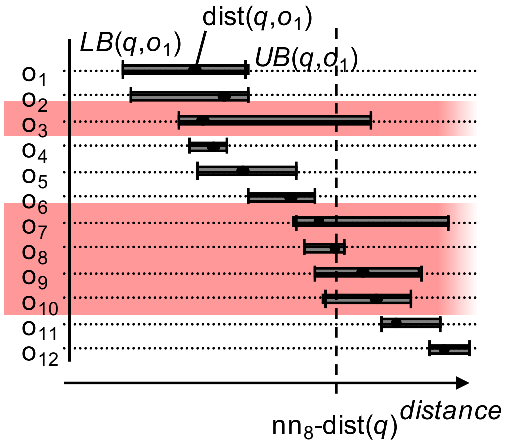
Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor (k-NN) Queries
  - k-NN-MultiStep-Optimal(DB,q) adapted for k-NN queries (k>1):

Objects refined with k-NN-MultiStep():



Objects refined with a refinement optimal solution:



- k-NN-MultiStep() is not refinement optimal.

Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor (k-NN) Queries
  - Idea of a refinement optimal k-NN multi-step query algorithm:
    - Iteratively, refine only an object o if it really needs to be refined, i.e. if $LB(q,o) \leqq NN_k\text{-}dist(q) \leqq UB(q,o)$.
    - Though NN_k-dist(q) is not known, we can conservatively approximate NN_k-dist(q) as follows:
      LB_k = k-th smallest LB(q,o) distance among all objects in DB.
      UB_k = k-th smallest UB(q,o) distance among all objects in DB.

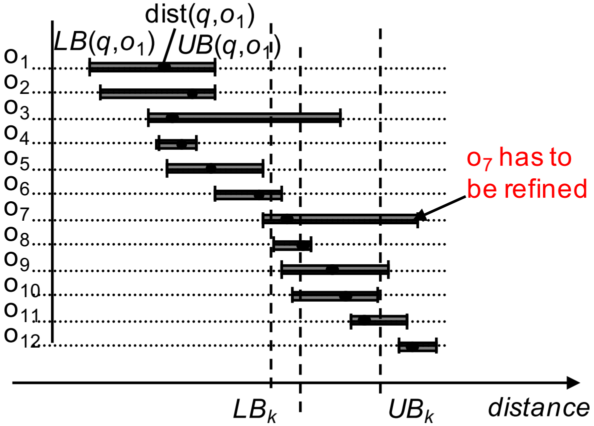
      $$LB_k \leqq NN_k\text{-}dist(q) \leqq UB_k.$$

    - LB_k and UB_k can be easily computed wihtout any refinement.
    - Refine only objects o where

      $$LB(q,o) \leqq LB_k \leqq NN_k\text{-}dist(q) \leqq UB_k \leqq UB(q,o)$$

      holds.

Example: k = 8

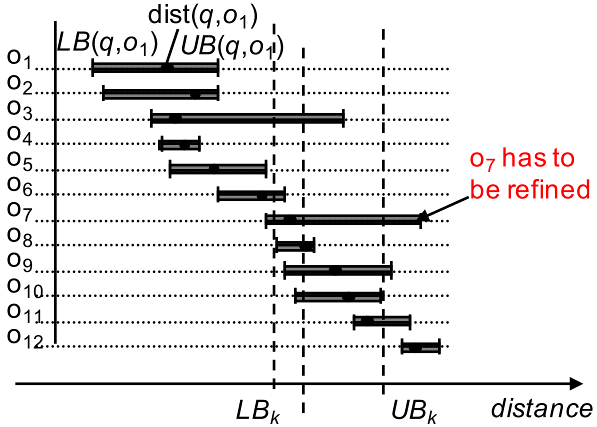


Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor (k-NN) Queries
  - Idea of a refinement optimal k-NN multi-step query algorithm:
    - Iteratively, refine only an object o if it really needs to be refined, i.e. if $LB(q,o) \leqq NN_k\text{-dist}(q) \leqq UB(q,o)$.
    - Though NN_k-dist(q) is not known, we can conservatively approximate NN_k-dist(q) as follows:
      LB_k = k-th smallest LB(q,o) distance among all objects in DB.
      UB_k = k-th smallest UB(q,o) distance among all objects in DB.

      $$LB_k \leqq NN_k\text{-dist}(q) \leqq UB_k.$$
    - LB_k and UB_k can be easily computed wihtout any refinement.
    - Refine only objects o where
      $LB(q,o) \leqq LB_k \leqq NN_k\text{-dist}(q) \leqq UB_k \leqq UB(q,o)$
      holds.
      Note: LB(q,o)=UB(q,o):=dist(q,o) as soon as o has been refined!!!

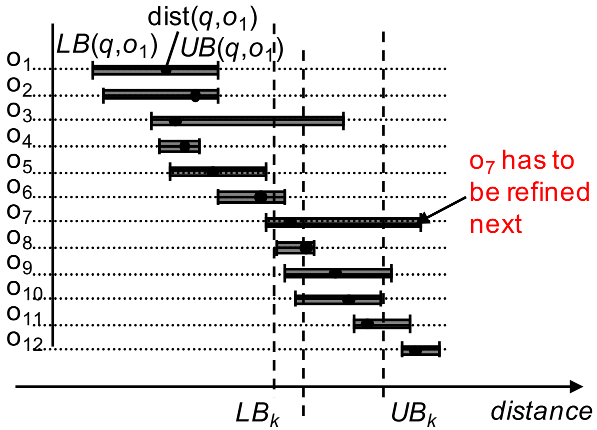
Example: k = 8



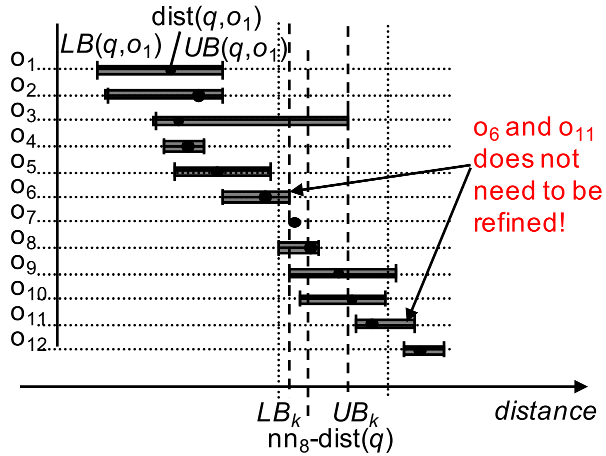
Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor (k-NN) Queries
  - Example of iterative refinement: k=8

Before first refinement



After first refinement



  - After each refinement, LB_k and UB_k have to be updated.
  - After first refinement, objects o_3 and o_9 have to be refined.

Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor (k-NN) Queries
  - Algorithm: k-NN-MultiStep-Optimal(DB, q)

    [Kriegel, Kröger, Kunath, Renz: Generalizing the Optimality of Multi-step k-Nearest Neighbor Query Processing, SSTD 2007]

k-NN-MultiStep-Optimal(DB, *q*)

Ranking = initialize ranking query on DB regarding LB fislter distance; // similar to k-NN-MultiStep

result = ∅; *candidates* = fetch first *k* Objekte from Ranking; initialize UB_k, LB_k from *candidates*;

REPEAT

// Step 1: fetch next candidate

if $LB_{next} \leq LB_k$ then // LB_{next} = LB(q,o_{next}), where o_{next} = next object in Ranking

p = Ranking.getNext();

add *p* to *candidates;*

endif;

update LB_k, UB_k, LB_{next};

// Step 2: Filter true hits and true drops (Filter Step)

for each $c \in$ *candidates* do

if UB(q,c) ≤ LB_k then remove *c* from *candidates* and add c to *result*; // true hit

if LB(q,c) > UB_k then remove *c* from *candidates;* // true drop

end for;

// Step 3: Refine next candidates (Refinement Step)

if |*result*|+|*candidates*| ≤ *k* und LB_{next} > UB_k then

add all $c \in$ *candidates* to *result*; // Stop criterion

else

refine all $c \in$ candidates, where LB(q,c)≤LB_k≤UB_k≤UB(q,c), i.e. compute dist(q,c) and update LB(q,c) = UB(q,c) = dist(q,c);

end if;

UNTIL(|*candidates*|=0 and LB_{next} > UB_k);

RETURN *result*;

Similarity Search Algorithms: Nearest Neighbor Query

- Refinement Optimal Multi-Step (k)-Nearest Neighbor (k-NN) Queries
  - Algorithm: k-NN-MultiStep-Optimal(DB, q)
    [Kriegel, Kröger, Kunath, Renz: Generalizing the Optimality of Multi-step k-Nearest Neighbor Query Processing, SSTD 2007]

```
k-NN-MultiStep-Optimal(DB, q)
  Ranking = initialize ranking query on DB regarding LB fislter distance;   // similar to k-NN-MultiStep
  result = ∅; candidates = fetch first k Objekte from Ranking; initialize UB_k, LB_k from candidates;
  REPEAT
    // Step 1: fetch next candidate
    if LB_next ≤ LB_k then   // LB_next = LB(q,o_next), where o_next = next object in Ranking
          p = Ranking.getNext();
          add p to candidates;
    endif;
    update LB_k, UB_k, LB_next;
    // Step 2: Filter true hits and true drops (Filter Step)
    for each c ∈ candidates do
          if UB(q,c) ≤ LB_k then remove c from candidates and add c to result;  // true hit
          if LB(q,c) > UB_k then remove c from candidates;  // true drop
     end for;
    // Step 3: Refine next candidates (Refinement Step)
    if |result|+|candidates| ≤ k und LB_next > UB_k then
          add all c ∈ candidates to result;  // Stop criterion
       else
          refine all c ∈ candidates, where LB(q,c)≤LB_k≤UB_k≤UB(q,c), i.e. compute dist(q,c) and update LB(q,c) = UB(q,c) = dist(q,c);
      end if;
  UNTIL(|candidates|=0 and LB_next > UB_k);
  RETURN result;
```

  - It can be proofed that in each iteration (as long as the final result has not been determined, there is at least one (unrefined) object o that meets the requirements being refined next, i.e. LB(q,o)≤LB_k≤UB_k≤UB(q,o),