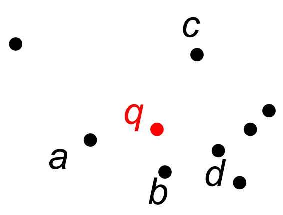
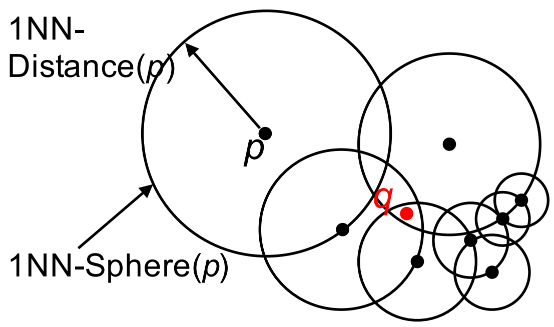
Similarity Search Algorithms: Outline

- 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: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Definition
    - Properties:
      - User defines query object q, and k
      - The result contains all objects o in the database DB, where q∈kNNQ(o,DB).
      - Ambiguities have been resolved approriately.
  - Formal: $RkNN(q,k) = \{o \in DB |\ q \in kNN(o,k)\}$





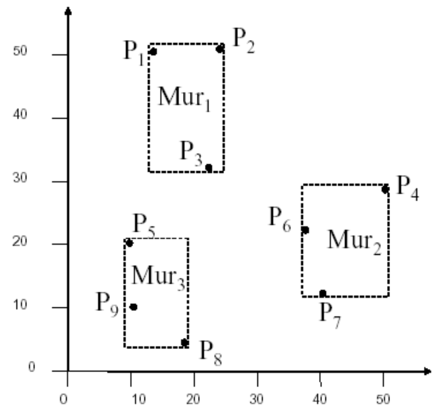
Objects a,b, and c are RkNN(q,1) results!

Similarity Search Algorithms: Reverse Nearest Neighbor Query

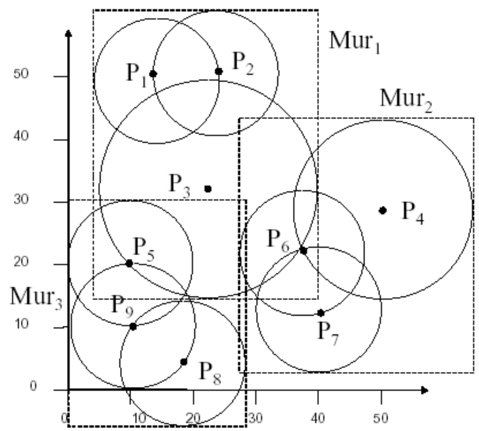
- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Basic sequential-scan-based solution:
    - Algorithm:
      - For each object o in DB, compute the kNNQ(o) result.
      - As result report those objects having q in the result of kNNQ(o).
    - Problem:
      - Very expensive, complexity $O(N^2)$, N=|DB|
    - Indexed based solutions
      - RNN-Tree [Korn, Muthukrishnan. ACM Int. Conf. Management of Data (SIGMOD), 2000]
      - RdNN-Tree [Yang, Lin. IEEE Int. Conf. Data Engineering (ICDE), 2001]
      - MRkNNCoP-Tree [Achtert, Böhm, Kröger, Kunath, Pryakhin, Renz. ACM Int. Conf. Management of Data (SIGMOD), 2006]
      - Geometric RkNN approach based on R-Tree [Tao, Papadias, Lian. Int. Conf. Very Large Databases (VLDB), 2004] [Emrich, Kriegel, Kröger, Renz, Züfle. ACM Int. Conf. Management of Data (SIGMOD), 2010].

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - RNN-Baum [Korn, Muthukrishnan. ACM Int. Conf. Management of Data (SIGMOD), 2000]
    - Preprocessing: Compute kNN-Distance for all objects in DB.
    - Use R-Tree to index all kNN-Spheres of all objects
    - Perform point query on the R-tree, i.e. RkNN result contains all objects for which their kNN-Sphere interects q.
    - Problems:
      - k has to be fixed for all queries
      - High overlap of page regions of the RNN-Tree → bad query performance



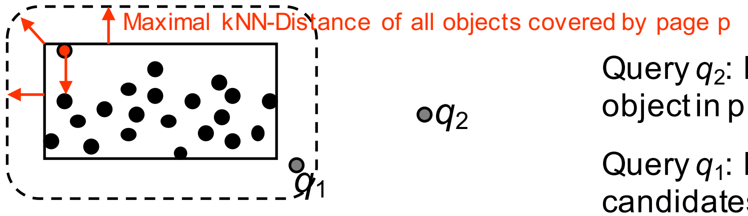
Normal R-Tree



RNN-Tree

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - RdNN-Baum [Yang, Lin. IEEE Int. Conf. Data Engineering (ICDE), 2001]
    - Use standard R-Tree to index all objects.
    - For each object o: Store the precomputed kNN-distance of o.
    - For each page region p: Store the maximum kNN-distance of all objects below p.
    - Query process:
      - Compute the minimal distance MINDIST(q,p) to page regions (starting at the root)
      - Prune subtree of page p if MINDIST(q,p) > kNN-distance assigned to p



Query q_2: Page p can be pruned, no object in p can have q_2 in its kNNs

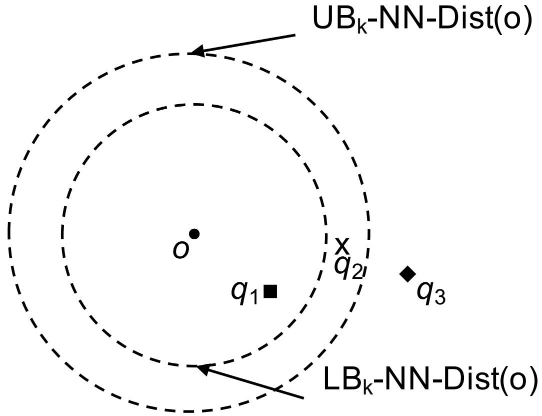
Query q_1: Page p may contain RkNN(q_1) candidates → load and explore page p

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Advantage of the RdNN-Tree approach over the RNN-Tree approach
    - Less page region overlap → better query performance
    - The approach can be easily transfered to the M-Tree → can be applied to general metric data
  - General problems with RNN and RdNN approaches:
    - Parameter k has to be fix → No flexibility w.r.t. general RkNN queries
    - High cost for updates, e.g. insert or delete operations → Recomputation of the index
  - To cope with the problem that the parameter k has to be fix, one can think about adapting the RdNN-Tree by precomputing and storing all possible kNN distances of all objects for an appropriate range of k, e.g. $1 \leqq k \leqq 100$
  - Problem: To high storage cost to manage k additional values for each object and each page region → bad index performance
  - Solution: Find a good (conservative) approximation of the behavior the k-NN distances for varying k → MRkNNCoP-Tree

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Using conservative approximations of the kNN distance:
    - Assumption: Instead of having given the exact kNN distance of an object o, we have given
      - the lower bounding kNN distance approaximation LB_k-NN-Dist(o)
      - the upper bounding kNN distance approaximation UB_k-NN-Dist(o)
    - If dist(o,q) $\leqq$ LB_k-NN-Dist(o) => o is a true hit, i.e. o $\in$ RkNN(q, k)
      In our example: $q = q_1$
    - If dist(o,q) $\geqq$ UB_k-NN-Dist(o) => o is a true drop, i.e. o $\notin$ RkNN(q, k)
      In our example: $q = q_3$
    - If LB_k-NN-Dist(o) < dist(o,q) < UB_k-NN-Dist(o) => o is a candidate and has to be refined in a final refinement step.
      In our example: $q = q_2$

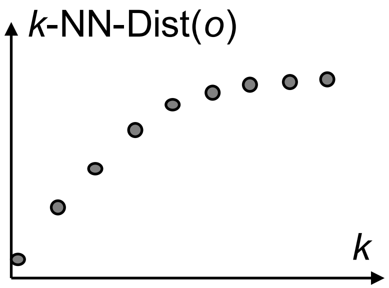


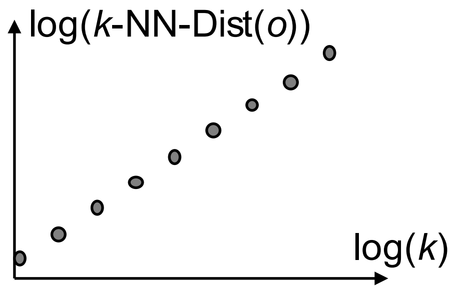
Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - How to find a good conservative approximations of the kNN distance?
    - Using the power law of the relationship between
      - the radius of an hypersphere ε
      - and encl(ε) = the number of objects covered by the hypersphere with radius ε.

$$encl(\varepsilon) \propto \varepsilon^{d_f}$$

      - where df = „fractal dimension“
    - Now, consider our k-NN-distance spheres:
      - ε = k-NN-Dist(o)
      - encl(ε) = k

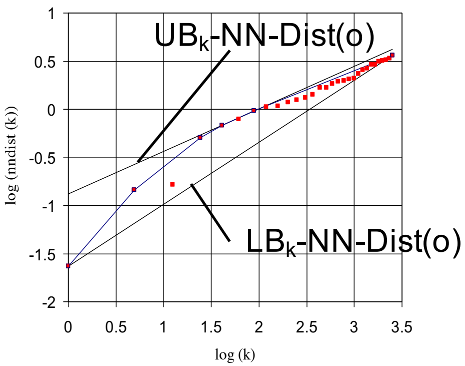


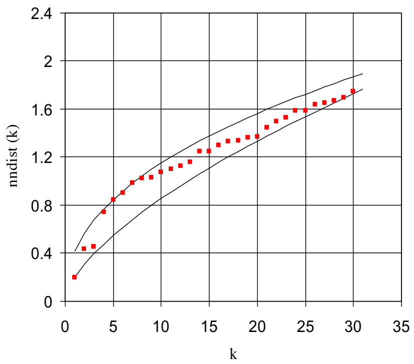


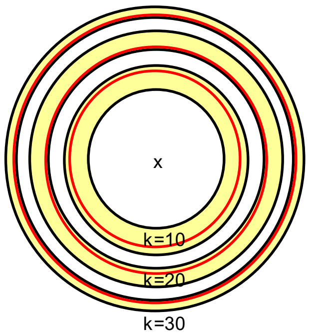
$\Rightarrow$ k approx. depends linear on the k-NN-Dist(o) in the log-log-space: $\log(k - \mathrm{NN} - \mathrm{Dist}(o)) \propto \frac{\log(k)}{d_f}$

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - How to find a good conservative approximations of the kNN distance?
    - In reality, log(k) and log(k-NN-Dist(o)) does not have a perfect linear relationship, but the dependency between k and k-NN-Dist(o) can be quite well approximated by two linear functions in the log(k)-log(k-NN-Dist(o))-space:







Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - MRkNNCoP-Tree [Achtert, Böhm, Kröger, Kunath, Pryakhin, Renz. ACM Int. Conf. Management of Data (SIGMOD), 2006]
    - Idea:
      - Manage all objects in a standard R-tree index
      - For each object o:
        - Compute the kNN distances (for $1 \leqq k \leqq k_{max}$) and build the optimal UB_k-NN-Dist(o) and LB_k-NN-Dist(o) functions (lines) in the log(k)-log(k-NN-Dist(o))-space.
        - Store these two functions together with the objects in the index.
      - For each page region p:
        - Build optimal conservative approaximations of all UB_k-NN-Dist(o) and LB_k-NN-Dist(o) functions (lines) of all objects covered by p => the result are again two functions in the log(k)-log(k-NN-Dist(o))-space.
        - Store these two linear functions together with the page region p.

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - MRkNNCoP-Tree [Achtert, Böhm, Kröger, Kunath, Pryakhin, Renz. ACM Int. Conf. Management of Data (SIGMOD), 2006]
    - Idea:
      - Query Algorithm:
        - Traverse the R-tree index (starting from the root)
        - Access the node entries:
          If entry is a page region p, then load the UB_k-NN-Dist(p) function
          If MINDIST(q,p) > UB_k-NN-Dist(p)(k), then prune p.
          else p could contain candidates and has to be accessed and explored.

          If entry is an object o, then load the LB_k-NN-Dist(o) and UB_k-NN-Dist(o) functions
          If dist(q,o) > UB_k-NN-Dist(o)(k), then prune o,
          If dist(q,o) < LB_k-NN-Dist(o)(k), then report o as result,
          else o is a candidate and has to be refined in a refinement step.

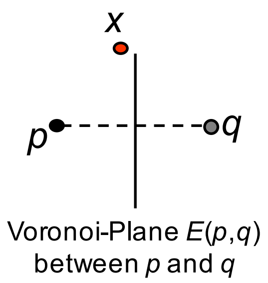
          Refinement:
          For all remaining candidates, compute the kNNs
          If q belongs to the kNNs of a candidate c, c is reported as result.

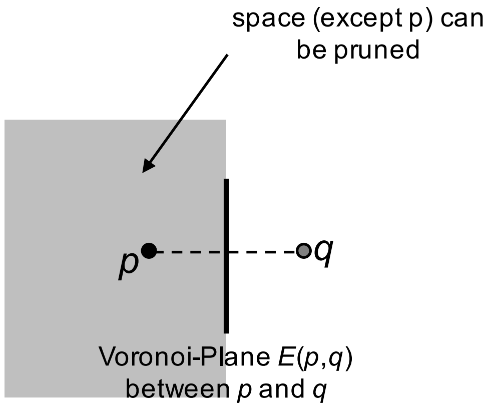
Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - MRkNNCoP-Tree [Achtert, Böhm, Kröger, Kunath, Pryakhin, Renz. ACM Int. Conf. Management of Data (SIGMOD), 2006]
    - Advantages:
      - More flexible w.r.t. parameter k
      - Similar idea can be used for M-Tree, i.e. metric data
    - Remaining Problems:
      - Still high cost for updates
    - General problem causing high update cost:
      - Up to now, all solutions are based on precomputed k-NN distances.
        Class of approaches using precomputed k-NN distances are called self pruning approaches.
    - To cope with this problem, we need an approach that is not based on pre-computed k-NN distances.
      → geometric RkNN-approach.

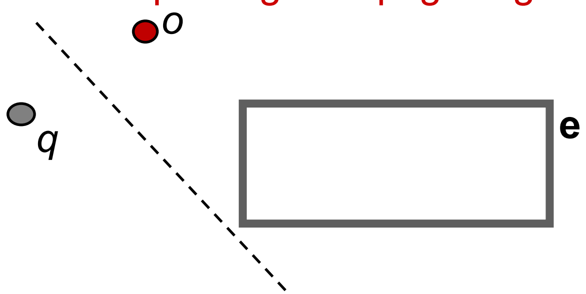
Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Geometric RkNN Query approach:
    - Basic Idea (k=1):
      - Given a query object q and another object p from DB, all objects on the side of the bisecting hyperplane between p and q that is opposite to q have p closer than q.





Geometric pruning of a page region (k=1):

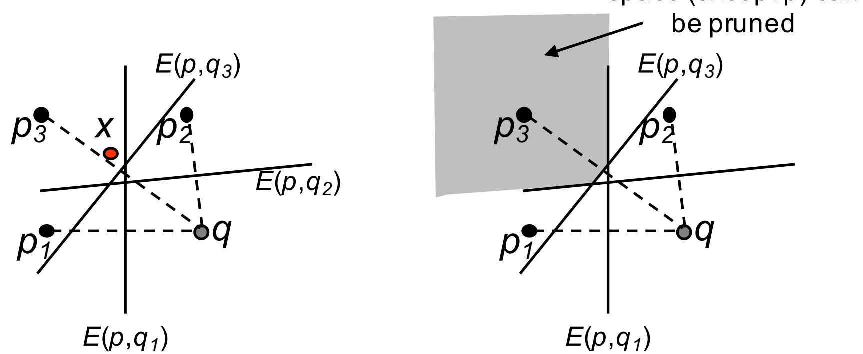


e „behind“ the hyperplane => no object in e can have q in it‘s NNs => prune e

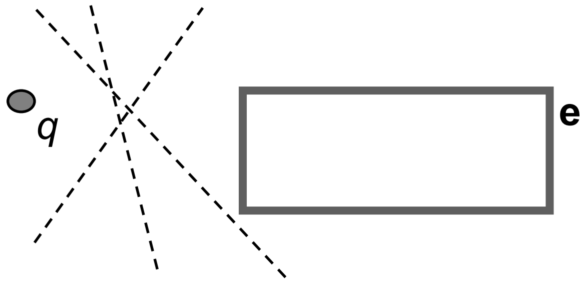
Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Geometric RkNN Query approach:
    - Basic Idea (k>1):
      - Given a query object q and another object p from DB, all objects on the side of the bisecting hyperplane between p and q that is opposite to q have p closer than q.

Example: k = 3



Geometric pruning of a page region (k>1):



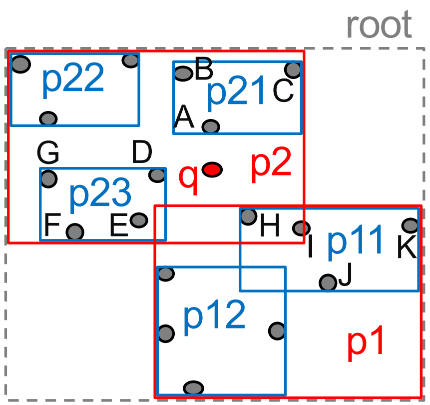
e „behind" k hyperplanes => no object in e can have q in it's kNNs => prune e

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Geometric RkNN Query approach:
    - Algorithm RkNN-Geom(q,DB):
      - Initialize an empty list of candidate objects CND.
      - Perform an R-tree based ranking query on DB analog to **NN-Index-HS(**pa,q**)** with APL**.**
      - Filter-Step: For each object o retrieved from the ranking do
        - If o cannot be pruned by k candidates in CND \\ use hyperplans between q and c in CND to prune objects then
          - insert o in CND;
          - prune all entries in APL based on the hyperplanes between q and candidates in CND;
      - Refinement-Step:
        - Compute k-NN query for each candidate c in CND and report c as result if q is in the k-NN result of c.

Similarity Search Algorithms: Reverse Nearest Neighbor Query

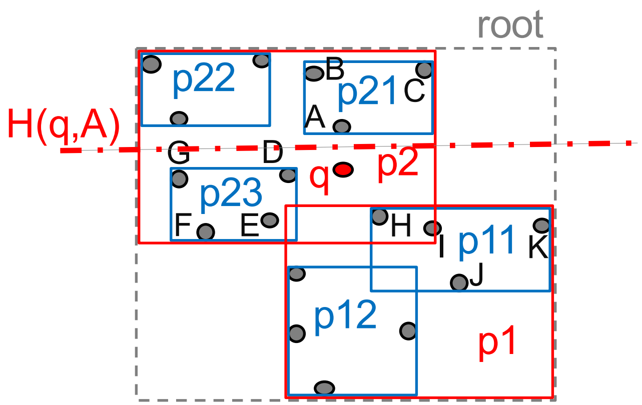
- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Geometric RkNN Query approach:
    - Example: RkNN-Geom(q,DB,k=1)



APL status:	CND:	APL entries pruned:
1: root 2: p2, p1 3: p21, p23, p1, p22 4: A, p23, p1, p22, B, C (object A prunes …)		

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Geometric RkNN Query approach:
    - Example: RkNN-Geom(q,DB,k=1)



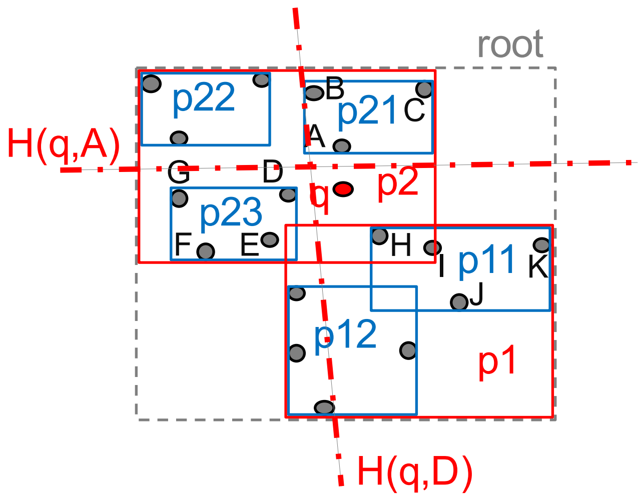
APL status:
1: root
2: p2, p1
3: p21, p23, p1, p22
4: A, p23, p1, p22, B, C
(object A prunes p22, B, C)
5: p23, p1
6: D, p1, E, F, G
(object D prunes ...)

CND:
A

APL entries pruned:
p22, B, C

Similarity Search Algorithms: Reverse Nearest Neighbor Query

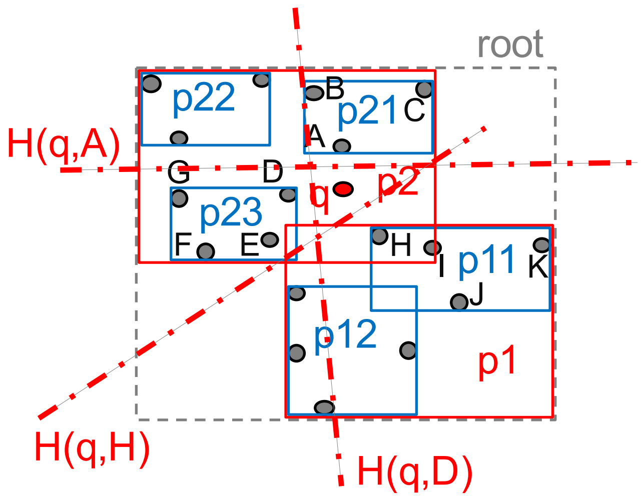
- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Geometric RkNN Query approach:
    - Example: RkNN-Geom(q,DB,k=1)



APL status:	CND:	APL entries pruned:
1: root	A	p22, B, C
2: p2, p1	D	E,F, G
3: p21, p23, p1, p22		
4: A, p23, p1, p22, B, C		
(object A prunes p22, B, C)		
5: p23, p1		
6: D, p1, E, F, G		
(object D prunes E, F, G)		
7: p1		
8: p11, p12		
9: H, I, p12, J, K		
(object H prunes ...)		

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Geometric RkNN Query approach:
    - Example: RkNN-Geom(q,DB,k=1)



APL status:
1: root
2: p2, p1
3: p21, p23, p1, p22
4: A, p23, p1, p22, B, C
(object A prunes p22, B, C)
5: p23, p1
6: D, p1, E, F, G
(object D prunes E, F, G)
7: p1
8: p11, p12
9: H, I, p12, J, K
(object H prunes I, J, K, p12)
10: empty → start refinement of candidates in CND

CND:	APL entries pruned:
A	p22, B, C
D	E,F, G
H	I, J, K, p12

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Geometric RkNN Query approach:
    - Advantages:
      - Totally flexible w.r.t. parameter k.
      - Supports DB updates efficiently.
    - Problems:
      - Can not be used for general metric data.
      - Cost for refinement could become high if many candidates remain.
        (Problem in particular for higher dimensional spaces.)

Similarity Search Algorithms: Reverse Nearest Neighbor Query

- Reverse (k-)Nearest Neighbor Queries (RkNNQ)
  - Overview:
    - Approaches based on self-pruning: RNN, RdNN, MRkNNCoP
      - pruning concept: precomputed kNN distances of objects used to prune themselves.
      - pros: very efficient pruning, applicable for general metric data (except RNN approach)
      - cons: fix k; high update cost
    - Approaches based on mutual pruning: RkNN-Geom (TPL)
      - pruning concept: hyperplane between an object and the query object is used to prune other objects.
      - pros: total flexibility w.r.t. k; no precomputation; efficient updates
      - cons: refinement can become expensive (for high dimensional spaces); not applicable for general metric data