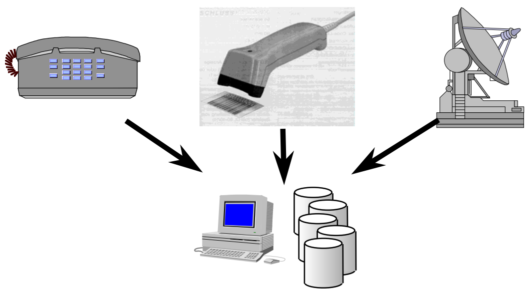
Christian Böhm
University for Health Informatics and Technology

Powerful Database Primitives to Support High Performance Data Mining

Tutorial, IEEE Int. Conf. on Data Mining, Dec/09/2002

1 Motivation

High Performance Data Mining



- Marketing
- Fraud Detection
- CRM
- Online Scoring
- OLAP

Fast decisions require knowledge just in time

Previous Approaches to Fast Data Mining

- Sampling
- Approximations (grid)
- Dimensionality reduct.

} Loss of quality

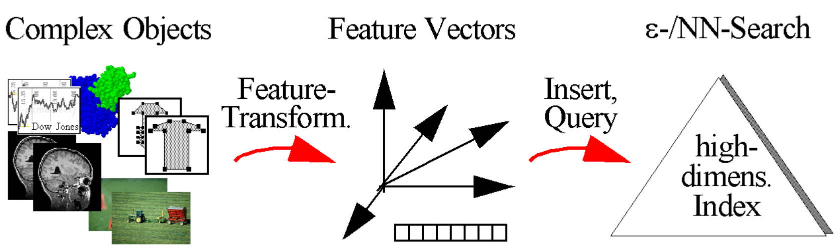
- Parallelism

Expensive & complex

All approaches combinable with DB primitives

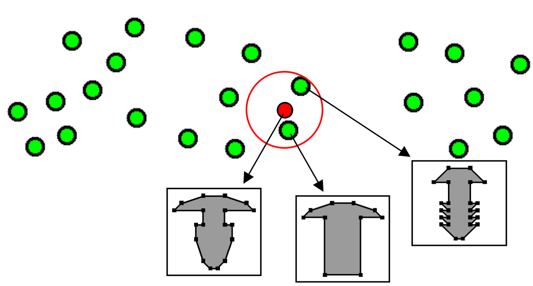
KDD appl. get parallelism for free

Feature Based Similarity

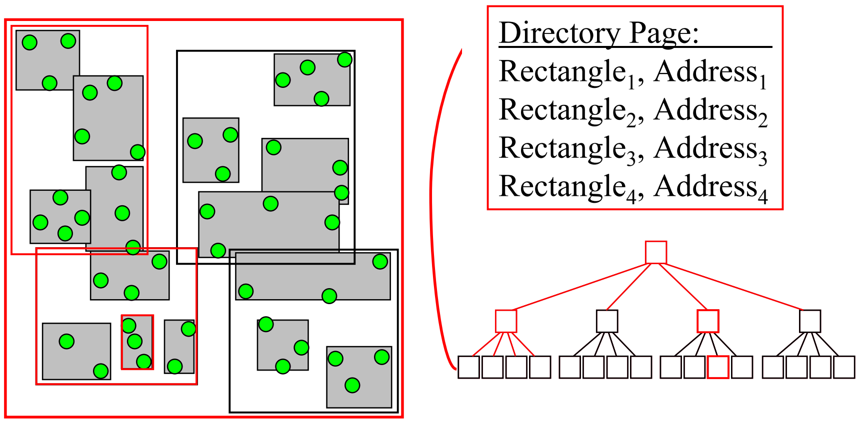


Simple Similarity Queries

- Specify query object and
  - Find similar objects – range query
  - Find the *k* most similar objects – nearest neighbor q.



Multidimensional Index Structure (R-Tree)

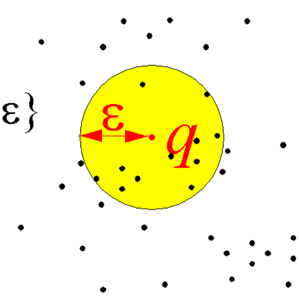


Similarity – Range Queries

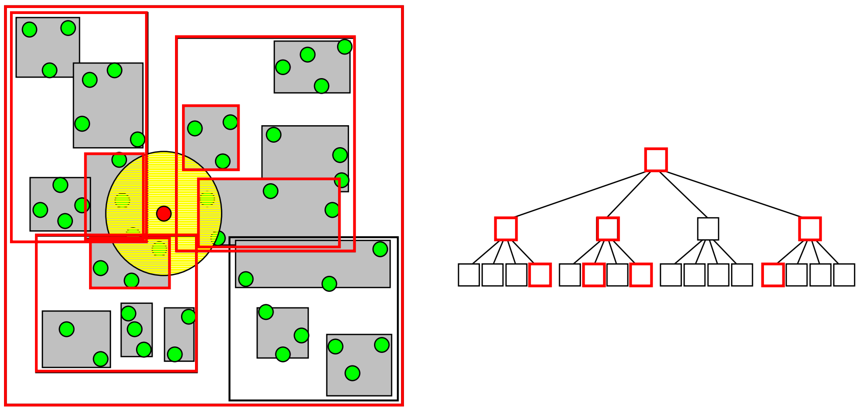
- Given: Query point q
  Maximum distance ε
- Formal definition:

  $$\mathrm{sim}_q(\varepsilon) := \{o \in DB \mid \mathrm{d}(q,o) \leq \varepsilon\}$$

- Cardinality of the result set is difficult to control:
  ε too small ➔ no results
  ε too large ➔ complete DB



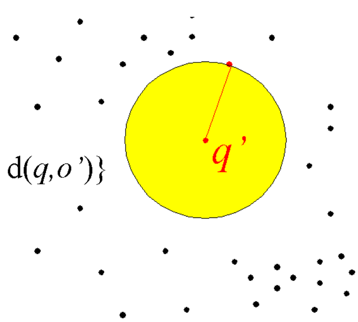
Index Based Processing of Range Queries



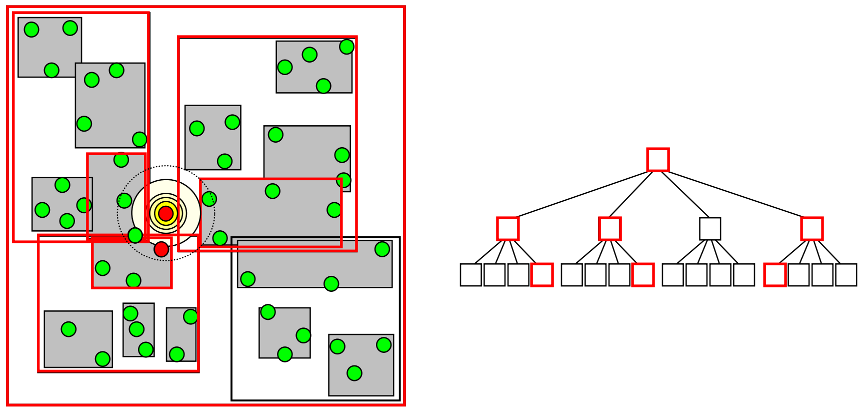
Similarity – Nearest Neighbor Queries

- Given:
  Query point q
- Formal definition:

  $NN_q := \{o \in DB \mid \forall o' \in DB \; d(q,o) \le d(q,o')\}$
- Ties must be handled:
  - Result set enlargement
  - Non-determinism (don't care)



Index Based Processing of NN Queries

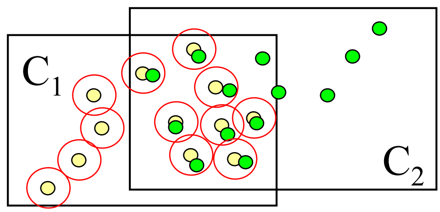


k-Nearest Neighbor Search and Ranking

- k-nearest neighbor query:
  - Do not only search only for one nearest neighbor but k
  - Stop distance is the distance of the k_{th} (last) candidate point
  - $k\mathrm{NN}_q$ is the smallest subset of DB that contains $\geq k$ elements with $\forall\ o \in k\mathrm{NN}_q,\ \forall\ o' \in DB \setminus k\mathrm{NN}_q\!: \|o - q\| < \|o' - q\|$
- Ranking-query:
  - Incremental version of k-nearest neighbor search
  - First call of FetchNext() returns first neighbor
  - Second call of FetchNext() returns second neighbor...
  - Typically only few results are fetched → Don't generate all!

Advanced Applications: Duplicates

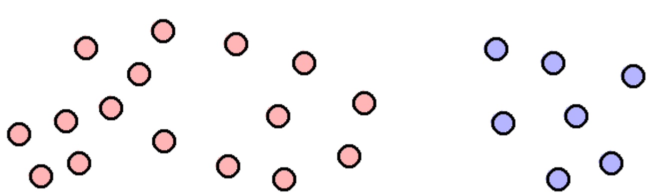
- Duplicate detection
  - E.g. Astronomical catalogue matching



- Similarity queries for large number of query obj

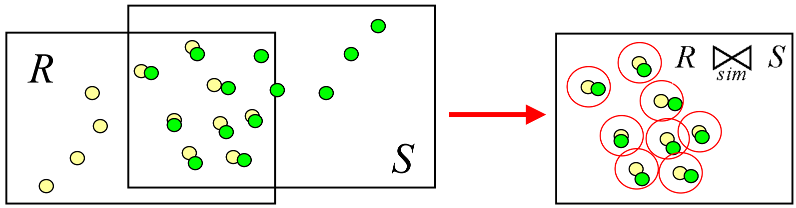
Advanced Applications: Data Mining

- Density based clustering (DBSCAN)



What is a Similarity Join?

- Given two sets *R*, *S* of points
- Find all pairs of points according to similarity



- Various exact definitions for the similarity join

Organization of the Tutorial

- Motivation
- Defining the Similarity Join
- Applications of the Similarity Join
- Similarity Join Algorithms
- Conclusion & Future Potential

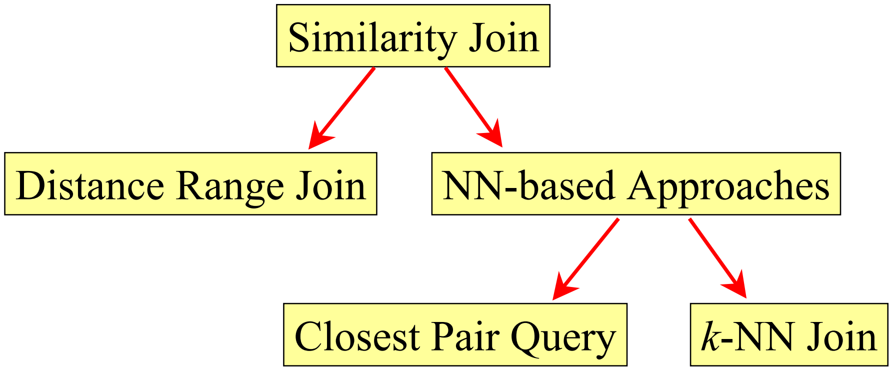
2 Defining the Similarity Join

What Is a Similarity Join?

Intuitive notion: 3 properties of the similarity join

- The similarity join is a **join** in the relational sense
  Two sets R and S are combined into one such that the new set contains pairs of points that fulfill a **join condition**
  $R \underset{sim}{\bowtie} S \subseteq R \times S$
- **Vector** or **metric objects**
  rather than ordinary tuples of any type
- The join condition involves **similarity**

What Is a Similarity Join?

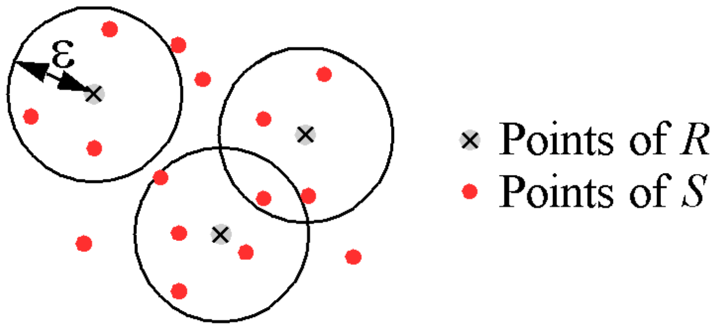


Distance Range Join (ε-Join)

- **Intuitition:** Given parameter ε
  All pairs of points where distance ≤ ε
- Formal Definition:
  $R \bowtie_{\varepsilon} S := \{(r_i, s_j) \in R \times S \colon \|r_i - s_j\| \leq \varepsilon\}$
- In SQL-like notation:
  SELECT * FROM *R*, *S* **WHERE** ||*R*.obj – *S*.obj|| ≤ ε

Distance Range Join (ε-Join)

- Most widespread and best evaluated join
- Often also called **the** similarity join



Distance Range Join (ε-Join)

- The distance range **self** join

$$R \bowtie_{\varepsilon} R$$

is of particular importance for data mining (clustering) and robust similarity search
- Change definition to exclude trivial results
- **Lemma 1.** the distance range *self* join is symmetric i.e.

$$(r_i, r_j) \in R \bowtie_{\varepsilon} R \Leftrightarrow (r_j, r_i) \in R \bowtie_{\varepsilon} R$$

Distance Range Join (ε-Join)

- Disadvantage for the user:
  Result cardinality difficult to control:
  - ε too small → no result pairs are produced
  - ε too large → all pairs from $R \times S$ are produced
- Worst case complexity is at least $o(|R|\cdot|S|)$
- For reasonable result set size, advanced join algorithms yield asymptotic behavior which is better than $O(|R|\cdot|S|)$

k-Closest Pair Query

- **Intuition**:
  Find those *k* pairs that yield least distance
- The principle of nearest neighbor search is applied on a basis **per pair**
- Classical problem of Computational Geometry
- In the database context introduced by
  [Hjaltason & Samet, *Incremental Distance Join Algorithms*, SIGMOD Conf. 1998]
- There called **distance join**

k-Closest Pair Query

- Formal Definition:

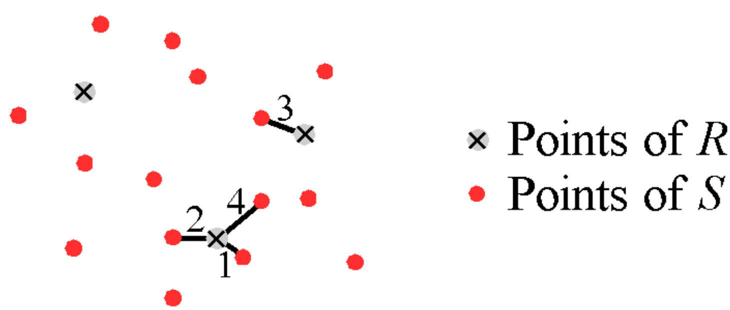
  $R \bowtie_{k\text{-CP}} S$ is the smallest subset of $R \times S$ that contains at least k pairs of points and for which the following condition holds:

  $$\forall\ (r,s) \in R \bowtie_{k\text{-CP}} S,\ \forall\ (r',s') \in R \times S \setminus R \bowtie_{k\text{-CP}} S: \|r-s\| < \|r'-s'\|$$

- Ties solved by **result set enlargement**
- Other possibility: **Non-determinism**
  (don't care which of the tie tuples are reported)

k-Closest Pair Query

In SQL notation: **SELECT * FROM** R, S
ORDER BY $\|R.\text{obj} - S.\text{obj}\|$
STOP AFTER k



k-Closest Pair Query

- Self-join:
  - Exclude $|R|$ trivial pairs (r_i, r_i) with distance 0
  - Result is symmetric
- Applications:
  - Find all pairs of stock quota in a database that are most similar to each other
  - Find music scores which are similar to each other
  - Noise robust duplicate elimination

k-Closest Pair Query

- Incremental **ranking** instead of exact specification of k
- No STOP AFTER clause:

  SELECT * **FROM** R, S
  ORDER BY $\|R.\text{obj} - S.\text{obj}\|$

- Open **cursor** and **fetch** results one-by-one
- Important: Only few results typically fetched
  → Don't determine the complete ranking

k-Nearest Neighbor Join

- **Intuition**:
  Combine each point with its k nearest neighbors
- The principle of nearest neighbor search is applied **for each point** of R

Christian Böhm

k-Nearest Neighbor Join

- Formal Definition:

  $R \bowtie_{k\text{-NN}} S$ is the smallest subset of $R \times S$ that contains for each point of R at least k points of S and for which the following condition holds:

  $\forall\ (r,s) \in R \bowtie_{k\text{-NN}} S,\ \forall\ (r,s') \in R \times S \setminus R \bowtie_{k\text{-NN}} S\text{: } \|r-s\| < \|r-s'\|$

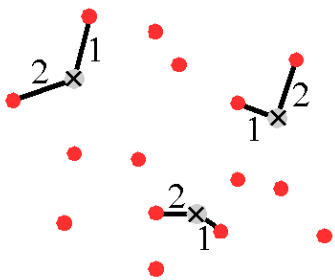
- Ties solved by **result set enlargement**
- Other possibility: **Non-determinism**
  (don't care which of the tie tuples are reported)

Christian Böhm

k-Nearest Neighbor Join

In SQL notation: (limited to k = 1)

SELECT * FROM R, S
GROUP BY R.obj
ORDER BY $\|R.\text{obj} - S.\text{obj}\|$



× Points of R
• Points of S

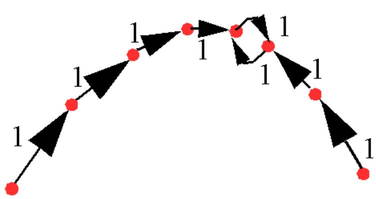
k-Nearest Neighbor Join

- The k-NN-join is inherently asymmetric:

  $R \bowtie_{k\text{-NN}} S$ and $S \bowtie_{k\text{-NN}} R$ have completely different meaning:

  $R \bowtie_{k\text{-NN}} S$ retrieves $k \cdot |R|$ pairs

  $S \bowtie_{k\text{-NN}} R$ retrieves $k \cdot |S|$ pairs



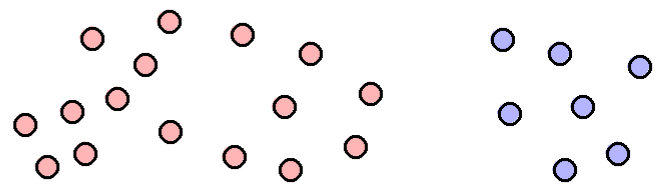
• Points of R

k-Nearest Neighbor Join

- Applications of the *k*-NN-join:
  - *k*-means and *k*-medoid clustering
  - Simultaneous nearest neighbor classification:
    A large set of new objects without class label are assigned according to the majority of *k* nearest neighbors of each of the new objects
    - Astronomical observation
    - Online customer scoring
- Ranking on the *k*-NN-join is difficult to define

3 Applications

Density Based Data Mining



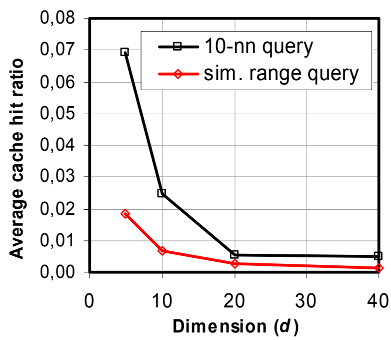
Schema for Data Mining Algorithms

Algorithmic Schema A_1

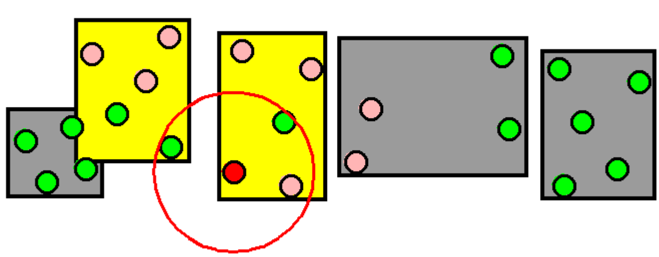
foreach Point $p \in D$
PointSet S := SimilarityQuery (p, ε);
foreach Point $q \in S$
DoSomething (p,q) ;

Iterative similarity queries and cache

- Due to curse of dimensionality:
  No sufficient inter-query locality of the pages

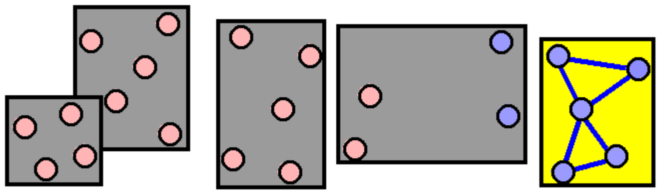


Iterative similarity queries and cache



Idea: Query Order Transformation

[Böhm, Braunmüller, Breunig, Kriegel: *High Perf. Clustering based on the Sim. Join*, CIKM 2000]



Schema Transformation

foreach DataPage P
    LoadAndPinPage (P) ;
    foreach DataPage Q
        if (mindist (P,Q) $\leq \varepsilon$)
            CachedAccess (Q) ;
            foreach Point $p \in P$
                foreach Point $q \in Q$
                    if (distance (p,q) $\leq \varepsilon$)
                        DoSomething' (p,q) ;
    UnFixPage (P) ;

Similarity Join

A_2 is a *Similarity-Join*-Algorithm:

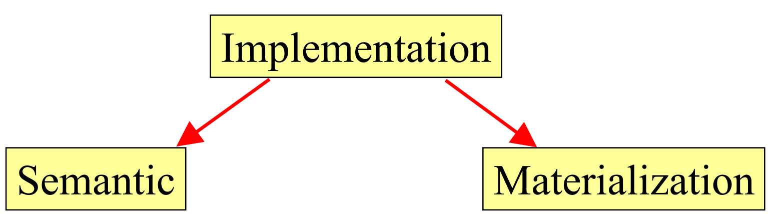
foreach PointPair $(p,q) \in R \bowtie_{\varepsilon} R$
DoSomething' (p,q) ;

Where $R \bowtie_{\varepsilon} R$ denotes the *Similarity-Join*:

SELECT * **FROM** $R\ r_1$, $R\ r_2$
WHERE distance (r_1.object, r_2.object) $\leq \varepsilon$

Implementation Variants

- Change of the order in which points are combined must partially be considered



Change algorithm to take unknown order into account

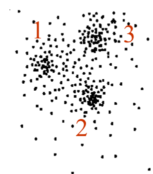
Materialize join result *j* and answer original queries by *j*

Example Clustering Algorithms

- **DBSCAN**
  [Ester, Kriegel, Sander, Xu: *A Density Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise´*, KDD 1996]
- Flat clustering (non hierarchical)

Semantic Rewriting

- **OPTICS**
  [Ankerst, Breunig, Kriegel, Sander: *OPTICS: Ordering Points To Identify the Clustering Structure*, SIGMOD Conf. 1999]
- Hierachical cluster-structure



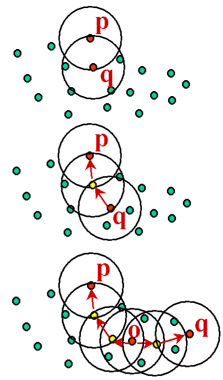
Materialization

Transformation by Semantic Rewriting

- Rewrite the algorithm to take the changed order of pairs into account
- Don´t assume any specific order in which pairs are generated
  → Arbitrary similarity join algorithm possible

Example: DBSCAN

- p ***core object*** in D wrt. ε, $MinPts$: $| N_\varepsilon(p) | \geq MinPts$
- p ***directly density-reachable*** from q in D wrt. ε, $MinPts$:
  1) $p \in N_\varepsilon(q)$ and
  2) q is a core object wrt. ε, $MinPts$
- ***density-reachable***: transitive closure.
- ***cluster***:
  - maximal wrt. density reachability
  - any two points are density-reachable from a third object



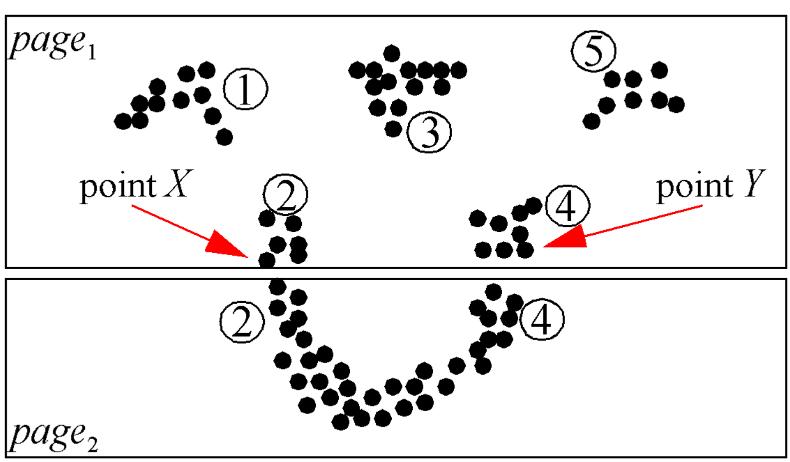
Implementation of DBSCAN on Join

- Core point property:
  DoSomething() increments a counter attribute
- Determination of maximal density-reachable clusters:
  DoSomething():
  - Assign ID of known cluster point to unknown cluster points
  - Unify two known clusters

Implementation of DBSCAN on Join

P_2 \ P_1		CORE POINT: ID	CORE POINT: NULL	NON-CORE POINT: ID	NON-CORE POINT: NULL
CORE POINT	ID	merge if P_1.ID<>P_2.ID (1)	P_1.ID = P_2.ID (2)	(3)	P_1.ID = P_2.ID (4)
	NULL	P_2.ID = P_1.ID (2)	P_1.ID = P_2.ID = new ID (5)	(6)	P_1.ID = P_2.ID = new ID (7)
NON-CORE POINT	ID	(3)	(6)	(8)	(8)
	NULL	P_2.ID = P_1.ID (4)	P_1.ID = P_2.ID = new ID (7)	(8)	(8)

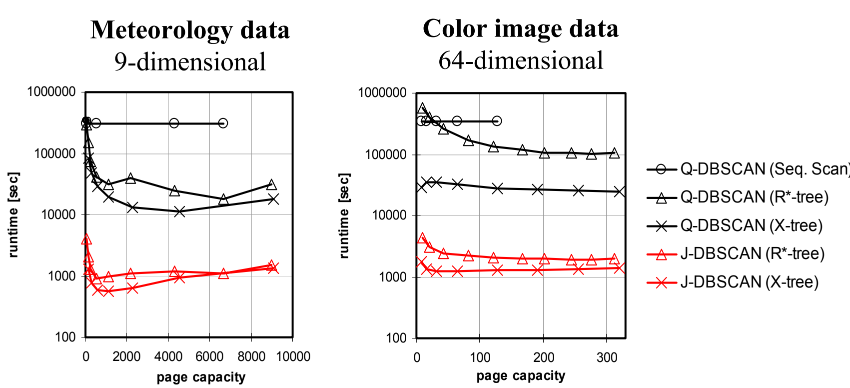
Implementation of DBSCAN on Join



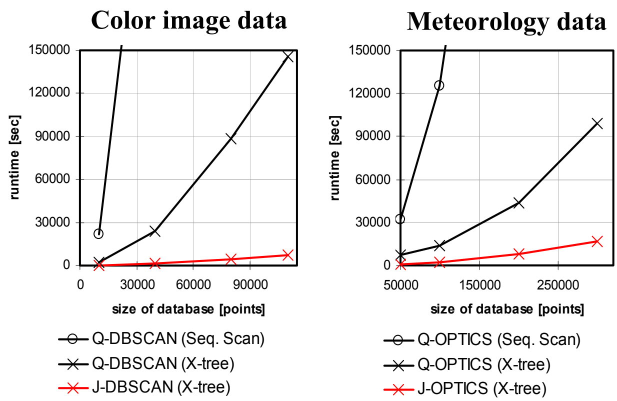
Implementing OPTICS (Materialization)

- The join result is predetermined before starting the actual OPTICS algorithm
- The result is materialized in some table with GROUP-BY on the first point of the pair
- The OPTICS algorithm runs unchanged
- Similarity queries are answered from the join materialization table (much faster)
- Disadvantage: High memory requirements

Experimental Results: Page Capacity



Experimental Results: Scalability



Robust Similarity Search

[Agrawal, Lin, Sawhney, Shim: Fast Similariy Search in the Presence of Noise,...., VLDB 1995]

- Usual similarity search with feature vectors: Not robust with respect to
  - Noise:
    Euclidean distance sensitive to mismatch in single dimension
  - Partial similarity:
    Not complete objects are similar, but parts thereof
- Concept to achieve robustness:
  Decompose each data object and query object into sub-objects and search for a maximum number of similar subobjects

Robust Similarity Search

- Prominent concept borrowed from IR research: String decomposition: Search for similar words by indexing of character triplets (*n*-lets)
- Query transformed to **set** of similarity queries → similarity join between query set and data set
- Robustness achieved in result recombination:
  - Noise robustness: Ignore missing matches
  - Partial search: Dont enforce complete recombination

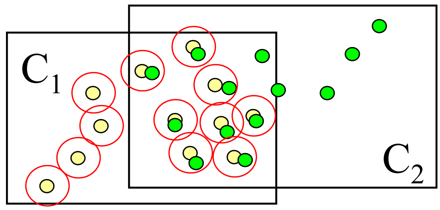
Robust Similarity Search

Applications:

- Robust search for sequences:
  [Agrawal, Lin, Sawhney, Shim: Fast Similariy Search in the Presence of Noise,...., VLDB 1995]
- Principle can be generalized for objects like
  - Raster images
  - CAD objects
  - 3D molecules
  - etc.

Astronomical Catalogue Matching

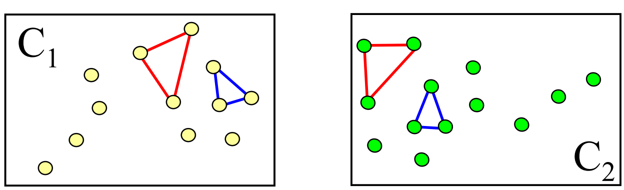
- Relative position of catalogues approx. known:
  - Position and intensity parameters in different bands



- $C_1 \bowtie_{\varepsilon} C_2$
- Determine ε according to device tolerance

Astronomical Catalogue Matching

- Relative position unknown:
  - Match according to triangles and intensity



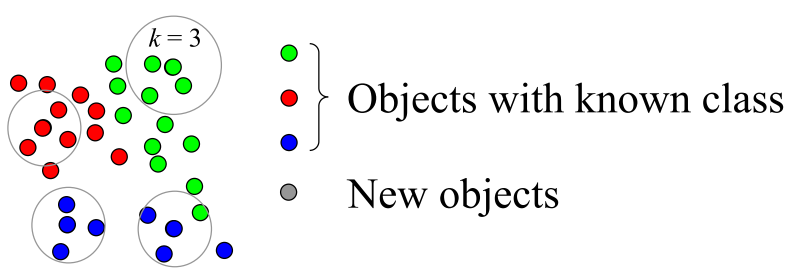
- Search triangles and store parameters (height,...)
- triangles (C_1) $\bowtie_{\varepsilon}$ triangles (C_2)

k-Nearest Neighbor Classification

- Simultaneous classification of many objects
  [Braunmüller, Ester, Kriegel, Sander: *Efficiently Supporting Multiple Similarity Queries for Mining in Metric Databases*, ICDE 2000]
  - Astronomy
    - Some 10,000 new objects collected per night
    - Classify according to some millions of known objects
  - Online customer scoring
    - Some 1,000 customers online
    - Rate them according to some millions of known patterns

k-Nearest Neighbor Classification

- Example:



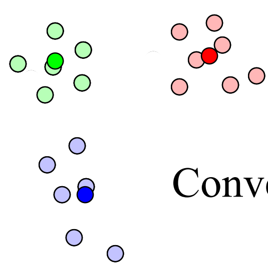
- New objects $\bowtie_{k\text{-NN}}$ Known objects

k-Means and k-Medoid Clustering

- k Points initially randomly selected („centers“)
- Each database point assigned to nearest center
- Centers are re-determined
  - k-means: Means of all assigned points (artificial p.)
  - k-medoid: One central database point of the cluster
- Assignment and center determination are repeated until convergence

k-Means and k-Medoid Clustering

- Example: (k-means with k = 3)



Convergence!

- Each assignment phase: DB-Points $\bowtie_{1\text{-NN}}$ Centers

4

Similarity Join Algorithms

Algorithms´ Overview

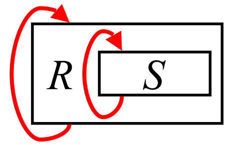
- Similarity join
  - Range dist. join
    - Index based
    - on-the-fly index
    - Hashing based
    - Sorting based
  - Closest pair qu.
  - *k-NN* join

Optimization
- Cost modeling
- CPU optimizing

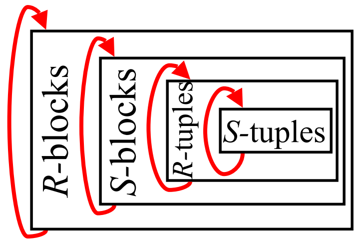
Christian Böhm

Nested Loop Join

- Simple nested loop join:
  - Iterate over *R*-points
  - Nested iteration over *S*-points
    → *S* is scanned |*R*| times, high I/O cost

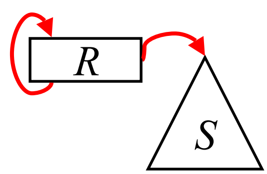


- Nested block loop join:
  - First iterate over blocks
  - Nested iterate over tuples
    → *S* scanned |*R*|/|*B*| times



Indexed Nested Loop Join

- Iterate over every point of *R*
- Determine matches in *S* by similarity queries on the index

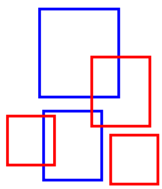


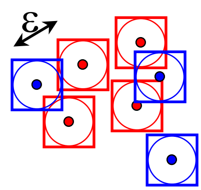
- Due to the curse of dimensionality:
  → Performance deterioration of the similarity q.
  → Then not competitive with nested loop join
  (Depends on dimensionality and selectivity determined by ε)

Spatial Join ↔ Similarity Join

- 2D polygon databases
- Join-predicate: Overlap
- Conserv. approximation: MBR (ax-par. rectangle)

- High-D point databases
- Join-predicate: Distance
- Map ε-join to spatial join Cube with edge-length ε





- Some strategies can be borrowed from the spatial join

R-tree Spatial Join (RSJ)

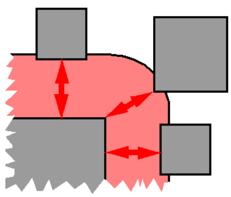
[Brinkhoff, Kriegel, Seeger: *Efficient Process. of Spatial Joins Using R-trees*, SIGMOD Conf. 1993]

- Originally: Spatial join for 2D rect. intersection
- Depth-first search in R-trees and similar indexes
- Assumption: Index preconstructed on *R* and *S*
- Simple recursion scheme (equal tree height):

  procedure r_tree_join (*R*, *S*: page)
      foreach $r \in R$.children **do**
          foreach $s \in S$.children **do**
              if intersect (*r,s*) **then** r_tree_join (*r,s*) ;

R-tree Spatial Join (RSJ)

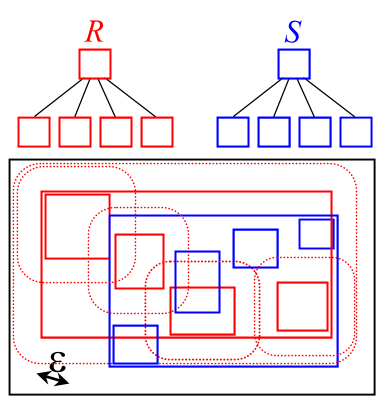
- Adaptation for the similarity join: Distance predicate rather than intersection
- For pair (R,S) of pages: mindist (R,S)
  → Least possible distance of two points in (R,S)



$$\text{mindist} = \sum_{0 \le i < d} \begin{cases} R.\text{lb}_i - S.\text{ub}_i & \text{if } R.\text{lb}_i > S.\text{ub}_i \\ 0 & \text{otherwise} \\ S.\text{lb}_i - R.\text{ub}_i & \text{if } S.\text{lb}_i > R.\text{ub}_i \end{cases}$$

R-tree Spatial Join (RSJ)

procedure r_tree_sim_join (R, S, ε)
  if IsDirpg (R) ∧ IsDirpg (S) **then**
    foreach $r \in R$.children **do**
      foreach $s \in S$.children **do**
        if mindist $(r,s) \le \varepsilon$ **then**
          CacheLoad(r); CacheLoad(s);
          r_tree_sim_join (r,s,ε) ;
  else (* assume R,S both DataPg *)
    foreach $p \in R$.points **do**
      foreach $q \in S$.points **do**
        if $|p - q| \le \varepsilon$ **then** report (p,q);



R-tree Spatial Join (RSJ)

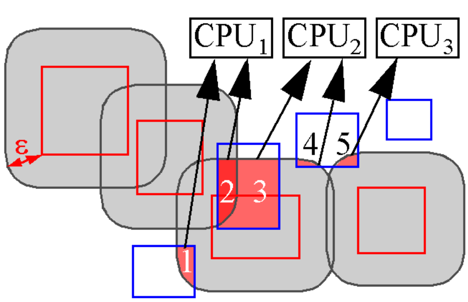
- Extension to different tree heights straightforw.
- Several additional optimizations possible
- CPU-bound
  - Cost dominated by point-distance calculations
- Disadvantages
  - No clear strategies for page access priorization
  - Single page accesses
    → Can be outperformed by nested block loop join

Christian Böhm

Parallel RSJ

[Brinkhoff, Kriegel, Seeger: *Parallel Processing of Spatial Joins Using R-trees*, ICDE 1996]

- A **task** corresponds to a pair of subtrees
  - At high tree level (e.g. root or second level)



Various Strategies:
- Static Range Assignment
- Static Round Robin
- Dynamic Task Assignment

Christian Böhm

Breadth-First R-tree Join (BFRJ)

[Huang, Jing, Rundensteiner: *Spatial Joins Using R-trees: Breadth-First Traversal...*, VLDB 1997]

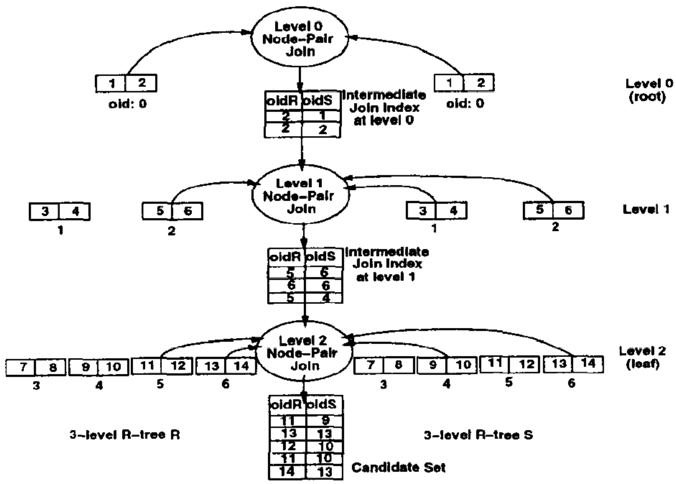
- Again spatial join for 2D rectangle intersection
- Shortcoming of RSJ:
  - No strategy in outer loop improving locality in inner
  - Depth-first traversal not flexible, because a pair of tree branches must be ended before next pair started

→ unnecessary page accesses

Breadth-First R-tree Join (BFRJ)

- Solution:
  - Proceed level by level (breadth-first traversal)
  - Determine all relevant pairs for the next level
    → **intermediate join index** (IJI)
  - Sort the IJI according to suitable order before accessing the next level
    → **global optimization** strategy

Breadth-First R-tree Join (BFRJ)



Approaches without Preconstructed Index

- Indexes can be constructed temporarily for join
- R-tree construction by INSERT too expensive
  → Use cheap bottom-up-construction
  - Hilbert R-trees: O (n log n)
    [Kamel, Faloutsos: *Hilbert R-trees: An Improved R-tree using Fractals*, VLDB 1994]
    Sort points by *SFC* and pack adjacent points to page
  - Buffer trees
    [van den Bercken, Seeger, Widmayer: *A Generic Approach to Bulk Loading..*, VLDB 1997]
  - Repeated partitioning
    [Berchtold, Böhm, Kriegel: *Improving the Query Performance ...*, EDBT 1998]
- Index construction can amortize during join

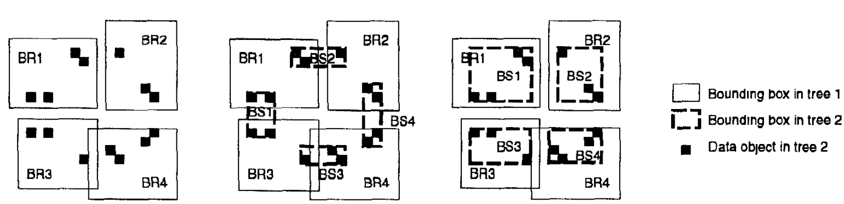
Seeded Trees

[Lo, Ravishankar: *Spatial Joins Using Seeded Trees*, SIGMOD Conf. 1994]

- Again spatial join for 2D rectangle intersection
- Assumption:
  Only **one** data set (*R*) is supported by index
- Typical application:
  Set *S* is subquery result
- Idea:
  Use partitioning of *R* as a template for *S*

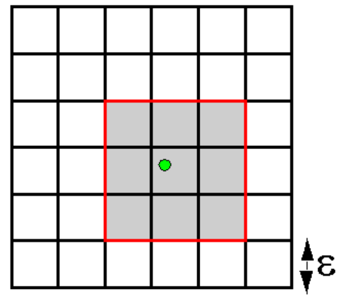
Seeded Trees

- Motivation
  - Early **inserts** to R-trees decide initial organization
  - We know that *S* will be matched with *R*
  - Start with small template tree instead of empty root
    → **seed levels**



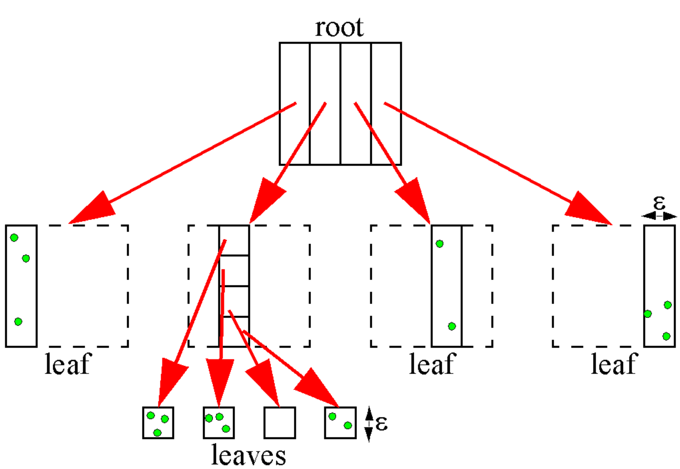
The ε-kdB-tree

[Shim, Srikant, Agrawal:
High-dimensional Similarity Joins, ICDE 1997]



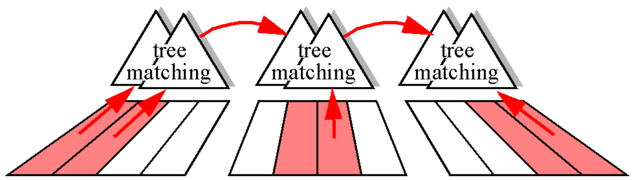
- Algorithm for the
  range distance self join
- General idea:
  Grid approximation where
  grid line distance = ε
- Not all dimensions used for decomposition:
  As many dimensions as needed to achieve a defined node capacity

The ε-kdB-tree



The ε-kdB-tree

- Node fanout: $\lceil 1/\varepsilon \rceil$ (assuming data space $[0..1]^d$)
- Tree structure is specific to given parameter ε
  → must be constructed for each join
- The ε-kdB-trees of two adjacent stripes are assumed to fit into main memory



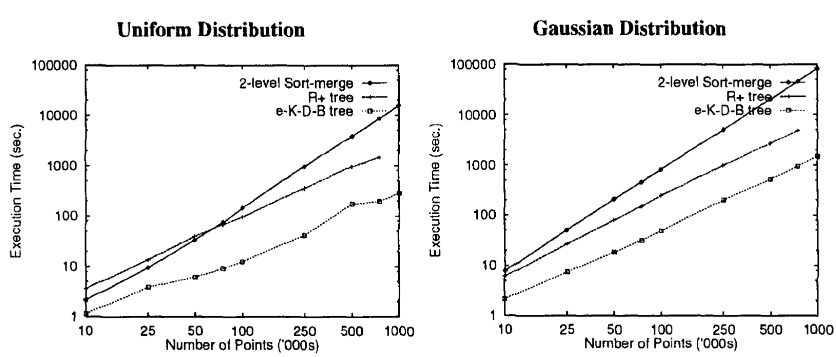
The ε-kdB-tree

procedure t_match (R, S: node)
  if is_leaf (R) $\vee$ is_leaf (S) **then**
    ...
  else
    for i:=1 to $\lceil 1/\varepsilon \rceil - 1$ **do**
      t_match(R.child[i], S.child [i]) ;
      t_match (R.child[i], S.child [i+1]) ;
      t_match (R.child[i+1], S.child[i]) ;
    t_match (R.child[$\lceil 1/\varepsilon \rceil$], S.child[$\lceil 1/\varepsilon \rceil$]) ;

The ε-kdB-tree

- Limitation:
  For large ε values not really scalable
- In high-dimensional cases, ε=0.3 can be typical
  → 60% of data must be held in main memory
- As long as data fit into main memory:
  ε-kdB-tree is one of the best similarity join algorithms

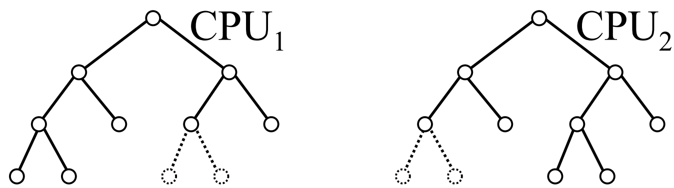
The ε-kdB-tree



The Parallel ε-kdB-tree

[Shafer, Agrawal: *Parallel Algorithms for High-dimensional Similarity Joins,* VLDB 1997]

- Parallel construction of the ε-kdB-tree:
  - Each processor has random subset of the data (1/*N*)
  - Each processor constructs ε-kdB-tree of its own set
  - Identical structure is enforced e.g. by split broadcast

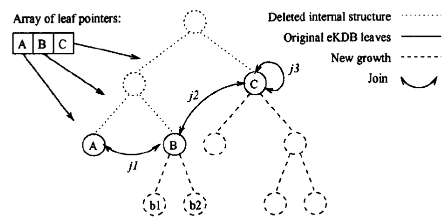


The Parallel ε-kdB-tree

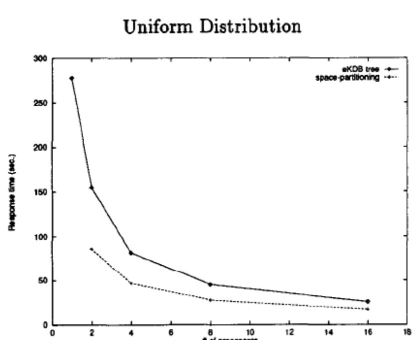
- Workload distribution:
  - Global determination of the cumulated node sizes
  - A unit **workload** is a pair (*r,s*) of leaf nodes
  - The cost of a workload is
    $|r|\cdot|s|$ for different leaves
    and $|r|\cdot(|r|+1)/2$ for a single leaf (self join)
  - Data is redistributed: Each processor gets 1/*N* work
    - join units are clustered to preserve locality
    - minimize redistribution (communication) and replication

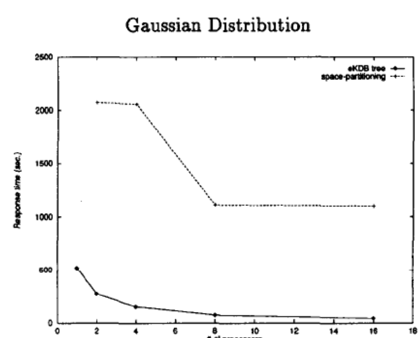
The Parallel ε-kdB-tree

- Workload execution:
  - delete internal structure
  - cum. node size too large → second growth phase
  - data redistribution performed asynchronously: Data sent in depth-first order of tree traversal to avoid network flooding



The Parallel ε-kdB-tree



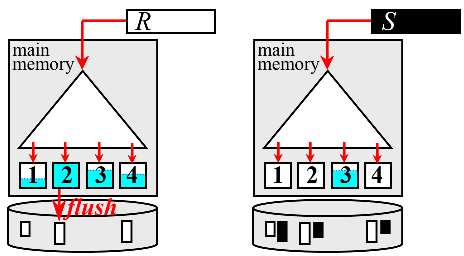


Plug & Join

[van den Bercken, Schneider, Seeger: *Plug&Join: An Easy-to-Use Generic Algorithm,* EDBT 2000]

Generic technique for several kinds of join

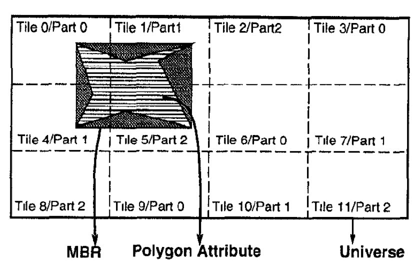
- Main-memory R-tree constructed from R-sample
- Partition *R* and *S* acc. to R-tree (buffers at leaves)



Christian Böhm

Partition Based Spatial Merge Join

- Spatial join method using replication
  [Patel, DeWitt: *Partition Based Spatial-Merge Join*, SIGMOD Conf. 1997]
  - Both sets *R* and *S* are partitioned with replication
  - Space is regularly tiled
  - Partitions either correspond to tiles or are determined from them using hashing



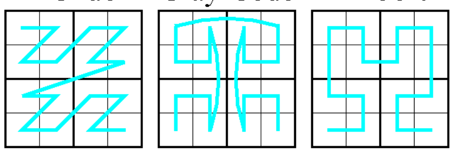
- Similar: Spatial Hash Join
  [Lo, Ravishankar: *Spatial Hash Joins*, SIGMOD Conf. 1996]

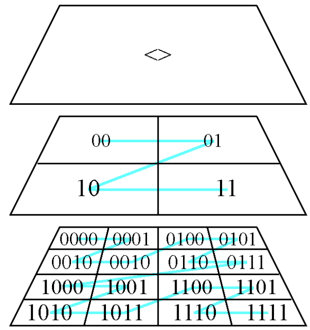
Christian Böhm

Approaches Using Space Filling Curves

- Space filling curves recursively decompose the data space in uniform pieces
- Various different orders:

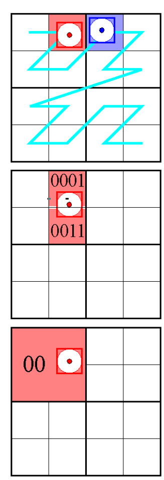
Z-Order Gray-Code Hilbert





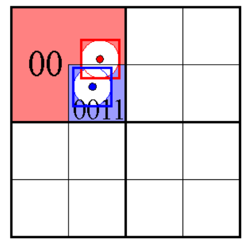
Approaches Using Space Filling Curves

- Efficient filter for the join:
  Objects in different cells cannot intersect each other
  → Sort-merge-join e.g. on Z-order
- Problem:
  Object may cross grid lines
  - either decompose object (redundant)
  - or assign to containing cell



Approaches Using Space Filling Curves

- If all cells have uniform size:
  → Equi-join on grid cell numbers (bit strings)
- If cells have varying size:
  → Bit strings of varying length
- Objects may intersect ...
  - **if** bitstr (r) is prefix of bitstr (s)
  - **or** bitstr (s) is prefix of bitstr (r)



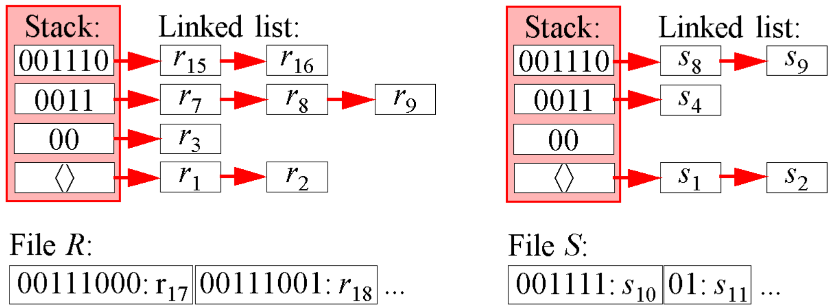
Orenstein‘s Spatial Join

[Orenstein: *An Algorithm for Computing the Overlay of k-Dim. Spaces*, SSD 1991]

- Allows (limited) redundancy, object decompos.
- Algorithm:
  - Objects are decomposed
  - Partial objects are ordered according to the lexicographical order of the bit strings
  - Objects are accessed in sort-merge like fashion
  - Two stacks are maintained to keep track of the prefix objects of R and S.

Orenstein‘s Spatial Join

- Stacks for prefix objects:



Multidimensional Spatial Join

[Koudas, Sevcik: *High-Dimensional Similarity Joins,* ICDE 1997, Best Paper Award]

- No redundancy allowed at all
- Instead of stacks:
  Separate **level files** for different bitstring length
- Problems with no redundancy:
  - With increasing dimension: increasing ε
  - Increasing chance that object intersects one of the primary decomposition lines → approx. by < >

Multidimensional Spatial Join

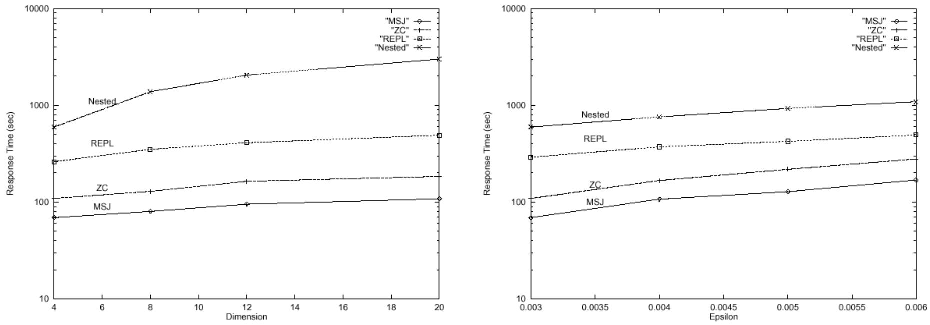
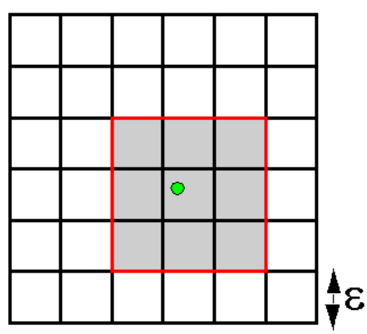

(a) Increasing dimension for epsilon=0.003 (b) Increasing epsilon for d=4

Figure 8: Performance of joins between stock market data

Epsilon Grid Order

[Böhm, Braunmüller, Krebs, Kriegel:
Epsilon Grid Order, SIGMOD Conf. 2001]

- Motivation like ε-kdB-tree: Based on **grid** with grid line distance ε
- Possible join mates restricted to 3^d cells



- Here no tree structure but sort order of points based on lexicographical order of the grid cells

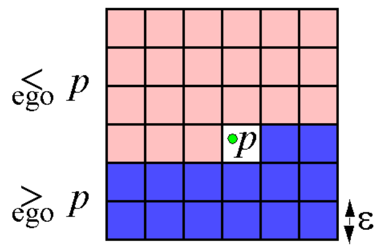
Epsilon Grid Order

Definition 1 Epsilon Grid Order $(\cdot \underset{\text{ego}}{\leq} \cdot)$

For two vectors p, q the predicate $p \underset{\text{ego}}{\leq} q$ is *true* if (and only if) there exists a dimension d_i such that the following conditions hold:

(1) $\left\lfloor \frac{p_i}{\varepsilon} \right\rfloor < \left\lfloor \frac{q_i}{\varepsilon} \right\rfloor$

(2) $\left\lfloor \frac{p_j}{\varepsilon} \right\rfloor = \left\lfloor \frac{q_j}{\varepsilon} \right\rfloor \qquad \forall j < i$



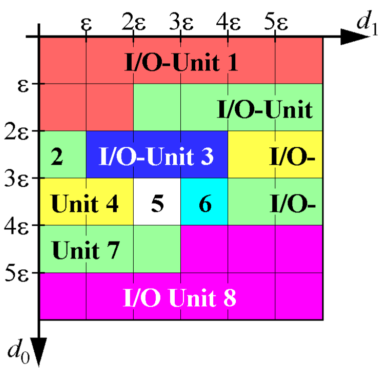
Epsilon Grid Order

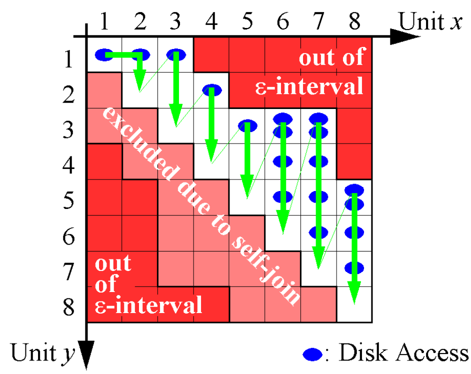
- A simple exclusion test (used for I/O):
  A point q with
  $q \underset{\text{ego}}{\leq} p - [\varepsilon,\varepsilon,\ldots,\varepsilon]^{\mathrm{T}}$ or $p + [\varepsilon,\varepsilon,\ldots,\varepsilon]^{\mathrm{T}} \underset{\text{ego}}{\leq} q$
  cannot be join mate of point p or any point beyond p (with respect to epsilon grid order)

- The interval between $p-[\varepsilon,\ldots,\varepsilon]^{\mathrm{T}}$ and $p+[\varepsilon,\ldots,\varepsilon]^{\mathrm{T}}$ is called ε-interval

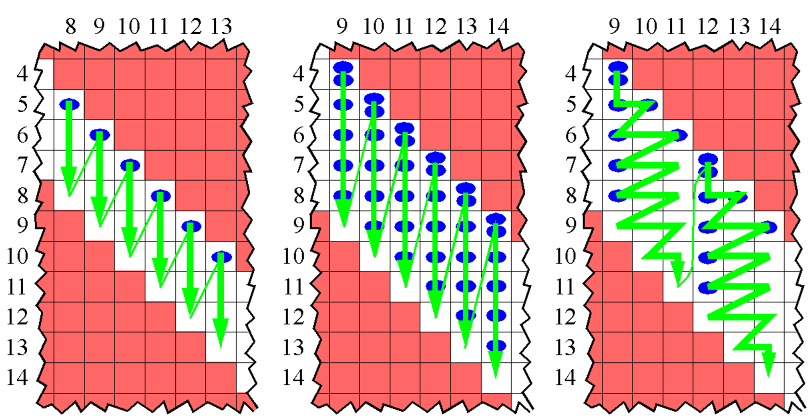
Epsilon Grid Order

- Sort file and decompose it into I/O units



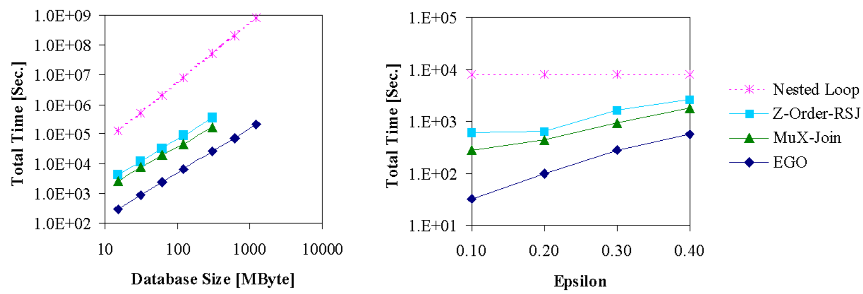


Epsilon Grid Order



(a) gallop mode (b) I/O thrashing (c) crabstep

Epsilon Grid Order



Christian Böhm

101
120

Closest Pair Queries

[Hjaltason, Samet: *Incremental Distance Join Algorithms for Spatial DB*, SIGMOD Conf. 1998]

- For both point objects and spatial objects
- Find *k* objects with least distance

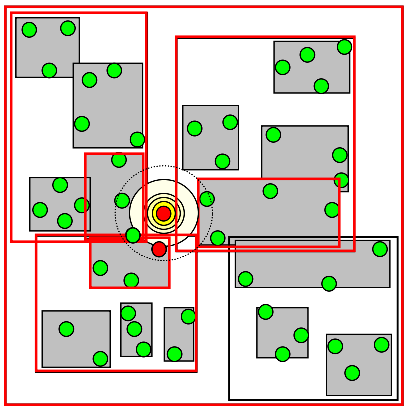
- Basis algorithm* for nearest neighbor search extended to take **point pairs** into account

* [Hjaltason, Samet: *Ranking in Spatial Databases*, SSD 1995]

Christian Böhm

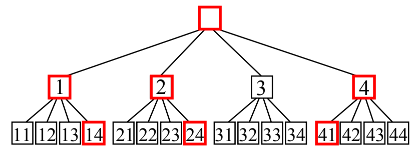
102
120

Basis Algorithm for NN Search



Active Page List:

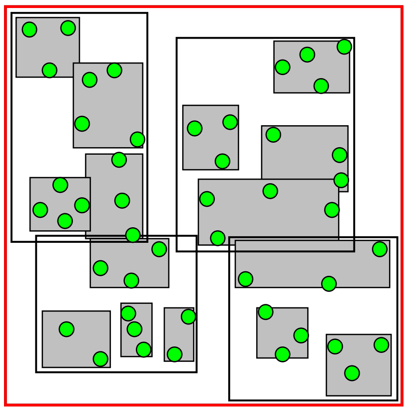
p_{14} | p_4 | p_{24} | p_3 | p_{12} | p_{23} | p_{13} | p_{21} | p_{22}



Hjaltason/Samet: Closest Pair Queries

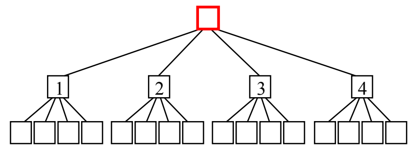
- **Nearest Neighbor → Closest Pair Query**
- *k* result points → *k* point pairs
- active page list → list of active page pairs
- initialization root → pair (root$_R$, root$_S$)
- distance point/query → distance of point pair
- mindist page/query → mindist betw. page pair

Hjaltason/Samet: Closest Pair Queries



Active Page List:

(root,p_1)|(root,p_2)|(root,p_3)|(root,p_4)



Hjaltason/Samet: Closest Pair Queries

- Unidirectional node expansion:
  Given a pair (r_i,s_j) only one node is expanded
- Closest pair ranking:
  Incremental version of k-closest pair queries
  → stop criterion is validation of next pair
- k-nearest neighbor join:
  Runs a closest pair ranking and filters out the $(k+1)_{st}$ occurrence (and more) of each point of R

Hjaltason/Samet: Closest Pair Queries

- Two strategies for tie breaks (same distance):
  - Depth-first
  - Breadth first
- Three policies for tree traversal
  - Basic (one tree determines priority)
  - Even (priority to node with shallower depth)
  - Simultaneous (all possible pairs are candidates for traversal)

Alternative Approaches

[Shin, Moon, Lee: *Adaptive Multi-Stage Distance Join Processing*, SIGMOD Conf. 2000]

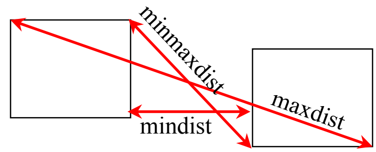
- Various improvements and optimizations
  - Bidirectional node expansion

(root,root) $\rightarrow$ (p_1,p_3) | (p_2, p_3) | (p_2, p_4) | (p_1, p_2) | (p_3, p_4) | (p_1, p_4)

  - Plane sweep technique for bidirectional node exp.
  - Adaptive multi-stage algorithm
    - Aggressive pruning using estimated distances

Alternative Approaches

[Corral, Manolopoulos, Theodoridis, Vassilakopoulos: *Closest Pair Queries in Spatial Databases*, SIGMOD Conf. 2000]



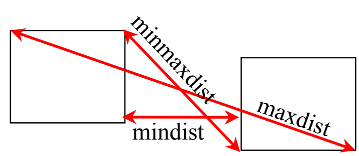
- 5 different algorithms for closest point queries
  - **Naive**: Depth-first traversal of the two R-trees
    → recursive call for each child pair (r_i, s_j) of (r, s)
  - **Exhaustive**: like **naive** but prune page pairs the mindist of which exceeds the current *k*-CP-dist
  - **Simple recursive**: addit. prune using minmaxdist

Christian Böhm

109
120

Alternative Approaches

- 5 different algorithms (...)



  - **Sorted distances recursive**:
    Before descending sort child pairs acc. to their mindist
    → fast get good distance for pruning. Analogous to
    [Roussopoulos, Kelley, Vincent: *Nearest Neighbor Queries*. SIGMOD Conf. 1995]
  - **Heap algorithm**:
    Similar to the algorithm by Hjaltason & Samet with some minor differences
- New strategies for ties and different tree height

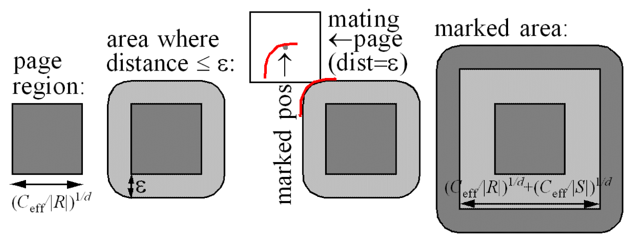
Christian Böhm

110
120

Modeling and Optimization

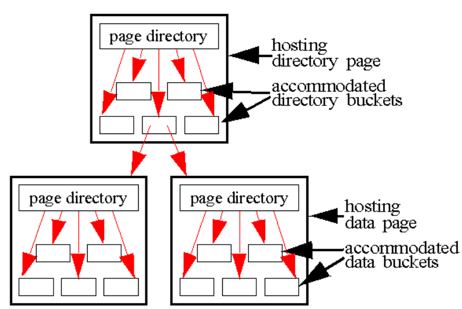
[Böhm, Kriegel: *A Cost Model and Index Architecture for the Similarity Join*, ICDE 2001]

- Mating probability of index pages:
  - Probability that distance between two pages ≤ ε
  - Two-fold application of Minkowski sum



Modeling and Optimization

- I/O cost:
  - High const. cost per page
  - Large capacity optimum
- CPU cost:
  - Low const. cost per page
  - Low capacity optimum



→ CPU-performance like CPU optimized index

→ I/O- performance like I/O optimized index

5 Conclusions

Summary

- Similarity join is a powerful database primitive
- Supports many new applications of
  - Data mining
  - Data analysis
- Considerable performance improvements

Summary

- Many different algorithms for the similarity join
  - Most for the **distance range join** (ε join)
  - Some approaches for **closest pair queries**
  - Important operation of **nearest neighbor join** has almost not been considered yet
- All 3 types of join have different applications
- Comparison of different ε join algorithms:
  - Mostly a competition for speed

Summary

- Only few other advantages/disadvantages:
  - Scalability:
    - MSJ and ε-kdB-tree have high main memory requirements in high-dimensional spaces
  - Existence of an index:
    - Actually no matter because R-trees can be fast constructed bottom-up. Construction time often much less than join time
    - Even if preconstructed indexes exist: Approaches based on sorting often better
  - No good criteria known for algorithm selection

Future Research Directions

- Applications:
  - Many standard data mining methods accelerable:
    - Outlier detection
    - Various clustering algorithms (e.g. obstacle clustering)
    - Hough transformation and similar analysis methods
    - ...
  - New data mining methods will become feasable:
    - Subspace clustering & correlation detection
    - Methods may become interactive
    - ...

Future Research Directions

- Algorithms
  - Sufficient research for ε join and closest pair query
  - Almost no convincing approaches for the *k*-NN-join
    Important database primitive for many applications
  - Parallel Algorithms
  - Non-vector metric data (e.g. text mining)
  - Approximative join algorithms
    - Similarity search: Approximative search often sufficient
    - Join performance could be considerably improved
  - ...

Future Research Directions

- Optimization of various critical parameters
  - Dimension
  - Replication
  - Index scan strategies
  - ...

Christian Böhm