6 Rank Aggregation and Top-k Queries

6.1 Computational Model for Top-k Queries over m-Dimensional Data Space

Assume sim. scoring of the form $score(q,d) = aggr\{s_i(q_i,d) \mid i = 1..m\}$
with an aggregation function $aggr : [0,1]^m \rightarrow [0,1]$
(or N_0 or R_0^+ instead of [0,1]
with the ***monotonicity*** property $\left(\forall i \in [1..m]:\ s_i(q,d') \ge s_i(q,d'')\right)$
$\Rightarrow\ aggr\{s_i(q,d') \mid i = 1..m\} \ge aggr\{s_i(q,d'') \mid i = 1..m\}$

Examples:

$score(q,d) = \sum_{i=1}^{m} s_i(q_i,d)$ $\quad$ $score(q,d) = \max\{s_i(q_i,d) \mid i = 1..m\}$

Key ideas:

1) process m index lists Li with ***sorted access*** to entries (d, si(q,d)) in descending order of si(q,d)
2) maintain for each candidate d a set E(d) of evaluated dimensions and a set R(d) of remaining dimensions, and a partial score
3) for candidate d with non-empty E(d) and non-empty R(d) consider looking up d in Li for all i∈R(d) by ***random access***
4) total execution cost = c_s * #sorted accesses + c_r * #random accesses

Wide Applicability of Algorithms

Ranked retrieval on
- ***multimedia data***: aggregation over features like color, shape, texture, etc
- ***product catalog data***: aggregation over similarity scores for cardinal properties such as year, price, rating, etc. and categorial properties such as
- ***text documents***: aggregation over term weights
- ***web documents***: aggregation over (text) relevance, authority, recency
- ***intranet documents***: aggregation over different feature sets such as text, title, anchor text, authority, recency, URL length, URL depth, URL type (e.g., containing „index.html“ or „~“ vs. containing „?“)
- ***metasearch engines***: aggregation over ranked results from multiple web search engines
- ***distributed data sources***: aggregation over properties from different sites
  e.g., restaurant rating from review site,
  restaurant prices from dining guide, driving distance from streetfinder

Fagin's Original Algorithm (FA) (PODS 96, JCSS 99)

Scan index lists in parallel (e.g. round-robin among L_1 .. L_m)
for each doc dj encountered in some list L_i do {
  E(dj) := E(dj) ∪ {i};
  lookup s_h(q,dj) in all lists L_h with h ∉E(dj) by random access;
  compute total score(q,dj); };
Stop when |{d | E(d)=[1..m]}| = k;
  // we have seen k docs in each of the lists

Execution cost is $\Omega\left(n^{\frac{m-1}{m}} \cdot k^{\frac{1}{m}}\right)$ with arbitrarily high probability

(for independently distributed Li lists)

Fagin's Threshold Algorithm (TA) (PODS 01, JCSS 03)

Scan index lists in parallel (e.g. round-robin among L_1 .. L_m)
for each doc dj encountered in some list L_i do {
  E(dj) := E(dj) ∪ {i};
  $high_i$:= si(q,dj);
  lookup s_h(q,dj) in all lists L_h with h ∉E(dj) by random access;
  compute total score(q,dj);
  min_k := minimum score among current top-k results;
  threshold := aggr($high_1$, ..., $high_m$);
};
Stop when $min_k \ge$ threshold
  // a hypothetical best document in the remainder lists
  // would not qualify for the top-k results

TA has much smaller memory cost than FA

Approximation TA

A ***θ-approximation*** T‘ for top-k query q with θ > 1
is a set T‘ of docs with:
- |T‘|=k and
- for each d‘∈T‘ and each d‘‘∉T‘: θ *score(q,d‘) ≥ score(q,d‘‘)

Modified TA:
  ...
  Stop when $min_k \ge$ aggr($high_1$, ..., $high_m$) / θ;

TA with Sorted Access Alone

computes only top k results
without necessarily knowing their total scores (cf. also Chapter 5)

Scan index lists in parallel (e.g. round-robin among L_1 .. L_m)
for each doc dj encountered in some list L_i do {
  E(dj) := E(dj) $\cup$ {i};
  $high_i$:= si(q,dj);
  bestscore(dj) := aggr{x1, ..., xm}
    with xi := si(q,dj) for i$\in$E(dj), $high_i$ for i $\notin$E(dj);
  worstscore(dj) := aggr{x1, ..., xm}
    with xi := si(q,dj) for i$\in$E(dj), 0 for i $\notin$E(dj);
  current top-k := k docs with largest worstscore;
  $worstmin_k$:= minimum worstscore among current top-k; };
Stop when bestscore(d | d not in current top-k results) $\leq$ $worstmin_k$;
Return current top-k;

Instance Optimality of TA

Definition:
For a class $\mathcal{A}$ of algorithms and a class $\mathcal{D}$ of datasets,
let cost(A,D) be the execution cost of A$\in\mathcal{A}$ on D$\in\mathcal{D}$.
Algorithm B is ***instance optimal*** over $\mathcal{A}$ and $\mathcal{D}$ if
  for every A$\in\mathcal{A}$ on D$\in\mathcal{D}$: cost(B,D) = O(cost(A,D)),
that is: cost(B,D) $\leq$ c*O(cost(A,D)) + c' with optimality ratio c.

Theorem:
TA is instance optimal over all algorithms that are based on sorted and random access to (index) lists.

6.2 Rank Aggregation

Consider sorted index lists L_i as permutations r_i of documents 1..n
(ranked lists containing all documents, not necessarily with scores)

A ***Kendall-optimal aggregation*** is a permutation r of [1..n] that minimizes the ***Kendall tau distance*** over all lists i$\in$[1..m]:

$$\sum_{i=1}^{m} K(r,r_i) \quad \text{with} \quad K(\pi,\sigma) := \left|\{(d,d') \mid (\pi(d) < \pi(d') \text{ and } \sigma(d') < \sigma(d)) \text{ or } (\pi(d') < \pi(d) \text{ and } \sigma(d) < \sigma(d'))\}\right|$$

A ***footrule-optimal aggregation*** is a permutation r of [1..n] that minimizes the ***footrule distance*** over all lists i$\in$[1..m]:

$$\sum_{i=1}^{m} F(r,r_i) \quad \text{with} \quad F(\pi,\sigma) := \sum_{j=1}^{n} |\pi(j) - \sigma(j)|$$

Computing a Kendall-optimal aggregation is NP-hard,
computing a footrule-optimal aggregation is possible in polynomial time.

Relationship to Median Rank

For permutations r_1, ..., r_m of docs $\in$ [1..n], let ***medrank(j)*** denote the median of {r_1(j), ..., r_m(j)}, i.e., a rank $\in$[1..n] with the property |{i | r_i(d) $\geq$ mr(d)}|=ceil(m/2) and |{i | r_i(d) $\leq$ mr(d)}|=floor(m/2)

Theorem:
If the medranks of docs are all distinct, then medrank yields a permutation that is footrule-optimal.
For permutations r_1, ..., r_m of [1..n] and a scoring function f: [1..n] $\rightarrow$ [0,1], medrank minimizes $\sum_{i=1}^{m}\sum_{j=1}^{n} |r_i(j) - f(j)|$

Theorem:
For permutations π, σ of [1..n]: K(π, σ) $\leq$ F(π, σ) $\leq$ 2K(π, σ).
A footrule-optimal aggregation is a 2-approximation to a Kendall-optimal aggregation.

Fagin's Median-Rank Algorithm (SIGMOD 03)

Find k documents d with highest median rank medrank(d) $\in$[1..n]

Initialize count(d) := 0 for all d;
Scan index lists in parallel (e.g. round-robin among L_1 .. L_m)
for each doc d encountered in some list L_i do
  count(d)++;
Stop when count(d) $\geq$ floor(m/2) +1 for at least k docs
// these are the top k results

The result of the Median-Rank algorithm satisfies the Condorcet criterion for robust voting:
  if a majority of voters prefers x over x'
  then x should be globally ranked higher than x'

Properties of the Median-Rank Algorithm

The Median-Rank algorithm is instance optimal over all algorithms that are based on sorted and random access to (index) lists.

For lists with independent rank distributions,
the expected scan depth of Median-Rank is $O(n^{1 - 2/(m+2)})$.

The algorithm can be generalized to arbitrary quantiles
(other than the 50% quantile).

Consider n points D={d1, ..., dn} in R^d and a query point q.
Randomly choose different unit vectors v1, ..., vm.
Produce ranked lists r1, ..., rm by projecting points onto v1, ..., vm and sorting d1, ..., dn by their distance to the projection of q.
Let z be the point with the best median rank over r1, ..., rm.
Then with probability at least 1-1/n we have:

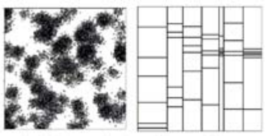
$$\|z-q\|_2 \leq (1+\epsilon)\|x-q\|_2 \quad \text{for all } x \in D$$

z is the ε-approximate nearest Euclidean-distance neighbor of q with high probability.

6.3 Mapping Top-k Queries onto Multidimensional Range Queries

1) Map top-k query for query point q into multidimensional range query with center q and an appropriate radius/width ε
2) Execute range query
3) Check if at least k results are returned; otherwise adjust ε and restart query

Key issue: how to estimate an appropriate radius/width ε
→ look up multidimensional histogram and construct synthetic relation R‘ with one tuple per histogram bucket cloned freq(bucket) times



from: N. Bruno, S. Chaudhuri, L. Gravano, Top-k Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation, ACM TODS 2002

Fig. 2. A 50-bucket histogram for the two-dimensional data set on the left.

Deriving Range Queries from Histograms (1)

Conservative strategy (NoRestarts):
1) for R‘ choose representative tuple t for bucket b such that t falls into b‘s region and has ***maximum distance*** to q
2) choose query width such that at least k tuples from R‘ are covered

Optimistic strategy (Restarts):
1) for R‘ choose representative tuple t for bucket b such that t falls into b‘s region and has ***minimum distance*** to q
2) choose query width such that at least k tuples from R‘ are covered

q=(20,15)
k=10

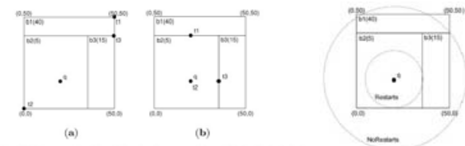


Fig. 3. A 3-bucket histogram H and the choice of tuples representing each bucket that strategies NoRestarts (a) and Restarts (b) make for query q.

Deriving Range Queries from Histograms (2)

Intermediate strategies (Inter1, Inter2):
set query width to: 2/3 width(NoRestarts) + 1/3 width(Restarts)
or to: 1/3 width(NoRestarts) + 2/3 width(Restarts)

Workload-adaptive strategy (Dynamic):
set query width to:
width(Restarts) + α (width(NoRestarts) – width(Restarts))
with α derived from (query-width, result-size) samples of the recent workload history (e.g., using linear regression)

Experimental Results

based on low-dimensional synthetic (Gauss, Array) and real data (US Census, cartographic data on forest coverage)

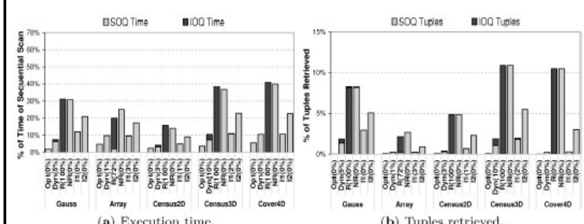


Fig. 16. Execution time and tuples retrieved for *Biased* workloads.

6.4 Multidimensional Index Structures for Similarity Search: R-Trees

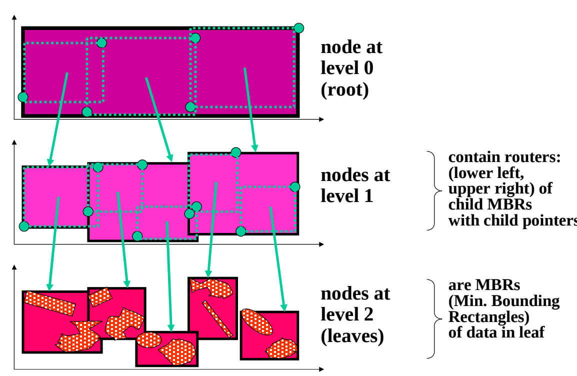
An R-tree is a B+-tree-like, page-structured, multiway search tree that manages
• multidimensional data points or rectangles as keys in leaves
• and **minimum bounding rectangles (MBRs)** as routers in inner nodes (represented by their lower left and upper right corners)

The key invariant of an R-tree is:
• the router MBR for subtree t is
• the MBR of all data points or MBRs in t.

A multidimensional range („window“) query traverses all subtrees whose MBRs intersect the query window.
The insertion of new data requires maintenance of the router MBRs, including possible node splits.

R-trees can manage multidimensional point data, as well as extended objects (e.g., polygons) by considering their MBRs

R-Trees



Range Query Algorithm for R-Tree

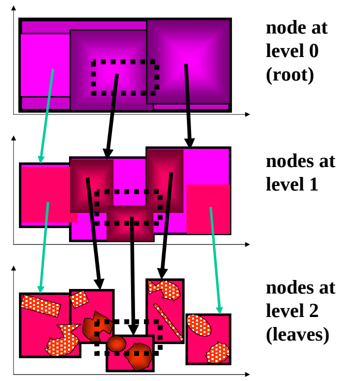
Multidimensional range („window“) query
with query MBR q:
  Find all data objects x that intersect with q
  (or all objects that are contained in q).

Algorithm:
  t := root of the R-tree;
  search (q, t);

search (q, n):
  if n is a leaf node then
    return all data objects x of n that intersect with q
  else
    T:= the set of router MBRs in n that intersect with q
    for each t in T do search (q, t) od;
  fi

Range Queries on R-Trees

Find all data that intersect a search window (a hyperrectangle that is parallel to the axes)



Bottom-Up Construction of R-Tree (1)

Given: n data points x1, ..., xn $\in [0,1]^m$
(e.g., the centers of the MBRs of the data objects)

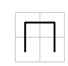
Consider an m-dimensional grid R = {i/k | i=0, ..., k-1}^{m}
with k cells per dimension, where k has the form 2^d,
and a space-filling curve ψ: R $\rightarrow$ {0, 1, ..., k^m},
where ψ is bijective and approximately preserves (Euclidean) distance

Bulk load algorithm:
1) Sort x1, ..., xn in descending order of ψ(x1), ..., ψ(xn)
2) Combine a suitable number of consecutive data points into one leaf node.
3) Construct the inner nodes and the root of the tree from the leaves in bottom-up manner.

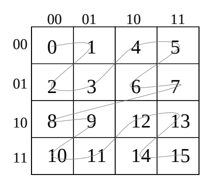
Bottom-Up Construction of R-Tree (2)

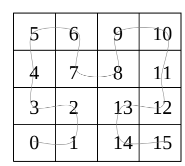
Suitable space-filling curves (fractals):

Peano curve (Z curve):
For point x with binary encoding
of its grid coordinates
x11, ..., x1d (in 1st dimension), ...
xm1, ..., xmd (in mth dimension):
ψ(x) = x11 x21 ... xm1 x12 ... xm2
... xm1 ... xmd
(bitwise interleaving)

Hilbert curve:
H1 for 2x2 grid: 

Hi for 2^ix2^i grid:
H1 curve for top level with
suitably rotated or mirrored
H(i-1) curve in each quadrant



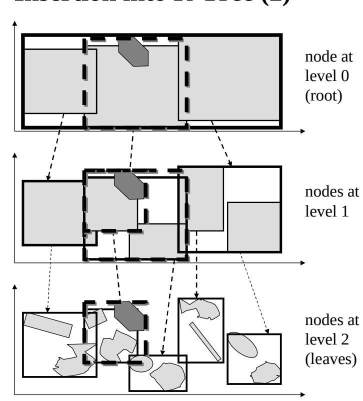


Insertion into R-Tree (1)

Insertion of MBR b (of a new data object):
  t := root of the R-tree;
  insert (b, t);

insert (b, n):
  if n is a leaf node then
    Insert b into n, recompute the MBR of n, and
    update the router MBR in the parent node of n;
    If n overflows, then split n into two nodes;
  else
    Determine among all router MBRs of n
    the most suitable MBR t (e.g., with regard to the volume or
    perimeter of the MBR for t $\cup$ b versus t);
    insert (b, t);
    Update the MBR of n if necessary;
  fi

Insertion into R-Tree (2)



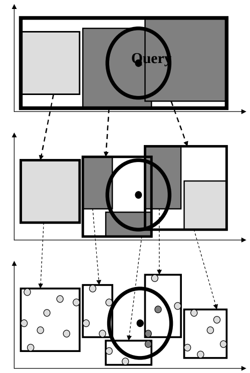
Split of R-Tree Node

Divide MBRs of node n (data objects or routers) onto two nodes n and n‘ such that
1) the sum of the volumes or perimeters of n and n‘ is minimal and
2) the storage utilization of n and n‘ does not drop below some specified threshold.

Heuristics:
Perform cluster analysis for the MBRs of n with 2 target clusters
or:
Determine among all MBRs of n two seed MBRs s and s‘ (e.g., those with maximum distance among all pairs) and assign MBR x to s or s‘ based on shorter distance
Store all MBRs assigned to s in n and all MBRs assigned to s‘ in n‘

ε-Neighborhood Search on R-Tree

Top-down search of all subtrees that intersect with a hypersphere with center q and radius ε (possibly approximated by searching the MBR of the hypersphere)



node at level 0 (root)

nodes at level 1

nodes at level 2 (leaves)

N-Nearest-Neighbor Search on R-Trees (1)

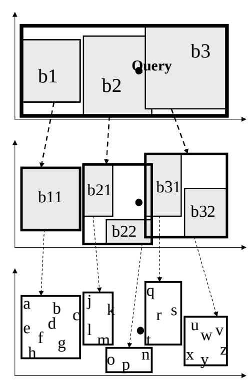
Find the N nearest neighbors of data point q

```
Algorithm:
NN: array [1..N] of record point: pointtype; dist: real end;
for i:=1 to N do NN[i].dist := ∞ od;
priority queue Q := root t;
repeat
  node n := first(Q);
  if n is a leaf node then
    for each p in n do if dist(p,q) < max(NN[1..N].dist)
                       then add p to NN fi od;
  else
    for each router MBR b of n do
      lowerbound := dist (q, closest point of MBR(n));
      if lowerbound < max(NN[1..N].dist)
      then insert(Q, b) fi
    od;
until Q is empty or dist(q, first(Q)) > max(NN[1..N].dist)
```


N-Nearest-Neighbor Search on R-Trees (2)

N = 4

NN: ---	Q: b2 b3 b1
NN: ---	Q: b3 b22 b21 b1
NN: ---	Q: b31 b22 b21 b32 b1
NN: t r s q	Q: b22 b21 b32 b1
NN: t n r p	Q: b21 b32 b1
NN: t n r p	Q: b32 b1
NN: t n r p	Q: ---



node at level 0 (root)

nodes at level 1

nodes at level 2 (leaves)

Literature

- R. Fagin, Amnon Lotem, Moni Naor: Optimal Aggregation Algorithms for Middleware, Journal of Computer and System Sciences Vol.66 No.4, 2003
- R. Fagin, R. Kumar, D. Sivakumar: Efficient Similarity Search and Classification via Rank Aggregation, SIGMOD Conf., 2003
- Ronald Fagin, Ravi Kumar, and D. Sivakumar: Comparing Top k Lists, SIAM Journal on Discrete Mathematics Vol.17 No.1, 2003
- R. Fagin, R. Kumar, K.S. McCurley, J. Novak, D. Sivakumar, J.A. Tomlin, D.P. Williamson: Searching the Workplace Web, WWW Conf., 2003
- R. Fagin: Combining Fuzzy Information: an Overview, ACM SIGMOD Record Vol.31 No.2, 2002
- N. Bruno, S. Chaudhuri, L. Gravano: Top-k Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation, ACM TODS Vol.27 No.2, 2002
- G. Hjaltason, H. Samet: Distance Browsing in Spatial Databases, ACM TODS Vol.24 No.2, 1999
- W. Kießling: Foundations of Preferences in Database Systems, VLDB Conf., 2002