Multi-dimensional data

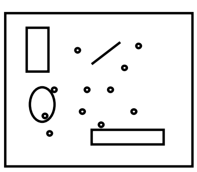
- Sources
  - Multimedia
  - Geographic data
  - Time series
  - Spatio-temporal data
  - Biological data
  - Text
  - Graph, tree
  - Stream

Multi-dimensional data

- Queries
  - exact match
  - k-nearest (kNN)
  - range
  - Combined range and kNN
  - Sub-object queries
    - Whole-whole
    - Part-whole
    - Part-part
    - Spatial predicates (contains, intersects, ..)
    - Spatial join
  - Multiple feature queries
  - Join queries

The indexing problem

- Given a collection of objects (points, lines, polygons, ...), organize them on disk, to answer spatial queries (range, nn, etc)



Index structures

- 1-d index structures
- In-memory index structures
- Disk-based index structures

Classification of index structures

- PAM vs. SAM
  - Point Access Methods
    - kd-tree, B+-tree, grid-file
  - Spatial Access Methods
    - R-tree, R*-tree
- Space partitioning vs. data partitioning
  - Space partitioning
    - Fanout independent of dimensionality
    - “Dead space”
    - No guarantee on space usage
    - kd-tree, quad tree, grid-file
  - Data partitioning
    - Fanout decreases with dimensionality
    - No “dead space”
    - Guaranteed space usage
    - R-tree, R*-tree

Disk-based index structures

- Grid files
- k-d-B tree
- R-tree
- R*-tree
- R+-tree
- X-tree
- Pyramid tree

Disk-based index structures

- Vamspilt R-tree
- SS tree
- SR-tree
- Hybrid tree
- VA-file
- M-tree

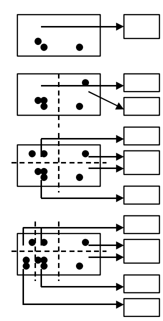
Requirements of disk-based index structures

- Page oriented access
  - Similar issues may arise in a multi-level cache hierarchy
- Time and space efficiency
  - Balanced structures
  - Good space utilization

Grid File

- Hashing methods for multidimensional points (extension of Extensible hashing)
- Idea: Use a grid to partition the space
  - many-to-one mapping from cells to pages

Grid File

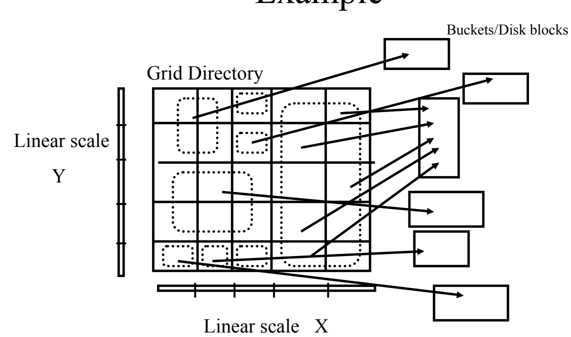


- Space Partitioning strategy
- Select dividers along each dimension. Partition space into cells
- Dividers cut all the way.
- Each cell corresponds to 1 disk page.
- Many cells can point to the same page.
- Cell directory potentially exponential in the number of dimensions

Grid File Implementation

- Dynamic structure using a grid directory
  - Grid array: a 2 dimensional array with pointers to buckets (this array can be large, disk resident) G(0,…, nx-1; 0, …, ny-1)
  - Linear scales: Two one-dimensional arrays that are used to access the grid array (main memory)
    - X(0, …, nx-1), Y(0, …, ny-1)

Example



Grid File Search

- Exact Match Search: Two I/Os assuming linear scales fit in memory.
  - first, use liner scales to determine the index into the cell directory
  - access the cell directory to retrieve the bucket address access the appropriate bucket
- Range Queries:
  - use linear scales to determine the index into the cell directory.
  - access the cell directory to retrieve the bucket addresses of buckets to visit.
  - access the buckets.

Grid File Insertions

- Determine the bucket into which insertion must occur.
- If there is space in bucket, insert.
- Else, split bucket (choice of dimension?)
  - Adjust cell directory
  - Adjust linear scales.
- Insertion of these new entries potentially requires a complete reorganization of the cell directory---expensive!!!
- Deletions are similar

R+ tree

- Ensure no overlap between nodes by splicing the data objects
- Need to search multiple paths to find the complete object
- Usually performs worse than R*-trees

Hilbert R-tree

- Use Hilbert order on centers of MBRs to order all siblings at a given level.
- Defer splits by moving entries among non-full s-1 sibling nodes or else splitting s nodes into s+1 nodes (called *s to s+1 split*).
  - Better space utilization

X-tree

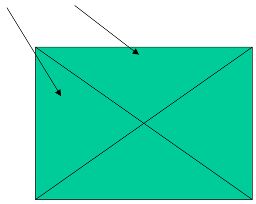
- Performance impacted by the amount of overlap between index nodes
  - Need to follow different paths
  - Alternative definitions of overlap
    - multi-overlap (count how many MBRs overlap in a certain region)
    - weighted overlap (count points)
  - Overlap approaches 100% for 6 dimensions
- Strategy: when an overflow occurs
  - Split into two nodes if overlap is small
  - Otherwise create a super-node with twice the capacity
  - Tradeoffs made locally over different regions of data space
    - Resembles an R*-tree when overlap is small
    - Resembles sequential access when overlap is large
- Experimental comparison with linear scan?

Pyramid Tree

- Designed for Range queries
- Map each d-dimensional point to 1-d value
- Build B+-tree on 1-d values
- A range query is transformed into 2d 1-d ranges
- Shown to be more efficient than X-tree, Hilbert R-tree, and sequential scan

Pyramid transformation

pyramids



- 2d pyramids with top at center of data-space (0.5,0.5,…,0.5)
  - (d-1) dimensional surfaces as base of pyramids
- points in different pyramids ordered based on pyramid id
- points within a pyramid ordered based on height
- value(v) = pyramid(v) + height(v)

VAMSplit R-tree

- Static structure for points
- Ensure no overlap at leaf level
- Split based on *variance* (V) and *approximate median* (AM)
  - Approximate median to maximize space usage
- Recursive partitioning from root node
- Good space utilization
- Extend to k-d-B trees
  - store splitting axis and value instead of MBR in index entries

SS-tree and SR-tree

- Use Spheres for index nodes (SS-tree)
  - Higher fanout since storage cost is reduced
- Use rectangles and spheres for index nodes
  - Index node defined by the intersection of two volumes
  - More accurate representation of data
  - Higher storage cost

K-D-B trees

- External memory kd-trees
  - K-D (K dimensions), B (bucket)
- Balanced structure.
- Nodes at each level partition the space.
- Internal (region) nodes grouped into buckets (pages).
  - Splits do not need to alternate
  - Sub-partitions at an internal node are like kd-trees.
- Splits defined using a separating hyper-plane
  - Point page split is simple: divide and insert an entry in parent region page
  - Region page split is harder (if we want to ensure good space utilization)
    - Downward propagation to redistribute data points
    - Upward propagation to parent

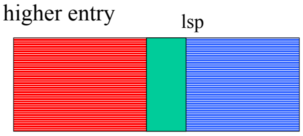
Bitmap index

- Given a collection of n objects, for a sparse attribute A,
  - For every value v of A (quantize if continuous)
    - Maintain a vector b of bits of size n
    - b(i) = 1 iff the value(A(i)) = v
- Use the bitmap vectors to support queries
- Number of vectors = (number of indexed attributes) * (number of values for the attribute)
- Support for logical operations by hardware
  - Attributes gender(M,F), permit_type(A,S,C)
  - How many female student permit holders?
- Divide each dimension into r ranges and maintain r bit vectors
  - Range query by a number of logical operations

Hybrid Tree

- Combine Data partitioning and Domain partitioning
- Split along a single dimension as in k-d tree
  - smaller index entry
- Allow overlaps as in R-tree
  - avoids costly cascading splits
  - Store high value of lower entry and low value of higher entry

lsp



rsp

lsp > rsp implies overlap

Hybrid tree

Index Structure	Number of dimensions used to split	Number of (k-1)-d hyperplanes used to split	Number of kd-tree nodes used to represent the split	Fanout	Degree of Overlap	Node Utilization Guarantee	Storage Redundancy
KDB-tree	1	1	1	High (Independent of k)	None	No	None
hB-tree	d $(1 \leq d \leq k)$	d	d	High (Independent of k)	None	Yes	Yes
R-tree	k	2k	-	Low for large k ($\propto \frac{1}{k}$)	High	Yes	None
Hybrid tree	1	1 or 2	1	High (Independent of k)	Low	Yes	None

Table 1. Splitting strategies for various index structures. k is the total number of dimensions.

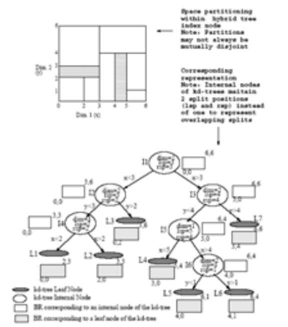
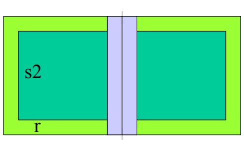


Figure 1. Mapping between each node and the corresponding BR. The shaded area represents overlap between BRs

Hybrid Tree

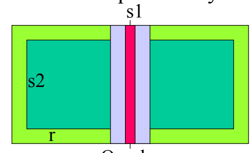
- Choice of splitting dimension for data (leaf) nodes
  - Choose dimension with maximum spread
    - differs from results reported in VAMSplit paper
  - minimize likelihood of access to both regions
  - assume uniform distribution
  - no overlap



$$\frac{\text{Prob of accessing both regions}}{\text{Prob of accessing any region}} = \frac{(2r)(s2+2r)}{(s1+2r)(s2+2r)} = \frac{2r}{(s1+2r)}$$

Hybrid Tree

- Choice of splitting position for data nodes
  - Mean as opposed to median (cubic regions)
    - while ensuring proper utilization
- Choice of splitting dimension for index nodes
  - Take overlap into account
  - use a probability distribution on query radius



$$\frac{\text{Prob of accessing both regions}}{\text{Prob of accessing any region}} = \frac{(2r+2p)(s2+2r)}{(s1+2r)(s2+2r)} = \frac{2r+2p}{(s1+2r)}$$

Hybrid Tree

- Choice of splitting position for index nodes
  - minimize overlap while maintaining space utilization
  - sort projections along split axis and cluster
    - sort entries into two lists based on left end point and right end point
    - alternate and assign from the sorted list into two partitions
    - once space utilization is achieved, place in order to minimize spread
  - needs to be done before determining splitting dimension in order to compute overlap

Hybrid Tree

- Dead space elimination
  - Tile the space uniformly and encode the extent of the live space
  - Space overhead
    - 4 bits per dimension
    - 64 dimensions
    - 256 bits = 32 bytes = 8 words per entry

Hybrid Tree

- Compared with hB-tree and SR-tree
- FOURIER dataset: 1.2 M 8-, 12-, and16-d vectors from Fourier transform of polygons
- COLOR dataset: 70K 16-, 32-, and 64-d color histograms from Corel Database
- Much better than sequential scan even at high dimensions

References

- (R-tree) A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD Conf.*, pp. 47–57, Boston, MA, June 1984.
- (R*-tree) N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM SIGMOD Conf.*, pp. 322–331, Atlantic City, NJ, May 1990.
- (X-tree) S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proc. of the 22nd International Conference on Very Large Data Bases (VLDB)*, pp. 28–39, Bombay, September1996.
- (Pyramid tree) S. Berchtold, C. Bohm and H.P. Kriegel, The Pyramid-Technique: Towards Breaking the Curse of Dimensionality, in Proc. of ACM-SIGMOD, 1998, pp. 142-153.
- (VA file) R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proc. of the 24th International Conference on Very Large Data Bases (VLDB)*, pp. 194–205, New York City, NY, August 1998.

References

- (K-D-B tree) J. T. Robinson. *The kdb-tree: A search structure for large multi-dimensional dynamic indexes*. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 10--18, 1981.
- (Hilbert R-tree) I. Kamel and C. Faloutsos. Hilbert R-tree: An Improved R-tree using Fractals. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pp. 500–509, Santiago, Chile, 1994.
- (SS tree) D. A. White and R. Jain. Similarity Indexing with the SS-tree. In *Proc. of IEEE 12th International Conference on Data Engineering*, pp. 516–523, 1996.
- (SR-tree) N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 369–380, May 1997.
- (VAMSplit R-tree) D. A. White and R. Jain. Similarity Indexing: Algorithms and Performance. In SPIE: Storage and Retrieval for Image and Video Databases IV , pages 62{75, 1996. Also available on the WWW at url http://vision.ucsd.edu/papers/sindexalg.
- (M-tree) P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB International Conference*, Athens, Greece, September 1997.