DECOUPLING: A FREE (?) SPATIAL LUNCH

HANAN SAMET

COMPUTER SCIENCE DEPARTMENT AND
CENTER FOR AUTOMATION RESEARCH AND
INSTITUTE FOR ADVANCED COMPUTER STUDIES
UNIVERSITY OF MARYLAND

COLLEGE PARK, MARYLAND 20742-3411 USA

Copyright © 2002 Hanan Samet

These notes may not be reproduced by any means (mechanical or electronic or any other) without express written permission of Hanan Samet
SORTING ON THE BASIS OF SPATIAL OCCUPANCY

- Decompose space from which data is drawn into regions called *buckets* (like hashing but preserves order)
- Interested in methods that are designed specifically for the spatial data type being stored
- Basic approaches to decomposing space
  1. minimum bounding rectangles
     - e.g., R-tree
     - good at distinguishing empty and non-empty space
     - drawbacks:
       a. non-disjoint decomposition of space
          • may need to search entire space
       b. inability to correlate occupied and unoccupied space in two maps
  2. disjoint cells
     - drawback: objects may be reported more than once
     - uniform grid
       a. all cells the same size
       b. drawback: possibility of many sparse cells
     - adaptive grid — quadtree variants
       a. regular decomposition
       b. all cells of width power of 2
     - partitions at arbitrary positions
       a. drawback: not a regular decomposition
       b. e.g., R+-tree
- Can use as approximations in filter/refine query processing strategy
MINIMUM BOUNDING RECTANGLES

- Rectangles grouped into hierarchies, stored in another structure such as a B-tree
- Drawback: not a disjoint decomposition of space
- Rectangle has single bounding rectangle, yet area it spans may be included in several bounding rectangles
- May have to visit several rectangles to determine the presence/absence of a rectangle
- Order \((m, M)\) R-tree
  1. between \(m \leq \lceil M/2 \rceil\) and \(M\) entries in each node except root
  2. at least 2 entries in root unless a leaf node
- Ex: order (2,3) R-tree
K-D-B-TREES

- Rectangular embedding space is hierarchically decomposed into disjoint rectangular regions
- No dead space in the sense that at any level of the tree, entire embedding space is covered by one of the nodes
- Blocks of k-d tree partition of space are aggregated into nodes of a finite capacity
- When a node overflows, it is split along one of the axes
- Originally developed to store points but may be extended to non-point objects represented by their minimum bounding boxes
- Drawback: in order to determine area covered by object, must retrieve all cells that it occupies
**R+-TREES**

- Rectangles are decomposed into disjoint subrectangles
- Subrectangles are aggregated hierarchically into larger disjoint rectangles
- Equivalent to a k-d-B-tree with difference that a rectangle at depth $i$ is a minimum bounding rectangle of the contained rectangles at depth $i+1$
- Advantage over k-d-B-tree in that false hits are reduced
- Same drawback of duplicate reporting as in k-d-B-tree
REGION QUADTREES

- Repeatedly subdivide until obtain homogeneous region
- For a binary image (BLACK ≡ 1 and WHITE ≡ 0)
- Can also use for multicolored data (e.g., a landuse class map associating colors with crops)
- Can also define data structure for grayscale images
- A collection of maximal blocks of size power of two and placed at predetermined positions
  1. could implement as a list of blocks each of which has a unique pair of numbers:
     - concatenate sequence of 2 bit codes corresponding to the path from the root to the block’s node
     - the level of the block’s node
  2. does not have to be implemented as a tree
     - tree good for logarithmic access
- A variable resolution data structure in contrast to a pyramid (i.e., a complete quadtree) which is a multiresolution data structure
PYRAMIDS

• Internal nodes contain summary of information in nodes below them

• Useful for avoiding inspecting nodes where there could be no relevant information
BV-TREES (FREESTON SIGMOD’95)

• Decouple hierarchy inherent to tree structure of a directory from the containment hierarchy associated with the process of recursive decomposition of the underlying space from which the data is drawn.

• Attempts to overcome problems when splitting a bucket in a structure such as a k-d-B-tree which causes the splitting to be propagated downward (e.g., pages A and G)

M. Freeston, A general solution of the $n$-dimensional B-tree problem, SIGMOD ’95, May 1995, 80–91
The interiors of any two regions \(a\) and \(b\) are either disjoint or one region is completely contained in the other region.

Splitting into contained regions is achieved via the extraction of smaller-sized regions thereby creating regions with holes.

Regions in the containment hierarchy are disjoint within each level and the boundaries of regions at different levels of the containment hierarchy are non-intersecting.

Disjointness of regions at a particular level in the decomposition hierarchy ensures that the outer boundaries of regions do not intersect (although they may touch or partially coincide).

Instead of propagating a split downward as in k-d-B-tree, the node that must be split is propagated upward.
BV-TREE EXAMPLE

Decomposition Hierarchy

1. Split into \( a_0 \) and \( b_0 \)

2. Add points so split space three times creating \( a_0, b_0, c_0, d_0, \) and \( e_0 \)

- notice: \( a_1 \) contains \( c_0 \) & \( d_0 \) while \( b_1 \) contains \( b_0 \) & \( e_0 \)
- instead of splitting \( a_0 \), it is propagated upward where it serves as a guard for \( a_1 \)
- guards are used to ensure that nondisjointness does not result in nonunique paths for point query
BV-TREE EXAMPLE (CONT)

3. Sequence of intermediate directory hierarchies

4. Final step
BV-TREE POINT QUERY

1. Root node:
   • find best match region $r$ containing point $q$
   • if $r$ is a guard, then find next best match region $n$
     and search it next with $r$ and any of its guards
   • else continue in the directory node pointed by $r$ and
     with a guard set formed by the current set of
     guards of $r$

2. Non-root node:
   • find best match region $r$ or the guard at this level
     from the previous step containing $q$
   • if best match region is a guard $g$ from before, then
     continue in $g$
   • else continue in the directory node pointed by $r$ and
     with a guard set formed by merging the current set
     of guards of $r$ with those from the previous step
EXAMPLE BV-TREE POINT QUERY \((w)\)

- Search for \(w\):

1. Search directory at level 3 and \(a_2\) is best match
2. Continue at \(a_2\) with guard set \(a_0\) and \(a_1\)
3. No match in entries in \(a_2\) (neither \(c_1\) nor \(d_1\) contain \(w\)) and use guard \(a_1\) from previous level as best match
4. Continue at \(a_1\) with guard set \(a_0\)
5. No best match in \(a_1\) as neither \(c_0\) nor \(i_0\) nor \(j_0\) contain \(w\), but guard \(a_0\) contains \(w\) and we are done
EXAMPLE BV-TREE POINT QUERY (x)

• Search for x:

1. Search directory at level 3 and guard $a_1$ is best match
2. Find smallest nonguard region containing $a_1$ which is $b_2$
3. Continue at $b_2$ with guard set $a_0$ and $a_1$
4. Best match is $b_1$ but the guard $a_1$ at this level in this set is a better match and search is continued in $a_1$ with a guard set $a_0$
5. No match in entries in $a_1$ as neither $c_0$ nor $i_0$ nor $j_0$ contain $x$, but guard $a_0$ contains $x$ and we are done
EXAMPLE BV-TREE POINT QUERY \((y)\)

- Search for \(y\):

1. Search directory at level 3 and guard \(a_0\) is best match
2. Find smallest nonguard region containing \(a_0\) which is \(b_2\)
3. Continue at \(b_2\) with guard set \(a_0\)
4. Best match is \(b_1\) and there is no guard at this level from the previous step and search is continued in \(b_1\) with a guard set \(a_0\) while ignoring \(e_0\) as it does not contain \(y\)
5. No match in entries in \(b_1\) as neither \(b_0\) nor \(n_0\) contain \(y\), but guard \(a_0\) contains \(y\) and we are done
EXAMPLE BV-TREE POINT QUERY (z)

- Search for z:

1. Search directory at level 3 and guard $a_0$ is best match
2. Find smallest nonguard region containing $a_0$ which is $b_2$
3. Continue at $b_2$ with guard set $a_0$
4. Best match is $b_1$ and there is no guard at this level from the previous step and search is continued in $b_1$ with a guard set $e_0$ while ignoring $a_0$ as $e_0$ is contained in it and $e_0$ contains $z$
5. No match in entries in $b_1$ as neither $b_0$ nor $n0$ contain $y$, but guard $e_0$ contains $z$ and we are done
BV-TREE SUMMARY

• Search always descends the same number of levels in the decomposition hierarchy

• The fact that the search process carries the guards as the tree is descended guarantees that only one path is followed in the point query which compensates for the fact that there is multiple coverage in the BV-tree as in the R-tree

• The search proceeds by levels in the decomposition hierarchy even though it appears to be backtracking in the directory hierarchy (e.g., when making use of guards $a_0$ and $a_1$ in search for $x$ while searching $b_2$)

• Containment hierarchy in BV-tree appears to be a special case of an R-tree in sense that the containment hierarchy serves as a constraint on the relationship between the regions that form the directory structure

• The constraint is that any pair of bounding rectangles of two sons $a$ and $b$ of an R-tree node $r$ must be either disjoint or one son must be completely contained in the other son

• If build an R-tree using these constraints, we must be sure upon insertion of a point that does not lie in the area of some existing rectangles so that the expanded region does not violate the containment rules

• Of course, if we do this, then we no longer have the property that all leaf nodes are at the same level

• We see a similar variance when examining the PK tree
PK TREES (Wang, Yang, and Muntz)

1. Similar to bucketing but very different
   - bucketing: split when more than $k$ items in a block (bucket capacity)
   - PK tree: group when fewer than $k$ items in a block ($k$-instantiation)

2. Generally want buckets to be the size of a disk page (i.e., large fanout)
   - let leaf nodes be buckets and thus a large fanout
   - want large fanout on nonleaf nodes as well
   - can increase fanout of nonleaf nodes by aggregating nonleaf nodes at successive levels of decomposition (e.g., quadgrid)
   - drawback of quadgrid is all blocks have same size
   - PK tree: not all nodes at same level of directory hierarchy span identically-sized regions

STRUCTURE OF PK TREES

• Decouples the tree structure of the partition process of the underlying space from that of the node hierarchy (i.e., the grouping process of the nodes resulting from the partition process) that makes up the directory

1. decomposing space into quadtree blocks, representing them by locational codes, and storing them in nodes of a B+-tree is an example of decoupling

2. shape of region spanned by node of B+-tree does not necessarily have the same shape as the blocks that have been aggregated while this is the case for the PK tree

3. BV-tree also decouples but motivation is different as it was to overcome drawback of disjoint space partitioning methods with respect to search
PK TREE EXAMPLE

1. Choose a partition process (e.g., PR quadtree where each block contains just one point)

2. Choose value of $k$ and in bottom-up manner remove all nonleaf nodes with less than $k$ nonempty sons while linking remaining nonempty children to their grandparent

• result is a PK PR quadtree
BUCKETING VERSUS INSTANTIATION

1. Bucketing is a top-down process while instantiation is a bottom-up process

2. Bucketing finds maximum enclosing block for $k$ or fewer objects, while instantiation finds minimum enclosing block for at least $k$ objects
PROPERTIES OF PK TREES

1. When $k=2$ result is equivalent to path compression (i.e., path-compressed PR quadtree or path-compressed PK k-d tree)

2. Can make internal nodes very large so each node can be stored on a disk page

3. A two-dimensional PK PR quadtree will have between $k$ and $4 \cdot (k-1)$ sons
   - PK PR k-d tree has a minimum of $k$ sons and a maximum of $2 \cdot (k-1)$ sons
   - resembles classical definition of B-tree (and R-tree) in terms of having nodes with a variable capacity while each node is guaranteed to be at least half full
   - not necessarily balanced

5. Similar to BD-tree but uniquely defined
   - but non-uniqueness of BD-tree enables variation so that a more balanced structure can be achieved
   - BD-tree decompose space into holey bricks while PK tree yields squares or other regular shapes
   - nodes other than leaf nodes have fanout of 2 while not so for PK tree
PK TREE INSERTION

• Can be done in top-down manner and hence structure is dynamic

• However, may have to apply checks for $k$-instantiation and $k$-deinstantiation at each level of the PK tree during the grouping process

• Example worst-case with PK MX quadtree for $k=5$ for insertion of P

• Tree depth grows by one level
PK TREE DELETION

• Can be done in top-down manner and hence structure is dynamic

• However, may have to apply checks for $k$-deinstantiation and $k$-instantiation at each level of the PK tree during the grouping process

• Example worst-Case with PK MX quadtree for $k=5$ for deletion of P

- Tree depth shrinks by one level
PK TREE DISCUSSION

1. PK tree depth can increase by at most one during insertion while depth increase in PR quadtree and PR k-d tree depend on the minimum separation between the newly inserted point and existing points.

2. Maximum depth of PK tree is $O(N)$ and this is the worst case cost of search and insertion although the expected cost is shown to be $O(\log N)$ under certain data distributions.

3. Some sensitivity to skewness.

4. $O(N)$ space requirements in contrast to most methods based on a regular decomposition where there is a dependence on the resolution of the embedding space (i.e., the maximum level of decomposition).

5. When using a PK PR k-d tree we get analog of B-tree while having rectangular blocks whose sides are a power of two although the structure is not balanced (i.e., no guarantee of $O(N)$ search time)
   - have both a minimum and maximum bucket occupancy.