GiS: A Generalized Search Tree for Database Systems

Joe Hellerstein

UC Berkeley
Road Map

- Motivation
- Intuition on Generalized Search Trees
- Overview of GiST ADT
- Example indices: integers, polygons & sets
- Implementation challenges
- Open problems in indexing research
Indexing in OO/OR Systems

- Quick access to user-defined objects
- Support queries natural to the objects
- Two previous approaches
  - Specialized Indices ("ABCDEFG-trees")
    » redundant code: most trees are very similar
    » concurrency control, etc. tricky!
  - Extensible B-trees & R-trees (Postgres/Illustra)
    » B-tree or R-tree lookups only!
    » E.g. ‘WHERE movie.video < ‘Terminator 2’
A Third Approach

- A generalized search tree. Must be:
  - Extensible in terms of queries
  - General (B+-tree, R-tree, etc.)
  - Easy to extend
  - Efficient (match specialized trees)
  - Highly concurrent, recoverable, etc.
Uses for GiSTs

■ New indexes needed for new apps...
  – find all superset of $S$
  – find all molecules that bind to $M$
  – your favorite query here (multimedia?)

■ ...and for new queries over old domains:
  – find all points in region from 12 to 2 o’clock
  – find all strings that match R. E.
Database Search Trees from 50,000 feet
Database Search Trees from 50,000 feet
Database Search Trees from 40,000 feet

Internal Nodes (directory)

Leaf Nodes (linked list)
Database Search Trees from 30,000 feet

Internal Nodes (directory)

Leaf Nodes (linked list)

key1   key2   ...

jmh - GiST
GiST: Generalized Search Tree

- Structure: balanced tree of \((p, \text{ptr})\) pairs
  - \(p\) is a key “predicate”
  - \(p\) holds for all objects below \text{ptr}
  - keys on a page may overlap

- Key predicates: a user-defined class
  - This is the only extensibility required!
Key Methods

- **Search:**
  - **Consistent**($E, q$): $E.p \land q$? (no/maybe)

- **Characterization**
  - **Union**($P$): new key that holds for all tuples in $P$

- **Categorization**
  - **Penalty**($E_1, E_2$):
    penalty of inserting $E_2$ in subtree $E_1$
  - **PickSplit**($P$): split $P$ into two groups of entries
Search

■ General technique:
  – traverse tree where Consistent is TRUE

■ For range predicates on ordered domain:
  – user specifies IsOrdered
  – user registers Compare($p_1, p_2$) operator
  – methods ensure ordered, non-overlapping keys
  – traverse leftmost Consistent branch
  – scan right across bottom.
Insert

- descend tree along least increase in Penalty
- if there’s room at leaf, insert there
- else split according to PickSplit
- propagate changes using Union

Notes:
- on overflow, can do R*-tree style reinsert
- for ordered keys, Penalty needs to keep order
Delete

- find the entry via Search, and delete it
- propagate changes using Union
- on underflow:
  - if ordered keys, do B+-tree style
    borrow/coalesce
  - else reinsert stuff on page and delete page
GiSTS over $\mathbb{Z}$ (B+-trees)

- Logically, keys represent ranges $[x,y)$
- Queries: **Contains**($[a,b)$, $v$)
- **Consistent**($E, q$): $(x < b) \land (y > a)$
- **Union**($P$): $[\text{MIN}(x_i), \text{MAX}(y_i))$
- **Penalty**($E_1, E_2$):
  - return $\text{MAX}(y_2 - y_1, 0) + \text{MAX}(x_1 - x_2, 0)$
  - if $E_1$ is leftmost or rightmost, drop a term
- **PickSplit**($P$): split evenly in order
Key Compression

- Keys may take up too much room on a page
- Two extra key methods:
  - Compress($E$)/Decompress($E$)
- Compression can be lossy:
  - over-generalization OK
A B+-tree Page

Logical Representation:

[∞, 40) [40, 60) [60, 137) [137, 201) [201, ∞)

Physical Representation (compressed):

<null> 40 60 137 201
B+-tree Compression

- **Compress**($E=([x,y], \text{ptr})$):
  - if $E$ is leftmost return NULL, else return $x$

- **Decompress**($E=(\pi, \text{ptr})$):
  - if $E$ is leftmost, let $x = -\infty$, else let $x = \pi$.
  - if $E$ is rightmost, let $y = \infty$, else let $y$ be the value stored in the next key on the right.
  - if $E$ is rightmost on a leaf page, let $y = x+1$. 
GiSTs over $R^2$ (R-tree)

- Logically, keys represent bounding boxes
- Queries: Contains, Overlaps, Equals
- Consistent($E,q$): does $E.p$ overlap $q$?
- Union($P$): bounding box of all entries
- Compress($E$): form bounding box
- Decompress($E$): identity function
- Penalty($E,F$): size(Union(${E,F}$)) - size($E$)
- PickSplit($P$): R-tree or R*-tree methods
GiSTs over $P(\mathbb{R})$ (RD-tree)

- Logically, keys represent bounding sets
- Queries: Contains, Overlaps, Equals
- Consistent($E,q$): does $E.p \cap q = \emptyset$?
- Union($P$): set-union of keys
- Compress($E$): Bloom filters, rangesets, etc.
- Decompress($E$): match compress
- Penalty($E,F$): $|E.p \cup F.p| - |E.p|
- PickSplit($P$): R-tree algorithms
An RD-tree

{CS1, CS11, Music1, Music2, Math221, Math22, Math223}

{CS1, Bus101, Bus102, Bus103, Ec121, Ec122, Ec123}

{CS1, CS786, CS888, Math221, Music1, Music788}

{Bus101, Bus102, Bus103, CS1}
{Bus101, Ec121, Ec122, Ec123}
{CS1, Bus101, Ec121}

{CS1, CS11, Math221}
{Music1, Music2, CS1}
{CS1, Math221, Math22, Math223}

{Music1, CS1, Math221}
{Music788, CS888, CS786}
{CS1}
Implementation Issues

- In-memory efficiency: Node subclass
- Concurrency, Recovery, Consistency
  - Kornacker & Banks, VLDB95
- Variable-Length Keys
- Bulk Loading
- Optimizer Integration
- Extensibility & Efficiency
GiST Performance

- B+-trees have O(log n) performance
- R-trees, RD-trees have no such guarantee
  - search may have to traverse multiple paths
  - worst-case O(2n) to traverse entire tree
  - aggravated by random I/O: much worse than scan!

SO: when does it pay to build/use an index?
GiST Performance, cont.

- As a first cut, look at 2 parameters:
  - data overlap & compression loss
- Experiment with Illustra’s R-trees
  - Comb sets: \{[1,10], [10001,10010], \ldots\}
  - 30 data sets, each of 10,000 combs
  - vary data overlap, numranges (compression)
  - 5 queries per dataset, searching for comb teeth
GiST Performance, cont.

![Graph showing Avg. Number of I/Os vs Compression Loss vs Data Overlap](image)

- **Compression Loss**
  - X-axis: Compression Loss
  - Y-axis: Data Overlap
  - Z-axis: Avg. Number of I/Os

- **Avg. Number of I/Os**
  - Range: 0 to 5000

- **Data Overlap**
  - Range: 0 to 1
Future Directions in Indexing

- Indexability theory:
  - when is an index useful? Papadimitriou?

- New things to index! Queries over:
  - sets, sequences/text (REs), graphs, multimedia, molecular structures...

- Lossy compression techniques

- Algorithmic improvements?
  - (R*-tree techniques?)
The Gist of the GiST

- Boil search trees down to their essence.
- Unify B+-tree, R-tree, etc. in one ADT.
- Extensible in terms of data and queries.
- Opens research on indexability.
Status

■ Prototype implementation in Postgres95
  – currently no variable-length keys, concurrency
■ Illustra/Informix port?
■ General purpose C++ library planned
■ Papers, etc. at:
  – http://www.cs.berkeley.edu/~jmh/