“Generalized Search Trees for Database Systems”
by Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeiffer

Motivation
- Extensible data and query models
- Ease construction of index structures for new data and query types
  > adding new data types
- Generalized tree structure for database systems
  > maintaining data and asking queries

Generalizing DB Search Trees
- Balanced tree
- High fanout
- Keys \rightarrow predicates, may overlap

Outline
- Motivation
- Generalized Search Trees (GiST)
- Algorithms
- Applications
- GiST Limitations, Extensions
- Conclusions

Running Example: Images

Warning: This index scheme may not be suitable for indexing real images.
Properties

- Internal nodes (other than the root) have between \( km \) and \( M \) index entries
  - \( k \): minimum fill factor, < \( \frac{1}{2} \)
  - entry: (key/predicate, pointer)

- Root has at least two children unless it is a leaf node
- All leaf nodes appear on same level
  - Balance tree
  - Bound height of tree

Images

- \( \text{size} < 10K \)
- \( 10K < \text{size} < 50K \)
- \( \text{image} \text{-} \text{contains} \text{purple} \)
- \( \text{purple} \)
- \( \text{yellow} \)

Key Methods

- Methods used by GIST to maintain invariants
- Implemented by index developer
- Application-specific policies
**Key Methods**

- Consistent(Entry E, Predicate q)
- Union(Entry E[])
- Compress(Entry E)
- Decompress(Entry E)
- Penalty(Entry E1, Entry E2)
- PickSplit(Entry E[])

**Recall:** Entry is (predicate p, pointer ptr)

**Consistent(Entry E, Predicate q)**
- returns false if p AND q are guaranteed unsatisfiable
- determines which tree(s) to search
- false positives but no false negatives

**Image Search**

Query: Image files that contain purple

size \( < 10 \text{K} \) \( \leq \text{size} < 50 \text{K} \) ...  

consistent for both paths  

contains_purple ... owned_by_Sara ...

purple yellow purple_blue yellow_green

**B+-tree Compress**

Lossless compression, from paper

(able, apache) (apache, apple) ...

Compress predicates

able apache ...
**Key Methods**

- Penalize (Entry E1, Entry E2)
  - Penalty for inserting E2 into E1’s subtree
    - Local not global penalty
  - Used for deciding where to insert entries or where to split a predicate
  - R-tree examples \( \Rightarrow \) minimizing increased area, minimizing overlap, minimizing perimeter

**Outline**

- Motivation
- Generalized Search Trees (GiST)
- Algorithms
- Applications
- GiST Limitations, Extensions
- Conclusions
Algorithms

- Searches, inserts, and deletes are based on the implemented key methods
  - generic algorithms for updating and accessing index structures
  - application-specific information is extracted into key methods
- Algorithms are handled by GiST, not defined by user

Algorithm: Search

- Search( GiST R, Predicate q )
  - start at root
  - go down path or paths where key predicates are consistent q
  - reach leaf → final consistency check
  - return array of objects or array of object pointers
- Only use Consistent key method
- Generalization → exact match, range queries

Algorithm: Insert

- Insert( GiST R, Entry E )
  - start at root
  - find leaf where E should be inserted
    - may require choosing among several different subtrees at each level along path
  - insert E
  - may require splitting leaf node and propagating/adjusting keys up the tree

Algorithm: Choose Subtree

- Calculate penalty of inserting entry in subtree
  - domain-specific penalty
  - minimize penalty locally not globally

Image Search

Query: Image files that contain purple

R-tree Insert

- Insert R₉ into R-tree
  - pick a region containing R₉ and follow the child pointer
Algorithm: Split

- Union on new elements → create a new key
- Modify old key → reduce overlap, tighter control
- Adjust keys up touched path

Applications

- GiST confines application specific code to six key methods
- Implementing a new tree only requires coding of key methods. GiST handles insert, delete and search
- Paper discusses B+, R and RD Tree implementations

Algorithm: Delete

- Delete( GiST G, Predicate q )
  - find element based on q
    - constrain query to return one element
  - delete
  - maintain balance, invariants up tree

Application: B+-tree

- Contains( (x, y), v )
  - If x ≤ v < y, return true; otherwise, return false
- Equal( x, v )
  - If x = v, return true; otherwise, return false

Outline

- Motivation
- Generalized Search Trees (GiST)
- Algorithms
- Applications
- GiST Limitations, Extensions
- Conclusions

Application: B+-tree

- Consistent( E, q )
  - If p=Contains([x_p, y_p], v) AND q=Contains([x_q, y_q], v), return true if (x_p < x_q) AND (y_q > x_p), false otherwise
  - If p=Contains([x_p, y_p], v) AND q=Equal(x_q, v), return true if x_p ≤ x_q ≤ y_p, false otherwise
- Union( (E_1, ..., E_n) )
  - E = ([x, y], ptr)
  - return [Min(x_1, ..., x_n), Max(y_1, ..., y_n)]
**Application: B+-tree**

- Compress \((E = (x, y, \text{ptr}))\)
  - Return \(x\), unless \(E\) is the leftmost key on an internal node (return a 0-byte object)
- Decompress \((E = (\pi, \text{ptr}))\)
  - Construct an interval \([x, y]\)
  - If \(E\) is leftmost key in internal node, \(x = -\infty\); otherwise, \(x = \pi\)
  - If \(E\) is rightmost key in internal node, \(y = \infty\); otherwise, \(y = \text{nextKey}()\)

**Outline**

- Motivation
- Generalized Search Trees (GiST)
- Algorithms
- Applications
- GiST Limitations, Extensions
- Conclusions

**Application: B+-tree**

- Penalty \((E = ([x_1, y_1], \text{ptr}_1), F = ([x_2, y_2], \text{ptr}_2))\)
  - If \(E\) is leftmost pointer on its node, return \(\max(y_2 - y_1, 0)\)
  - If \(E\) is rightmost pointer on its node, return \(\max(x_1 - x_2, 0)\)
  - Otherwise, return \(\max(y_2 - y_1, 0) + \max(x_1 - x_2, 0)\)

**GiST Limitations/Extensions**

- Aggregate queries
- Nearest-neighbor, i.e., “like” queries
  - both addressed in “Generalizing ‘Search’ in Generalized Search Trees”, ICDE 1999
- Concurrency, recovery implementation
  - naïve: strict 2PL
  - addressed in “Concurrency and Recovery in Generalized Search Trees”, SIGMOD 1997

**Application: B+-tree**

- PickSplit( \(P\))
  - \(P = \{ E_1, ..., E_n \}\)
  - \(E_i < E_j\) for \(i < j\)
  - Return \(P_1 = \{ E_1, ..., E_{\text{floor}(n/2)} \}\) and \(P_2 = \{ E_{\text{ceiling}(n/2)}, ..., E_n \}\)
  - Guarantees a minimum fill factor of \(M2\)

**GiST Conclusions**

- Identify the fundamentals of search trees
- One ADT describes many search trees, e.g. B+-tree, R-tree, etc.
- Allows extensible data and query types
Discussion

- Questions?
- Time for quiz!

Quiz

- Define a GiST. What are its primary benefits?
- Would you use a GiST to implement a new DB search tree? Specifically, consider ease of implementing your tree. What are the tradeoffs?
- What is Sara’s favorite color?
- By how much did this presentation improve your understanding of GiST?
  Scale: [1. more confused than ever, 5. damn-near an expert]