External Memory Geometric Data Structures

Lars Arge
Duke University

June 29, 2002

Summer School on Massive Datasets
So Far So Good

• **Yesterday** we discussed “dimension 1.5” problems:
  – Interval stabbing and point location

• We developed a number of useful tools/techniques
  – Logarithmic method
  – Weight-balanced B-trees
  – Global rebuilding

• On **Thursday** we also discussed several tools/techniques
  – B-trees
  – Persistent B-trees
  – Construction using buffer technique
**Interval Management**

- Maintain $N$ intervals with unique endpoints dynamically such that stabbing query with point $x$ can be answered efficiently

- Solved using **external interval tree**
- We obtained the same bounds as for the $Id$ case
  - Space: $O(N/B)$
  - Query: $O(\log_B N + T/B)$
  - Updates: $O(\log_B N)$ I/Os
External memory data structures

Interval Management

- External interval tree:
  - Fan-out $\Theta(\sqrt{B})$ weight-balanced B-tree on endpoints
  - Intervals stored in $O(B)$ secondary structure in each internal node
  - Query efficiency using filtering
  - Bootstrapping used to avoid $O(B)$ search cost in each node
    * Size $O(B^2)$ underflow structure in each node
    * Constructed using sweep and persistent B-tree
    * Dynamic using global rebuilding
3-Sided Range Searching

- Interval management corresponds to simple form of 2d range search

- More general problem: Dynamic 3-sided range searching
  - Maintain set of points in plane such that given query \((q_1, q_2, q_3)\), all points \((x,y)\) with \(q_1 \leq x \leq q_2\) and \(y \geq q_3\) can be found efficiently
3-Sided Range Searching: Static Solution

- **Construction**: Sweep top-down inserting $x$ in persistent B-tree at $(x,y)$
  - $O(N/B)$ space
  - $O(N/B \log_B N)$ I/O construction using buffer technique

- **Query** $(q_1, q_2, q_3)$: Perform range query with $[q_1, q_2]$ in B-tree at $q_3$
  - $O(\log_B N + T_B)$ I/Os

- **Dynamic using logarithmic method**
  - Insert: $O(2 \log_B N)$
  - Query: $O(2 \log_B N + T_B)$

- Improve to $O(\log_B N)$? Deletes?
External memory data structures

**Internal Priority Search Tree**

- **Base tree on** $x$-coordinates with nodes augmented with points
- **Heap on** $y$-coordinates
  - Decreasing $y$ values on root-leaf path
  - $(x,y)$ on path from root to leaf holding $x$
  - If $v$ holds point then $parent(v)$ holds point

![Diagram of Internal Priority Search Tree](image-url)
**Internal Priority Search Tree**

Insert (10, 21)

- Linear space
- **Insert** of \((x, y)\) (assuming fixed \(x\)-coordinate set):
  - Compare \(y\) with \(y\)-coordinate in root
  - Smaller: Recursively insert \((x, y)\) in subtree on path to \(x\)
  - Bigger: Insert in root and recursively insert old point in subtree

⇒ \(O(\log N)\) update
External memory data structures

**Internal Priority Search Tree**

- **Query** with \((q_1, q_2, q_3)\) starting at root \(v\):
  - Report point in \(v\) if satisfying query
  - Visit both children of \(v\) if point reported
  - Always visit child(s) of \(v\) on path(s) to \(q_1\) and \(q_2\)

\[\Rightarrow O(\log N + T)\text{ query}\]
External memory data structures

Externalizing Priority Search Tree

• Natural idea: Block tree
• Problem:
  – $O(\log B N)$ I/Os to follow paths to to $q_1$ and $q_2$
  – But $O(T)$ I/Os may be used to visit other nodes ("overshooting")
  $\Rightarrow O(\log_B N + T)$ query
External memory data structures

Externalizing Priority Search Tree

• Solution idea:
  – Store $B$ points in each node ⇒
    * $O(B^2)$ points stored in each supernode
    * $B$ output points can pay for “overshooting”
  – Bootstrapping:
    * Store $O(B^2)$ points in each supernode in static structure
External Priority Search Tree

- **Base tree**: Weight-balanced B-tree on $x$-coordinates ($a,k=B$)
- **Points in “heap order”**:  
  - Root stores $B$ top points for each of the $\Theta(B)$ child slabs  
  - Remaining points stored recursively  
- **Points in each node stored in “$O(B^2)$-structure”**  
  - Persistent B-tree structure for static problem

\[ \downarrow \]

Linear space
External Priority Search Tree

- **Query** with \((q_1, q_2, q_3)\) starting at root \(v\):
  - Query \(O(B^2)\)-structure and report points satisfying query
  - Visit child \(v\) if
    * \(v\) on path to \(q_1\) or \(q_2\)
    * All points corresponding to \(v\) satisfy query
External Priority Search Tree

- Analysis:
  - $O(\log_B B^2 + \frac{T_v}{B}) = O(1 + \frac{T_v}{B})$ I/Os used to visit node $v$
  - $O(\log_B N)$ nodes on path to $q_1$ or $q_2$
  - For each node $v$ not on path to $q_1$ or $q_2$ visited, $B$ points reported in parent($v$)

$O(\log_B N + \frac{T_v}{B})$ query
External Priority Search Tree

- **Insert** \((x, y)\) (assuming fixed \(x\)-coordinate set – static base tree):
  - Find relevant node \(v\):
    * Query \(O(B^2)\)-structure to find \(B\) points in root corresponding to node \(u\) on path to \(x\)
    * If \(y\) smaller than \(y\)-coordinates of all \(B\) points then recursively search in \(u\)
  - Insert \((x, y)\) in \(O(B^2)\)-structure of \(v\)
  - If \(O(B^2)\)-structure contains \(>B\) points for child \(u\), remove lowest point and insert recursively in \(u\)

- **Delete**: Similarly
External Priority Search Tree

• Analysis:
  – Query visits $O(\log_B N)$ nodes
  – $O(B^2)$-structure queried/updated in each node
    * One query
    * One insert and one delete

• $O(B^2)$-structure analysis:
  – Query: $O(\log_B B^2 + B / B) = O(1)$
  – Update in $O(1)$ I/Os using update block and global rebuilding

\[ O(\log_B N) \text{ I/Os} \]
Removing Fixed $x$-coordinate Set Assumption

- **Deletion:**
  - Delete point as previously
  - Delete $x$-coordinate from base tree using global rebuilding
  \[ \Rightarrow O(\log_B N) \text{ I/Os amortized} \]

- **Insertion:**
  - Insert $x$-coordinate in base tree and rebalance (using splits)
  - Insert point as previously

- **Split:** Boundary in $v$ becomes boundary in $\text{parent}(v)$
Removing Fixed x-coordinate Set Assumption

• **Split**: When \( v \) splits \( B \) new points needed in \( \text{parent}(v) \)

• One point obtained from \( v' \) (\( v'' \)) using “bubble-up” operation:
  – Find top point \( p \) in \( v' \)
  – Insert \( p \) in \( O(B^2) \)-structure
  – Remove \( p \) from \( O(B^2) \)-structure of \( v' \)
  – Recursively bubble-up point to \( v \)

• **Bubble-up** in \( O(\log_B w(v)) \) I/Os
  – Follow one path from \( v \) to leaf
  – Uses \( O(1) \) I/O in each node

\[ \text{Split in } O(B \log_B w(v)) = O(w(v)) \text{ I/Os} \]
Removing Fixed $x$-coordinate Set Assumption

- $O(1)$ amortized split cost:
  - Cost: $O(w(v))$
  - Weight balanced base tree: $\Omega(w(v))$ inserts below $v$ between splits

- External Priority Search Tree
  - Space: $O(N/B)$
  - Query: $O(\log_B N + T/B)$
  - Updates: $O(\log_B N)$ I/Os amortized

- Amortization can be removed from update bound in several ways
  - Utilizing lazy rebuilding
Summary: 3-sided Range Searching

- **3-sided range searching**
  - Maintain set of points in plane such that given query \((q_1, q_2, q_3)\), all points \((x, y)\) with \(q_1 \leq x \leq q_2\) and \(y \geq q_3\) can be found efficiently

- We obtained the same bounds as for the 1d case
  - Space: \(O(N/B)\)
  - Query: \(O(\log_B N + T/B)\)
  - Updates: \(O(\log_B N)\) I/Os
Summary: 3-sided Range Searching

- Main problem in designing external priority search tree was the increased fanout in combination with “overshooting”

- Same general solution techniques as in interval tree:
  - Bootstrapping:
    * Use $O(B^2)$ size structure in each internal node
    * Constructed using persistence
    * Dynamic using global rebuilding
  - Weight-balanced B-tree: Split/fuse in amortized $O(1)$
  - Filtering: Charge part of query cost to output
Two-Dimensional Range Search

- We have now discussed structures for special cases of two-dimensional range searching
  - Space: $O(N/B)$
  - Query: $O(\log_B N + T/B)$
  - Updates: $O(\log_B N)$

- Cannot be obtained for general 2d range searching:
  - $O(\log_B^c N)$ query requires $\Omega\left(\frac{N}{B} \log_B N \right)$ space
  - $O\left(\frac{N}{B}\right)$ space requires $\Omega\left(\sqrt{\frac{N}{B}}\right)$ query
External memory data structures

External Range Tree

- **Base tree:** Fan-out $\Theta(\log B N)$ weight balanced tree on $x$-coordinates
  \[ O\left(\frac{\log B N}{\log B \log B N}\right) \text{height} \]

- Points below each node stored in 4 linear space secondary structures:
  - “Right” priority search tree
  - “Left” priority search tree
  - B-tree on $y$-coordinates
  - Interval tree
  \[ \Omega\left(\frac{N}{B} \frac{\log B N}{\log B \log B N}\right) \text{space} \]

Lars Arge
External Range Tree

- Secondary interval tree structure:
  - Connect points in each slab in $y$-order
  - Project obtained segments in $y$-axis

- Intervals stored in interval tree
  * Interval augmented with pointer to corresponding points in $y$-coordinate B-tree in corresponding child node
• **Query** with \((q_1, q_2, q_3, q_4)\) answered in top node with \(q_1\) and \(q_2\) in different slabs \(v_1\) and \(v_2\)

• **Points in slab** \(v_1\)
  – Found with 3-sided query in \(v_1\) using right priority search tree

• **Points in slab** \(v_2\)
  – Found with 3-sided query in \(v_2\) using left priority search tree

• **Points in slabs between** \(v_1\) and \(v_2\)
  – Answer stabbing query with \(q_3\) using interval tree
    ⇒ first point above \(q_3\) in each of the \(O(\log B N)\) slabs
  – Find points using y-coordinate B-tree in \(O(\log B N)\) slabs

---

**External Range Tree**

\[ \Theta(\log B N) \]
External memory data structures

External Range Tree

• Query analysis:
  – \( O(\log_B N) \) I/Os to find relevant node
  – \( O(\log_B N + T/B) \) I/Os to answer two 3-sided queries
  – \( O(\log_B N + \log_B N/B) = O(\log_B N) \) I/Os to query interval tree
  – \( O(\log_B N + T/B) \) I/Os to traverse \( O(\log_B N) \) B-trees

\[ O(\log_B N + T/B) \] I/Os
External Range Tree

- **Insert:**
  - Insert $x$-coordinate in weight-balanced B-tree
    * Split of $v$ can be performed in $O(w(v) \log_B w(v))$ I/Os
    $\Rightarrow O\left(\frac{\log^2 N}{\log_B \log_B N}\right)$ I/Os
  - Update secondary structures in all $O\left(\frac{\log_B N}{\log_B \log_B N}\right)$ nodes on one root-leaf path
    * Update priority search trees
    * Update interval tree
    * Update B-tree
    $\Rightarrow O\left(\frac{\log^2 N}{\log_B \log_B N}\right)$ I/Os

- **Delete:**
  - Similar and using global rebuilding
Summary: External Range Tree

- **2d range searching** in $O\left(\frac{N}{B} \frac{\log_B N}{\log_B \log_B N}\right)$ space
  - $O(\log_B N + \frac{T}{B})$ I/O query
  - $O\left(\frac{\log_B N}{\log_B \log_B N}\right)$ I/O update

- **Optimal** among $O(\log_B N + \frac{T}{B})$ query structures
External memory data structures

**kdB-tree**

- **kd-tree:**
  - Recursive subdivision of point-set into two half using vertical/horizontal line
  - Horizontal line on even levels, vertical on uneven levels
  - One point in each leaf

\[\downarrow\]

Linear space and logarithmic height
External memory data structures

**kdB-tree**

- **Query:**
  - Recursively visit node corresponding to regions intersected query
  - Report point in trees/nodes completely contained in query
- **Analysis:**
  - Number of regions intersecting horizontal line satisfy recurrence
    \[
    Q(N) = 2 + 2Q\left(\frac{N}{4}\right) \Rightarrow Q(N) = O(\sqrt{N})
    \]
  - Query intersects \(4 \cdot O(\sqrt{N}) + T = O(\sqrt{N} + T)\) regions
**Lars Arge**

External memory data structures

---

**kB-tree**

- **KdB-tree:**
  - Blocking of kd-tree but with $B$ point in each leaf
- **Query** as before
  - Analysis as before except that each region now contains $B$ points
  \[
  O(\sqrt{\frac{N}{B}} + \frac{T}{B}) \text{ I/O query}
  \]
kdB-tree

- kdB-tree can be constructed in $O\left(\frac{N}{B} \log_B N\right)$ I/Os
  - somewhat complicated

↓

- Dynamic using logarithmic method:
  - $O\left(\sqrt{\frac{N}{B}} + \frac{T}{B}\right)$ I/O query
  - $O\left(\log_B^2 N\right)$ I/O update
  - $O(N/B)$ space
O-Tree Structure

- O-tree:
  - B-tree on $\Theta(\sqrt{\frac{N}{B}}/\log_B N)$ vertical slabs
  - B-tree on $\Theta(\frac{N}{B}/\log_B N)$ horizontal slabs in each vertical slab
  - kdB-tree on $\Theta(\sqrt{\frac{N}{B}/\log_B N})^2 = \Theta(B \log^2_B N)$ points in each leaf
O-Tree Query

- Perform rangesearch with $q_1$ and $q_2$ in vertical B-tree
  - Query all kB-trees in leaves of two horizontal B-trees with $x$-interval intersected but not spanned by query
  - Perform rangesearch with $q_3$ and $q_4$ horizontal B-trees with $x$-interval spanned by query
  * Query all kB-trees with range intersected by query

\[
\frac{\sqrt{N/B}}{\log_B N} \quad \frac{N/B}{\log_B^2 N} \quad B \log_B^2 N
\]
O-Tree Query Analysis

- **Vertical B-tree query:** $O(\log_B (\sqrt{N/B} / \log_B N)) = O(\sqrt{N/B})$
- **Query of all kB-trees in leaves of two horizontal B-trees:**
  
  
  $$O(\sqrt{N/B} / \log_B N) \cdot O(\sqrt{B \log_B^2 N/B + T_B}) = O(\sqrt{N/B} + T_B)$$

- **Query $O(\sqrt{N/B} / \log_B N)$ horizontal B-trees:**
  
  $$O(\sqrt{N/B} / \log_B N) \cdot O(\log_B (\sqrt{N/B} / \log_B N)) = O(\sqrt{N/B})$$

- **Query $2 \cdot O(\sqrt{N/B} / \log_B N)$ kB-trees not completely in query**
  
  $$2 \cdot O(\sqrt{N/B} / \log_B N) \cdot O(\sqrt{B \log_B^2 N/B + T_B}) = O(\sqrt{N/B} + T_B)$$

- **Query in kB-trees completely contained in query:** $O(T_B)$

  \[ \downarrow \]

  $$O(\sqrt{N/B} + T_B) \text{ I/Os}$$
O-Tree Update

- **Insert:**
  - Search in **vertical** B-tree: \(O(\log_B N)\) I/Os
  - Search in **horizontal** B-tree: \(O(\log_B N)\) I/Os
  - Insert in **kB-tree**: \(O(\log_B^2 (B \log_B^2 N)) = O(\log_B N)\) I/Os
- **Use global rebuilding** when structures grow too big/small
  - B-trees not contain \(\Theta(\sqrt{N/B}/\log_B N)\) elements
  - kB-trees not contain \(\Theta(B \log_B^2 N)\) elements
  \[ \downarrow \]
  \(O(\log_B N)\) I/Os

- **Deletes** can be handled in \(O(\log_B N)\) I/Os similarly
Summary: O-Tree

- **2d range searching** in linear space
  - $O(\sqrt{\frac{N}{B}} + \frac{T}{B})$ I/O query
  - $O(\log_B N)$ I/O update

- **Optimal** among structures using linear space

- Can be extended to work in $d$-dimensions with optimal query bound $O((\frac{N}{B})^{1-\frac{1}{d}} + \frac{T}{B})$
Summary: 3 and 4-sided Range Search

• 3-sided 2d range searching: External priority search tree
  – $O(\log_B N + \frac{T}{B})$ query, $O(\frac{N}{B})$ space, $O(\log_B N)$ update

• General (4-sided) 2d range searching:
  – External range tree: $O(\log_B N + \frac{T}{B})$ query, $\Omega(\frac{N}{B} \frac{\log_B N}{\log_B \log_B N})$ space, $O(\frac{\log^2_B N}{\log_B \log_B N})$ update
  – O-tree: $\Omega(\sqrt{\frac{N}{B}} + \frac{T}{B})$ query, $O(\frac{N}{B})$ space, $O(\log_B N)$ update
Techniques (one final time)

- **Tools:**
  - B-trees
  - Persistent B-trees
  - Buffer trees
  - Logarithmic method
  - Weight-balanced B-trees
  - Global rebuilding

- **Techniques:**
  - Bootstrapping
  - Filtering
Other results

- Many other results for e.g.
  - Higher dimensional range searching
  - Range counting
  - Halfspace (and other special cases) of range searching
  - Structures for moving objects
  - Proximity queries

- Many heuristic structures in database community

- Implementation efforts:
  - LEDA-SM (MPI)
  - TPIE (Duke)
THE END