Efficient Data Structures for Range-Aggregate Queries on Trees

Hao Yuan    Mikhail J. Atallah

Purdue University

March 23, 2009
1 Introduction

2 K-Depth Subtree Aggregation

3 K-Radius Subtree Aggregation

4 Conclusion and Future Work
Introduction

Range: Orthogonal or Proximity-Based
Aggregation Operators: SUM, MAX, MIN, MEDIAN, AVERAGE · · ·

- Geometric Aggregation
- OLAP Data Cube (Grid Graphs)
Introduction

Aggregations on Trees - Path Aggregation

- Applications
  - Bottleneck bandwidth of a path in a network
  - Verification of spanning trees [Tarjan 79, JACM] [Komlós 84, FOCS]
  - Aggregation along XML Path

- Process \( m \) online queries over a tree of size \( n \): \( \Theta(m\alpha(m,n)) \) time

  - Semigroup model: [Yao 82, STOC] [Chazelle 84, FOCS]
  - MAX or MIN operator: [Pettie 02, FOCS]
Our Contributions: Data Structures for Static Trees to support Online Queries. The operator is MIN (or MAX). Implementation in RAM model.
Applications: Web Site Hierarchy

- Before batch download those $k$-depth web pages rooted at $v$, want to know the total size of them (i.e., the SUM operator)
- What is the most visited web page? (MAX operator)
Applications: Computer Network

- Find a most powerful machine to install a database service to serve an application at $u$, but the chosen machine must be within $k$ hops of $u$ (i.e., response speed constraint) ($\text{MAX}$ operator)

- Or find the cheapest service provider ($\text{MIN}$ operator)
### Our Results

Worst-Case Complexities for SUM and MIN: \( O(\cdot) \) omitted

<table>
<thead>
<tr>
<th></th>
<th>Preprocess</th>
<th>Time and Space</th>
<th>Query Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Preprocessing</td>
<td>0</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Store all queries</td>
<td>( N^2 )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Our K-Depth Subtree</td>
<td>( N )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Our K-Radius Subtree</td>
<td>( N \log N )</td>
<td>( \log N )</td>
<td></td>
</tr>
</tbody>
</table>

Solutions for the SUM operator were given in [Yuan & Atallah 08, ICDCS] (see the revised version on the author’s web page). The present work focuses on the MIN operator.

**Difficulty in MIN**: No inverse exists to do the subtractions of prefix sums.
**K-Depth Subtree Aggregation**

Sketch of our solution:
- $O(N)$ comparisons
- Implementation in RAM

![Diagram showing KDS(u, k) and k hops in a tree structure]
Basic data structure: For each node $v$, compute its min-vector:

$$\{ \min \text{KDS}(v, k) \mid 0 \leq k \leq \text{Height}(v) \}$$

$O(N^2)$ space in the worst-case.
Naïve Bottom-Up Construction: Merge the min-vector from children. Need $\sum_{\nu} (1 + \text{Height}(\nu))$ comparisons. If the tree is balanced, then it’s $\Theta(n)$; but worst case is still $\Theta(n^2)$. 
Better Merging:

- Choose one of ν’s child with the largest Height, denoted by \( HC(ν) \)
- Use the min-vector of \( HC(ν) \) as a base min-vector of ν
- Merge the min-vector of other children into the base min-vector
Better Merging:
- Choose one of $v$’s child with the largest Height, denoted by $HC(v)$
- Use the min-vector of $HC(v)$ as a base min-vector of $v$
- Merge the min-vector of other children into the base min-vector

Total number of comparisons:

$$\sum_{v} \left( F(v) + \sum_{v_c \in \text{Children}(v) \setminus HC(v)} (1 + \text{Height}(v_c)) \right),$$

where $F(v)$ is amortized $O(1)$, and we proved that

$$\sum_{v} \sum_{v_c \in \text{Children}(v) \setminus HC(v)} \text{Height}(v_c) \leq N.$$
Convert a linear-comparison scheme to a linear RAM scheme:
Three Subqueries:

- **Top**: Reduced to do querying in a tree with \( \log N \) heights.

- **Middle**: Only \( O(N/ \log N) \) nodes need to take care of. \( O(T \log T) = O(N) \) data structures affordable with \( T = N/ \log N \).

- **Bottom**: Reduced to a special kind of 2D RMQ (Range Minimum Query).
Key Components:

- **Range Minimum Query**: Preprocess a given array to answer queries that ask for the minima of a contiguous subarray.

- **Level Descendant Query**: Find the leftmost (or rightmost) descendant of a node on a specific level.
  
  We showed that:
  
  Level Descendant Query \( \leq_{\text{LINEAR}} \) Level Ancestor Query
  
  Linear Time/Space Preprocessing and Constant Query Time
  
  [Berkman and Vishkin 94, JCSS]

- See also the unpublished note of [Amir Ben-Amram 07]
Top Subquery:

- **Top**: Reduced to do querying in a tree with $\log N$ heights — The min-vector of any node can be compressed into a single word!
KDS Aggregation - RAM Model

min-vector compression:

changed $\leftrightarrow 1$

same $\leftrightarrow 0$

min KDS( v, k )

[Diagram of tree aggregations]
min-vector compression:

\[ \text{changed} \leftrightarrow 1 \]

\[ \text{same} \leftrightarrow 0 \]
KDS Aggregation - RAM Model

Query Processing:

1. Find the level of the last changed

2. Locate level descendants

3. Do RMQ on that level

H. Yuan & M. J. Atallah (PURDUE)
KDS Aggregation - RAM Model

Query Processing:

1. Find the level of the last changed
2. Locate level descendants
3. Do RMQ on that level

compressed
min KDS( v, k )
K-RADIUS SUBTREE AGGREGATION

KRS(u, k)
k hops around u

\( O(n \log n) \) preprocessing time and space, \( O(\log n) \) query answering.
Heavy Path Decomposition [ Harel and Tarjan 84, SICOMP ]
Heavy Path Decomposition [ Harel and Tarjan 84, SICOMP ]
Heavy Path Decomposition [ Harel and Tarjan 84, SICOMP ]
Heavy Path Decomposition [ Harel and Tarjan 84, SICOMP ]
Recursively: big KRS $\rightarrow$ 2 KDS $\cup$ smaller KRS

Heavy Path Decomposition guarantees: 1 KRS $\rightarrow$ $2\log N$ KDS
**KRS Aggregation — Divide and Conquer**

Recursively: big KRS $\rightarrow$ 2 KDS $\cup$ smaller KRS

Heavy Path Decomposition guarantees: 1 KRS $\rightarrow$ $2 \log N$ KDS
Recursively: big KRS → 2 KDS ∪ smaller KRS

Heavy Path Decomposition guarantees: 1 KRS → 2\log N \text{ KDS}
Recursively: $\text{big KRS} \rightarrow 2 \text{ KDS } \cup \text{ smaller KRS}$

Heavy Path Decomposition guarantees: $1 \text{ KRS} \rightarrow 2 \log N \text{ KDS}$
Efficient data structures are designed to solve the proximity-based subtree aggregations

Future Work
- Dynamic Trees
- General Graphs (Fast approximation of the query result?)
- Cache Oblivious Data Structures
- Succinct Data Structures
Thanks!