A Linear-Time Algorithm for Optimal Tree Sibling Partitioning and Approximation Algorithms in Natix

VLDB 2006

Carl-Christian Kanne
University of Mannheim
kanne@db.informatik.uni-mannheim.de

Guido Moerkotte
University of Mannheim
moer@db.informatik.uni-mannheim.de
XML Document Tree Storage

Introduction

- XML Document Tree Storage
- Definition: Tree Sibling Partitioning

Flat Tree Partitioning
Deep Tree Partitioning
Approximation Algorithms
Evaluation
Conclusion

Carl-Christian Kanne, September 12, 2006

A Linear-Time Algorithm for Optimal Tree Sibling Partitioning... - p. 2/23
XML Document Tree Storage

Weight Limit: 5
XML Document Tree Storage

Tree Partitioning

Weight Limit: 5
Minimal Tree Partitioning

Weight Limit: 5
XML Document Tree Storage

Minimal Tree Partitioning

Single-edge partitions: 4

Weight Limit: 5
Minimal Tree Partitioning

Single-edge partitions: 4

Weight Limit: 5
Minimal Tree Partitioning

Single-edge partitions: 4
Sibling partitions: 3
Weight Limit: 5
Definition: Tree Sibling Partitioning

- $P$ set of sibling intervals $(l, r)$
  - 1 interval defines 1 partition
  - may cut more than one edge

- Single-edge partitioning: Intervals with $(l = r)$

- Weight of partition: Sum of node weights

- Feasible partitioning, given constant $K$
  All partition weights $\leq K$

- Optimal tree sibling partitioning $P$
  - Feasible
  - Minimal
  - (Lean)
Flat Tree Partitioning

Weight Limit $K = 5$:

```
   r:2
  /   \
 a:2   b:3
   |   /  \   \
 c:2 / \ d:3
  / \  / \
 e:1 f:2
```

Weight Limit $K = 5$:

```
   r:2
  /   \
 a:2   b:3
   |   /  \   \
 c:2 / \ d:3
  / \  / \
 e:1 f:2
```
Alternatives for last leaf $v$:
1. Put into partition with parent
2. Put into own partition $(v, v)$
3. Put into partition with up to $K - 1$ preceding siblings
Example

Weight Limit $K = 5$:

```
Weight Limit K = 5 :

r:2
a:2  b:3  c:2  d:3  e:1  f:2
```

- r:2
- a:2
- b:3
- c:2
- d:3
- e:1
- f:2
Parent Partition

Weight Limit $K = 5$:

![Weight Limit Diagram](image-url)
Own Partition

Weight Limit $K = 5$:

```
\begin{itemize}
  \item a:2
  \item b:3
  \item c:2
  \item d:3
  \item e:1
  \item f:2
\end{itemize}
```
Weight Limit $K = 5$:

```
With Left Sibling
```

```
K = 5:

r:2

a:2  b:3  c:2  d:3

| e:1   | e:2 |
```

```
With Left Sibling
```

```
Algorithm FDW
```

```
Deep Tree Partitioning
```

```
Approximation Algorithms
```

```
Evaluation
```

```
Conclusion
```

---

Carl-Christian Kanne, September 12, 2006

A Linear-Time Algorithm for Optimal Tree Sibling Partitioning... - p. 9/23
Algorithm FDW

- Flat trees, Dynamic programming, Width
- Dynamic programming table $D^s_j$, Optimal partitioning for $j$ leaves, root weight $s$
- Overall solution is $D^\text{weight(root)}_n$
- Each entry $D^s_j$ computed in $K + 1$ steps
  - Based on entries $D^{s'}_{j'}$ with $j < j'$ and $s' \geq s$
- Overall run-time: $O(nK^2)$
Deep Trees: Bottom-Up Processing

- Extend to deep trees
  - Process deep trees bottom-up
  - Process flat subtrees with FDW
  - Collapse processed trees into single node

Algorithm GHDW
(Greedy Height, Dynamic Width)
**Bottom-Up Processing with GHDW**

Weight Limit $K = 5$:

Collapse b using partitioning: $(e, e)$
Collapsing the Tree

Weight Limit $K = 5$:

Collapse $f$, $\{(e, e)\}$
Weight Limit $K = 5$:

Now use FDW on root

$\{(e, e)\}$
GHDW Result

Weight Limit $K = 5$:

Final result of GHDW Algorithm:
\[ \{(e, e), (b, b), (f, h), (a, a)\} \]
Problematic Case: GHDW

GHDW Result $K = 5$:
Problematic Case: GHDW

GHDW Result $K = 5$: 

```
GHDW Result K = 5:

K = 5:
a:5
b:1
c:1
d:2
e:2
f:1
```

```
Carl-Christian Kanne, September 12, 2006
```
Problematic Case: GHDW

GHDW Result $K = 5$:
4 Partitions

$K = 5$:
- $a$: 5
- $b$: 1
- $c$: 1
- $d$: 2
- $e$: 2
- $f$: 1
Problematic Case: GHDW

GHDW Result $K = 5$:
4 Partitions

Optimal Result $K = 5$:

```
K = 5:
4 Partitions
```

```
K = 5:
```

```
Problematic Case: GHDW
```
Problematic Case: GHDW

GHDW Result $K = 5$:
4 Partitions

Optimal Result $K = 5$:
3 Partitions

- $a:5$
- $b:1$
- $c:1$
- $d:2$
- $e:2$
- $f:1$

K = 5:

- 4 Partitions
- 3 Partitions
Optimal Substructure (Deep Trees)

- For each subtree, global optimum contains one of
  - locally optimal solution
  - locally nearly optimal solution (+1 interval)

- Algorithm DHW (Dynamic Height&Width)
  - integrate choice into dynamic programming
  - $O(nK^3)$
Alternative Algorithms

**GHDW** Always use optimal subtree partitionings

**DFS** Assign nodes to current partition in depth-first order
- new partition if full or not connected

**BFS** As above, but with breadth-first search

**KM** Kundu and Misra (1977)
- While subtree weight $> K$:
  - Cut edge of heaviest son
- Optimal for single-edge partitions!
Enhanced Kundu and Misra (EKM)

binary representation of n-ary tree
Enhanced Kundu and Misra (EKM)

Run Kundu and Misra Algorithm on binary representation of n-ary tree!
Enhanced Kundu and Misra (EKM)

Run Kundu and Misra Algorithm on binary representation of n-ary tree!
Enhanced Kundu and Misra (EKM)

Run Kundu and Misra Algorithm on binary representation of n-ary tree!
Enhanced Kundu and Misra (EKM)

Run Kundu and Misra Algorithm on binary representation of n-ary tree!
**Number of Partitions**

<table>
<thead>
<tr>
<th>Method</th>
<th>DBLP</th>
<th>partsupp.xml</th>
<th>XMark</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHW (Opt)</td>
<td>382</td>
<td>1083</td>
<td>8603</td>
</tr>
<tr>
<td>GHDW</td>
<td>384</td>
<td>1083</td>
<td>8838</td>
</tr>
<tr>
<td>EKM</td>
<td>402</td>
<td>1091</td>
<td>8975</td>
</tr>
<tr>
<td>KM</td>
<td>1294</td>
<td>15876</td>
<td>20519</td>
</tr>
<tr>
<td>DFS</td>
<td>1153</td>
<td>2282</td>
<td>25046</td>
</tr>
<tr>
<td>BFS</td>
<td>2987</td>
<td>8192</td>
<td>42155</td>
</tr>
</tbody>
</table>

\[ K = 256 \text{ slots} \text{ (1 slot = 8 bytes)} \]
## Partitioning Time

<table>
<thead>
<tr>
<th>Method</th>
<th>Elapsed CPU time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DBLP</td>
</tr>
<tr>
<td>DHW (Opt)</td>
<td>24.83</td>
</tr>
<tr>
<td>GHDW</td>
<td>0.28</td>
</tr>
<tr>
<td>EKM</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>KM</td>
<td>0.05</td>
</tr>
<tr>
<td>DFS</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>BFS</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Query</td>
<td>KM</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>Space</td>
<td>ca. 8192KB</td>
</tr>
<tr>
<td>Q1</td>
<td>0.065s</td>
</tr>
<tr>
<td>Q2</td>
<td>0.033s</td>
</tr>
<tr>
<td>Q3</td>
<td>0.770s</td>
</tr>
<tr>
<td>Q4</td>
<td>0.344s</td>
</tr>
<tr>
<td>Q5</td>
<td>0.150s</td>
</tr>
<tr>
<td>Q6</td>
<td>0.870s</td>
</tr>
<tr>
<td>Q7</td>
<td>0.854s</td>
</tr>
</tbody>
</table>

XPathMark on XMark scaling factor 0.1
Conclusion

- Allow siblings to share a partition
  - 50%-90% fewer partitions
  - Query performance $\times 2$
- Optimal dynamic programming algorithm $O(nK^3)$
- Very good approximation algorithm
  - run Kundu and Misra on binary tree (EKM)