LCS-TRIM: Dynamic Programming Meets XML Indexing and Querying

S. Tatikonda, S. Parthasarathy, M. Goyder

Presented by Wanxing Xu
Main Idea

- Convert XML documents (tree structure) to sequences (linear structure)
- Do the subsequence matching.
- Do the structure refinement
Approach

• Data representation
• Matching
  – Subsequence matching
  – Structure matching
• Indexing
• Optimizations
  – Labeling Filtering
  – Dominant Match Processing
Data Representation

• Convert the XML documents (tree structure) into sequences (linear structure)

• Main idea:
  – Numbering the nodes
    • Post-order
  – In some order, record the number and/or the label of the nodes
    • Post-order, Pre-order
Prüfer Sequence

• Constructed by two sequences:
  – Numbered Prüfer Sequence (NPS)
  – Label Sequence LS

• How to convert?
  – Number the nodes by post-order traversal.
  – Delete the node with the smallest number:
    • To NPS, append the number of its parent.
    • To LS, append the label of itself.
  – PRIX uses both the number and label of the parent of the deleted-node.
Example

- Post-order numbering
Construct the Sequences

Each entry in CPS is an edge.

Each entry in PRIIX is about the same node.

**CPS**

NPS: 2 9 4 7 6 7 8 9

LS: F B D B D C A E A

**NPS:** 2 9 4 7 6 7 8 9

**PRIIX**

NPS: 2 9 4 7 6 7 8 9

LPS: B A B A C A E A

Index: 1 2 3 4 5 6 7 8 9
Approach

- Data representation
- **Matching**
  - Subsequence matching
  - Structure matching
- Indexing
- Optimizations
  - Labeling Filtering
  - Dominant Match Processing
Main Idea

• Theorem 3.1 Consider a tree \( T \) and a twig query \( Q \) with their label sequences \( LS_T \) and \( LS_Q \), respectively. If \( Q \) is a subtree of \( T \), then \( LS_Q \) is a subsequence of \( LS_T \).

• Subtree \( \rightarrow \) Subsequence

• Subsequence \( \rightarrow \) Subtree ?

• NOT sufficient! More conditions needed!

• First find subsequence, then check more conditions and then find the subtrees.
Subsequence Matching

- LCS: Longest Common Subsequence
- Using Dynamic Programming to solve LCS
- Use a matrix $R$, $R[i,j]$ records the length of the LCS between $s_1[0..i]$ and $s_2[0..j]$.

$$R[i, j] = \begin{cases} 
0 & i = 0, j = 0 \\
R[i-1, j-1] + 1 & s_1[i] = s_2[j] \\
\max(R[i-1, j], R[i, j-1]) & s_1[i] \neq s_2[j]
\end{cases}$$
Example of LCS

\[ R[i, j] = \begin{cases} 
0 & i = 0, j = 0 \\
R[i - 1, j - 1] + 1 & s_1[i] = s_2[j] \\
\max(R[i - 1, j], R[i, j - 1]) & s_1[i] \neq s_2[j] 
\end{cases} \]

- Numbers in red are matches.
Subsequence Matching

• Property 3.1 If a label sequence $LS_Q$ is a subsequence of another label sequence $LS_T$, then $LS_Q$ is the longest common subsequence (LCS) of $LS_Q$ and $LS_T$.

• Each node in the query needs to match one in the document.

• The length of the LCS should be the same as the length of $LS_Q$. 
Subsequence Matching

• Two steps:
  – Construct the $R$ matrix, check the length of LCS (whether $L_{S_Q}$ is a subsequence of $L_{S_T}$)
  – Using backtrack to get all the matches

• Complexity
  – Time: $O(mn)$
  – Space: $O(mn)$
Example

Document

Query

R Matrix

Subsequence Matches:
M1(2, 3, 7)  M2(2, 5, 7)
M3(4, 5, 7)  M4(2, 3, 9)
M5(2, 5, 9)  M6(4, 5, 9)
**Definition:** 3.2. **Structure Agreement:** Consider two sequentures, derived from two trees $T_1$ and $T_2$, $S_1 = ((A_1, B_1) \ldots (A_m, B_m))$ and $S_2 = ((C_1, D_1) \ldots (C_m, D_m))$, where $A_i$’s and $C_i$’s define the structure; $B_i$’s and $D_i$’s provide the labels. Both $S_1$ and $S_2$ are said to agree on structure at position $i$ if and only if the following three conditions hold:

1. $1 \leq i \leq m$,
2. $B_i$ is equal to $D_i$,
3. If $A_i$ is the parent of $B_i$ in $T_1$ then $C_i$ is the parent of $D_i$ or the nearest ancestor of $C_i$ that is in $S_2$ must agree on structure with $S_1$ at position $A_i$.


Structure Agreement

- To check two nodes \((NPS_{Ti}, LS_{Ti})\) and \((NPS_{Qj}, LS_{Qj})\)
- \(NPS_{Ti}\) and \(NPS_{Qj}\) are their parents.
- Either the parents share the same label,
- or the NEAREST ancestor of \(NPS_{Qj}\) matches \(NPS_{Ti}\).
- (Apply some level-wise constraints for wildcards “*”\), etc).
Order of the Matching

- For each pair of nodes in the document and the query, we want to check whether their parents matches each other.
- In the CPS, we can see that child always appears before its parent.
- So, we need to match the nodes from the end of the sequence to the beginning.
Algorithm 2 Subtree matching

Input: CPS(Q), CPS(T), SM=(i₁, .., iₘ)
Output: mapping: positions at which Q matches to a subtree in T

1: mapping[m] ← iₘ
2: for k = m – 1 to 1 do
3:   p_q ← NPS_Q[k]
4:   p_t ← NPS_T[iₖ]
5:   if mapping[p_q] is equal to p_t or is an ancestor of p_t in T
6:      mapping[k] ← iₖ
7:     else
8:       Report that Q is not an embedded subtree of T
9:      Report that Q is an embedded subtree of T
For each pair of nodes...

- We have $P_q$: the parent of the node in Q
- $P_t$: the parent of the node in T
- mapping[$P_q$] the node in T that is already matched with $P_q$ in Q
- $P_t$ must be the same or the NEAREST ancestor of mapping[$P_q$]
- NEAREST: search each ancestor of $P_t$ bottom up, until the first already mapped node, it should be the same as mapping[$P_q$]
Example

Document

Query

R Matrix

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>B</th>
<th>D</th>
<th>B</th>
<th>D</th>
<th>C</th>
<th>A</th>
<th>E</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>B 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D 2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A 3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Subsequence Matches:
M1(2, 3, 7)    M2(2, 5, 7)
M3(4, 5, 7)    M4(2, 3, 9)
M5(2, 5, 9)    M6(4, 5, 9)
Q2 matches T5?
\[ P_q = 3 \]
\[ P_t = 6 \]
\[ mp[3] = 7 \neq 6 \]
In T, 7 is the parent of 6.
Match!

Q1 matches T4?
\[ P_q = 3 \]
\[ P_t = 7 \]
\[ mp[3] = 7 \]
Match!
Example

Document

Query

M2(2, 5, 7)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>mp</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Q2 matches T5?
P_q = 3
P_t = 6
mp[3] = 7 ≠ 6
In T, 7 is the parent of 6.
Match!

Q1 matches T2?
P_q = 3
P_t = 9
mp[3] = 7
not the parent of 9!
Nearest

- NEAREST: search each ancestor of $P_t$ bottom up, until the first already mapped node, it should be the same as $mp[P_q]$.
- Search for the ancestors one by one, we need $O(\text{depth of the tree})$, which is $O(n)$.
- The node scope representation DOES NOT work!
**Example**

**Document**

- Tree A: A (9) → B (2) → F (1) → D (3)
- Tree E: E (8) → B (1) → A (7) → C (6) → D (5)

**Query**

- Tree A: A (4) → B (1) → E (3)

**M (4, 6, 8, 9)**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>mp</td>
<td>X</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

**Q1 matches T4?**

- $P_q = 4$
- $P_t = 7$
- $mp[4] = 9$

From Pt, the first ancestor that already matched is E(8), which is not A(9)!
Approach

• Data representation
• Matching
  – Subsequence matching
  – Structure matching
• Indexing
• Optimizations
  – Labeling Filtering
  – Dominant Match Processing
Indexing

- For each label, collect the documents where it occurs.
- Only index infrequent labels (indexing frequent labels takes much space but not very helpful)
- $\alpha$-infrequent: appears in less then a fraction of $\alpha$ trees in the database.
- For a query, find the label which occurs in lest documents, only search among those documents.
Example

- Totally 10,000 documents
- $\alpha=50\%$
- A occurs in 6,000 documents, so not indexed.
- B occurs in 4,000 documents;
- C occurs in 3,000
- E occurs in 3,500
- Use the list of C to do the match.
Approach

• Data representation
• Matching
  – Subsequence matching
  – Structure matching
• Indexing
• Optimizations
  – Labeling Filtering
  – Dominant Match Processing
Label Filtering

- The dynamic programming asks for $O(mn)$ in both time and space.
- Eliminate the irrelevant labels (labels not in the query) from the document.

Query: BDA
Tree: FBDBDCAEA

Ignore the label F, C, E from the tree, because they are not in the query.

$O(3*9) \rightarrow O(3*6)$
Dominant Match

- Dominant match:
  - \( LS_T[i] = LS_Q[j] \)
  - \( R[i, j] = i \)

- Consider only dominant matches, ignore other cells.

Numbers in red are dominant matches.

Notice that \( R[3, 1] \) is only a match, but not dominant. It cannot appear in any result.
Put All Together

• For each query Q:
  – Using indexing to get a short list of candidate documents.
  – For each document T:
    • Using Label Filtering
    • Construct R matrix
    • Check the length of the LCS
    • Back track:
      – Find each dominant match
      – do the structure match at the same time

---

**Algorithm 3** The unified subtree matching algorithm

**Input**: A database tree T and a twig query Q

- `labelFilter ( T, Q ) \{T contains the filtered sequence\}
- `R ← computeLcsMatrix ( T, Q )`
- `if R[m, n] \neq m then`
  - Report that Q is not a subtree of T
- `SM ← null`
- `processLCS ( m, n, m )`

**Function**: `processLCS ( Qind, Tind, matchLen )`

1: if `matchLen = 0 then`
2: Report `SM` as the twig match
3: for `i = Tind` to `1` do
4: if `R[Qind][i]` is dominant & `R[Qind][Tind] = matchLen` then
5: if `isInAgreement(CPS(Q), SM, Qind)` then
6: `SM[Qind] ← CPS_T[Tind]`
7: `processLCS ( Qind-1, Tind-1, matchLen-1 )`
Early prune

- Subsequence matches:
  - M1(2, 3, 7)  M2(2, 5, 7)
  - M3(4, 5, 7)  M4(2, 3, 9)
  - M5(2, 5, 9)  M6(4, 5, 9)

- In the backtrack, say A9 is a match but D5 is not a match, we won’t continue to process B4 or B2. Prune M5 and M6 early! Instead, check D3.
Results

• With/without optimization
• Compare with PRIX
• Compare with TwigStack
With/without optimization

- No Opt
- Label Filtering & Dominant Match
- LCS-TRIM (back tract and structure match together)
Compare with PRIX

Figure 4: Performance comparison with PRIX on different data sets

Figure 5: Performance and Index size comparison on NLM data set
PRIX

• PRIX: subsequence matching + structure refinements (3 phases)
Why?

<table>
<thead>
<tr>
<th>Why?</th>
<th>PRIX uses B+tree, virtual trie and node scope to do the subsequence match. LCS-TRIM uses dynamic programming.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PRIX takes all the subsequences (false positive intermediate results) to do the structure refinements. LCS-TRIM prunes them very early.</td>
</tr>
</tbody>
</table>
Compare with TwigStack
Conclusion

• Novel sequence based representations
• Using dynamic programming of LCS
• Using inverted tree index
• Using several optimizations
• Prune out false candidate matches early
• Magnitude speedup over PRIX and TwigStack
Thank you!

Questions?