TASM: Top-k Approximate Subtree Matching

Nikolaus Augsten\textsuperscript{1} \hspace{1cm} Denilson Barbosa\textsuperscript{2}  
Michael Böhlen\textsuperscript{3} \hspace{1cm} Themis Palpanas\textsuperscript{4}  

\textsuperscript{1}Free University of Bozen-Bolzano, Italy  
augsten@inf.unibz.it  
\textsuperscript{2}University of Alberta, Canada  
denilson@cs.ualberta.ca  
\textsuperscript{3}University of Zurich, Switzerland  
boehlen@ifi.uzh.ch  
\textsuperscript{4}University of Trento, Italy  
themis@disi.unitn.eu  

ICDE 2010, March 3  
Long Beach, CA, USA
Outline

1 Motivation and Problem Definition

2 TASM-Postorder
   - Upper Bound on Subtree Size
   - Prefix Ring Buffer Pruning

3 Experiments

4 Conclusion and Future Work
1 Motivation and Problem Definition
2 TASM-Postorder
   - Upper Bound on Subtree Size
   - Prefix Ring Buffer Pruning
3 Experiments
4 Conclusion and Future Work
Motivation and Problem Definition

**Motivation**

**Query** (XML fragment)

```
article
  authors
    author
    author
  booktitle
    ICDE
```

**Document** (very large XML)

```
DBLP
  28M nodes, 531MB
```

Rank the *top-k matches* for the *article query* in the *DBLP document*!
Motivation and Problem Definition

**Query (XML fragment)**
```
article
  authors
    author
      Tim
  booktitle
    ICDE
author
author
ICDE
Tim
John
```

**Document (very large XML)**
```
DBLP
28M nodes, 531MB
```

**Motivation**

*Query* (XML fragment)

**Problem Definition**

Rank the top-$k$ matches for the *article* query in the DBLP document!

**Example Answer:** $k = 3$
```
inproceedings
  authors
    author
      Tim
  booktitle
author
author
ICDE
Tim
John
(1 error)
```
Motivation

Query (XML fragment)

```
article
   authors
      Tim
      John
   booktitle
      ICDE
```

Document (very large XML)

```
DBLP
28M nodes, 531MB
```

Rank the top-k matches for the article query in the DBLP document!

Example Answer: \( k = 3 \)

```
inproceedings
   authors
      Tim
      John
   booktitle
      ICDE

article
   authors
      Tim
      John
   booktitle
      TKDE
```
(1 error) (2 errors)
**Motivation and Problem Definition**

**Query (XML fragment)**

```
article
  authors
    author author ICDE
      Tim   John
  booktitle
```

**Document (very large XML)**

DBLP 28M nodes, 531MB

**top-k matches?**

**Rank the top-k matches for the article query in the DBLP document!**

Example Answer: \( k = 3 \)

1. **inproceedings**
   - authors
     - author author ICDE
       - Tim   John
     - booktitle
   - (1 error)

2. **article**
   - authors
     - author
       - Tim
   - booktitle
     - ICDE
   - (2 errors)

3. **inproceedings**
   - authors
     - author
       - Tim
   - booktitle
     - ICDE
   - (3 errors)
Definition (TASM: Top-k Approximate Subtree Matching)

**Given:** query tree $Q$, document tree $T$, size $k$ of ranking

**Goal:** Compute a

- top-k ranking $R = (T_1, T_2, \ldots, T_k)$
- of all subtrees $T_i$ of document $T$
- with respect to query $Q$
- using the tree edit distance for the ranking.
Definition (TASM: Top-k Approximate Subtree Matching)

Given: query tree $Q$, document tree $T$, size $k$ of ranking

Goal: Compute a

- top-k ranking $R = (T_1, T_2, \ldots, T_k)$
- of all subtrees $T_i$ of document $T$
- with respect to query $Q$
- using the tree edit distance for the ranking.

Subtree $T_i$:

- a node and all its descendants
- largest subtree is document itself
Definition (TASM: Top-k Approximate Subtree Matching)

Given: query tree $Q$, document tree $T$, size $k$ of ranking

Goal: Compute a

- **top-k ranking** $R = (T_1, T_2, \ldots, T_k)$
- of all **subtrees** $T_i$ of document $T$
- with respect to query $Q$
- using the tree edit distance for the ranking.

**Subtree $T_i$:**

- a node and all its descendants
- largest subtree is document itself

**top-k ranking** $R = (T_1, T_i, \ldots, T_k)$

- subtrees sorted by distance to query
- **best k** subtrees: $T_i \notin R \Rightarrow \text{ted}(Q, T_k) \leq \text{ted}(Q, T_i)$
Motivation and Problem Definition

Ranking Function: Tree Edit Distance (TED)

- **Tree Edit Distance**: Minimum number of node edit operations (insert, rename, delete) that transform one tree into the other.
Motivation and Problem Definition

Ranking Function: Tree Edit Distance (TED)

- **Tree Edit Distance**: Minimum number of node edit operations (insert, rename, delete) that transform one tree into the other.
Motivation and Problem Definition

Ranking Function: Tree Edit Distance (TED)

- **Tree Edit Distance**: Minimum number of node edit operations (insert, rename, delete) that transform one tree into the other.
**Motivation and Problem Definition**

**Ranking Function: Tree Edit Distance (TED)**

- **Tree Edit Distance**: Minimum number of node edit operations (*insert*, *rename*, *delete*) that transform one tree into the other.
- **TASM** computes TED between *query* and *document subtrees*.

```
article
   ├── authors
   │    ├── author
   │    │    └── Tim
   │    └── author
   │         └── John
   └── booktitle
       └── ICDE

article
   ├── author
   │    └── author
   │         └── John
   └── booktitle
       └── ICDE

article
   ├── author
   │    └── author
   │         └── Tim
   └── booktitle
       └── TKDE
```
Motivation and Problem Definition

Ranking Function: Tree Edit Distance (TED)

- **Tree Edit Distance**: Minimum number of node edit operations (insert, rename, delete) that transform one tree into the other.
- **TASM** computes TED between query and document subtrees
- **Size and number** of computed subtrees define TASM complexity
**Motivation and Problem Definition**

### State of the Art

- **TASM-Dynamic**: dynamic programming solution\(^1\)
  - computes distance to every subtree of the document
  - use smaller subtrees to compute larger ones
  - rank subtrees by visiting memoization table
- **Space complexity**: \(O(mn)\), \(m\): query size, \(n\): document size

\(^1\)Zhang and Shasha 1989, Demaine et al. 2007
State of the Art

- **TASM-Dynamic**: dynamic programming solution\(^1\)
  - computes distance to **every subtree** of the document
  - use smaller subtrees to compute larger ones
  - rank subtrees by visiting **memoization table**
  - **Space complexity**: \(O(mn)\), \(m\): query size, \(n\): document size

- **Space complexity limits** application to databases
  - in database applications \(n\) is huge (database size!)
  - TASM-Dynamic maintains two \(m \times n\) matrixes in RAM
  - > 6GB RAM for our tiny query \((m = 8)\) on DBLP \((n = 28 \times 10^6)\)

---

\(^1\)Zhang and Shasha 1989, Demaine et al. 2007
**TASM-Dynamic**: dynamic programming solution\(^1\)
- computes distance to every subtree of the document
- use smaller subtrees to compute larger ones
- rank subtrees by visiting memoization table
- **Space complexity**: \(O(mn)\), \(m\): query size, \(n\): document size

**Space complexity limits** application to databases
- in database applications \(n\) is huge (database size!)
- TASM-Dynamic maintains two \(m \times n\) matrixes in RAM
- \(> 6\text{GB}\) RAM for our tiny query \((m = 8)\) on DBLP \((n = 28 \times 10^6)\)

For database size solutions dynamic programming is too expensive.

\(^1\)Zhang and Shasha 1989, Demaine et al. 2007
State of the Art

- **TASM-Dynamic**: dynamic programming solution\(^1\)
  - computes distance to **every subtree** of the document
  - use smaller subtrees to compute larger ones
  - rank subtrees by visiting **memoization table**
  - **Space complexity**: \(O(mn)\), \(m\): query size, \(n\): document size

- **Space complexity limits** application to databases
  - in database applications \(n\) is huge (database size!)
  - TASM-Dynamic maintains two \(m \times n\) matrixes in RAM
  - \(> 6GB\) RAM for our tiny query \((m = 8)\) on DBLP \((n = 28 \times 10^6)\)

For database size solutions **dynamic programming** is too expensive.

State-of-the-art algorithms do not scale!

\(^1\) Zhang and Shasha 1989, Demaine et al. 2007
Find a solution for **TASM** (Top-k Approximate Subtree Matching) that
- scales to **very large documents**
- runs in **small memory**
- ranks subtrees **correctly** (no heuristics!)
Outline

1. Motivation and Problem Definition

2. TASM-Postorder
   - Upper Bound on Subtree Size
   - Prefix Ring Buffer Pruning

3. Experiments

4. Conclusion and Future Work
Outline

1. Motivation and Problem Definition

2. TASM-Postorder
   - Upper Bound on Subtree Size
   - Prefix Ring Buffer Pruning

3. Experiments

4. Conclusion and Future Work
Subtree Size Upper Bound in Three Steps

1. **Rank first** \( k \) **subtrees** of \( T \) in postorder: \( R' = (T'_1, T'_2, \ldots, T'_k) \)
Subtree Size Upper Bound in Three Steps

1. Rank first $k$ subtrees of $T$ in postorder: $R' = (T'_1, T'_2, \ldots, T'_k)$

   $\text{(i) } t_{ed}(Q, T'_k) \leq |Q| + |T'_k|$
Subtree Size Upper Bound in Three Steps

1. **Rank first** $k$ **subtrees** of $T$ in postorder: $R' = (T'_1, T'_2, \ldots, T'_k)$

   \[(i) \quad \text{ted}(Q, T'_k) \leq |Q| + |T'_k|\]

2. **Final ranking** $R = (T_1, T_2, \ldots, T_k)$ (≡TASM result)
Subtree Size Upper Bound in Three Steps

1. **Rank first** $k$ **subtrees** of $T$ in postorder: $R' = (T'_1, T'_2, \ldots, T'_k)$

   \[ \text{ted}(Q, T'_k) \leq |Q| + |T'_k| \]

2. **Final ranking** $R = (T_1, T_2, \ldots, T_k)$ (**TASM result**)

   $T_i$’s in $R$ are better than worst match $T'_k$ of $R'$

   \[ \text{ted}(Q, T_i) \leq \text{ted}(Q, T'_k) \]
Subtree Size Upper Bound in Three Steps

1. **Rank first** $k$ **subtrees** of $T$ in postorder: $R' = (T'_1, T'_2, \ldots, T'_k)$

   \[(i) \quad ted(Q, T'_k) \leq |Q| + |T'_k|\]

2. **Final ranking** $R = (T_1, T_2, \ldots, T_k)$ (＝TASM result)

   $T_i$’s in $R$ are better than worst match $T'_k$ of $R'$

   \[(ii) \quad ted(Q, T_i) \leq ted(Q, T'_k) \leq |Q| + |T'_k|\]
Subtree Size Upper Bound in Three Steps

1. **Rank first** $k$ **subtrees** of $T$ in postorder: $R' = (T'_1, T'_2, \ldots, T'_k)$

   \[
   (i) \text{ted}(Q, T'_k) \leq |Q| + |T'_k|
   \]

2. **Final ranking** $R = (T_1, T_2, \ldots, T_k)$ (\(=\)TASM result)

   $T_i$’s in $R$ are better than worst match $T'_k$ of $R'$

   \[
   (ii) \text{ted}(Q, T_i) \leq \text{ted}(Q, T'_k) \leq |Q| + |T'_k|
   \]

3. **Size upper bound** for subtree $T_i$

   \[
   |T_i| - |Q| \leq \text{ted}(Q, T_i)
   \]
Subtree Size Upper Bound in Three Steps

1. **Rank first** \( k \) **subtrees** of \( T \) in postorder: \( R' = (T'_1, T'_2, \ldots, T'_k) \)

(i) \( \text{ted}(Q, T'_k) \leq |Q| + |T'_k| \)

2. **Final ranking** \( R = (T_1, T_2, \ldots, T_k) \) (=TASM result)

\( T_i \)'s in \( R \) are better than worst match \( T'_k \) of \( R' \)

(ii) \( \text{ted}(Q, T_i) \leq \text{ted}(Q, T'_k) \leq |Q| + |T'_k| \)

3. **Size upper bound** for subtree \( T_i \)

\( |T_i| - |Q| \leq \text{ted}(Q, T_i) \)

\( |T_i| \leq \text{ted}(Q, T_i) + |Q| \)
Subtree Size Upper Bound in Three Steps

1. Rank first $k$ subtrees of $T$ in postorder: $R' = (T'_1, T'_2, \ldots, T'_k)$

   (i) $ted(Q, T'_k) \leq |Q| + |T'_k|$

2. Final ranking $R = (T_1, T_2, \ldots, T_k)$ (=TASM result)

   $T_i$’s in $R$ are better than worst match $T'_k$ of $R'$

   (ii) $ted(Q, T_i) \leq ted(Q, T'_k) \leq |Q| + |T'_k|$

3. Size upper bound for subtree $T_i$

   $|T_i| - |Q| \leq ted(Q, T_i)$

   $|T_i| \leq ted(Q, T_i) + |Q| \leq 2|Q| + |T'_k|$
Subtree Size Upper Bound in Three Steps

1. **Rank first** \( k \) **subtrees** of \( T \) in postorder: \( R' = (T'_1, T'_2, \ldots, T'_k) \)

   (i) \( \text{ted}(Q, T'_k) \leq |Q| + |T'_k| \)

2. **Final ranking** \( R = (T_1, T_2, \ldots, T_k) \) (=TASM result)

   \( T_i \)’s in \( R \) are better than worst match \( T'_k \) of \( R' \)

   (ii) \( \text{ted}(Q, T_i) \leq \text{ted}(Q, T'_k) \leq |Q| + |T'_k| \)

3. **Size upper bound** for subtree \( T_i \)

   \[ |T_i| - |Q| \leq \text{ted}(Q, T_i) \]

   \[ |T_i| \leq \text{ted}(Q, T_i) + |Q| \leq 2|Q| + |T'_k| \]
Subtree Size Upper Bound in Three Steps

1. Rank first $k$ subtrees of $T$ in postorder: $R' = (T'_1, T'_2, \ldots, T'_k)$
   
   \begin{align*}
   (i) \quad & ted(Q, T'_k) \leq |Q| + |T'_k| \\
   \end{align*}

2. Final ranking $R = (T_1, T_2, \ldots, T_k)$ (=TASM result)
   
   $T_i$'s in $R$ are better than worst match $T'_k$ of $R'$
   
   \begin{align*}
   (ii) \quad & ted(Q, T_i) \leq ted(Q, T'_k) \leq |Q| + |T'_k| \\
   \end{align*}

3. Size upper bound for subtree $T_i$
   
   \begin{align*}
   |T_i| - |Q| & \leq ted(Q, T_i) \\
   \end{align*}

   \begin{align*}
   |T_i| & \leq ted(Q, T_i) + |Q| \leq 2|Q| + |T'_k| \leq 2|Q| + k \\
   \end{align*}
TASM needs to consider only small document subtrees of size $\tau$ or less:

$$\tau = 2|Q| + k$$
Upper Bound on Subtree Size

**Theorem (Upper Bound on Subtree Size)**

*TASM needs to consider only small document subtrees of size $\tau$ or less:*

$$\tau = 2|Q| + k$$

Upper bound is very powerful:

- **Independent of document** size and structure!
- **Linear in query** size and $k$
Theorem (Upper Bound on Subtree Size)

TASM needs to consider only small document subtrees of size $\tau$ or less:

$$\tau = 2|Q| + k$$

Upper bound is very powerful:

- independent of document size and structure!
- linear in query size and $k$

Example: top-10 with example query $|Q| = 8$ on DBLP (28M nodes)
- with bound: max subtree size $\tau = 2 \times 8 + 10 = 26$
- without bound: maximum subtree size is 28M (whole document)!
**Theorem (Upper Bound on Subtree Size)**

*TASM needs to consider only small document subtrees of size $\tau$ or less:*

$$\tau = 2|Q| + k$$

Upper bound is very powerful:

- *independent of document* size and structure!
- *linear in query* size and $k$

**Example:** top-10 with example query $|Q| = 8$ on DBLP (28M nodes)

- *with* bound: max subtree size $\tau = 2 \times 8 + 10 = 26$
- *without* bound: maximum subtree size is *28M* (whole document)!

*Document-independent upper bound on subtree size!*
Outline

1. Motivation and Problem Definition

2. TASM-Postorder
   - Upper Bound on Subtree Size
   - Prefix Ring Buffer Pruning

3. Experiments

4. Conclusion and Future Work
Document Format: Postorder Queue

- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., (John,1)
  - no random access!
**Document Format: Postorder Queue**

- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!
Document Format: Postorder Queue

- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!
**Postorder queue**: queue of (label,size)-pairs
- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
Document Format: Postorder Queue

- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!
**Postorder queue**: queue of (label,size)-pairs
- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
**Postorder queue**: queue of (label, size)-pairs

- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
**Document Format: Postorder Queue**

- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!
**Document Format: Postorder Queue**

- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!
**Postorder queue**: queue of (label,size)-pairs

- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
**Postorder queue**: queue of (label,size)-pairs

- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
**Postorder queue**: queue of (label, size)-pairs
- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
**Postorder queue**: queue of (label,size)-pairs
- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
**Document Format: Postorder Queue**

- **Postorder queue**: queue of (label, size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!
**Document Format: Postorder Queue**

- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!
**Postorder queue**: queue of (label, size)-pairs

- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
**Document Format: Postorder Queue**

- **Postorder queue**: queue of (label,size)-pairs
  - dequeue removes leftmost element, e.g., (John, 1)
  - no random access!
**Postorder queue**: queue of (label,size)-pairs

- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
**Document Format: Postorder Queue**

- **Postorder queue**: queue of (label,size)-pairs
  - dequeue removes leftmost element, e.g., (John, 1)
  - no random access!
**Document Format: Postorder Queue**

- **Postorder queue**: queue of (label, size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!

```
[John,1] [auth,2] [X1,1] [title,2] [article,5]
[VLDB,1] [conf,2] [Peter,1] [auth,2] [X3,1]
[title,2] [article,5] [Mike,1] [auth,2] [X4,1]
[title,2] [article,5] [proc,13] [X2,1]
```
**Postorder queue**: queue of (label,size)-pairs
- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
Document Format: Postorder Queue

- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!

```
John, 1   auth, 2   X1, 1   title, 2   article, 5

VLDB, 1   conf, 2   Peter, 1   auth, 2   X3, 1

title, 2   article, 5   Mike, 1   auth, 2   X4, 1

title, 2   article, 5   proc, 13   X2, 1   title, 2

book, 3
```
**Postorder queue**: queue of (label, size)-pairs

- **dequeue** removes leftmost element, e.g., (John, 1)
- no random access!
Postorder queue: queue of (label,size)-pairs
- dequeue removes leftmost element, e.g., (John, 1)
- no random access!

Relevant and state-of-the-art for XML Parsing
- full subtree known only at closing tag
- closing tags appear in postorder
**Document Format: Postorder Queue**

- **Postorder queue**: queue of (label,size)-pairs
  - **dequeue** removes leftmost element, e.g., (John, 1)
  - no random access!

- **Relevant and state-of-the-art for XML Parsing**
  - full subtree known only at closing tag
  - closing tags appear in postorder

- **Implementation** is efficient and heavily used for
  - XML streams
  - plain XML files (e.g., SAX)
  - XML in database (Dewey, interval encoding, ...)

---

**Prefix Ring Buffer Pruning**

**Postorder Queue**

```
John,1 → auth,2 → X1,1 → title,2 → article,5 →
  → VLDB,1 → conf,2 → Peter,1 → auth,2 → X3,1 →
  → title,2 → article,5 → Mike,1 → auth,2 → X4,1 →
  → title,2 → article,5 → proc,13 → X2,1 → title,2 →
  → book,3 → dblp,22 →
```

- `posting` field: label of posting
- `size` field: size of posting
- `root` field: root of posting
Candidate Subtrees

- **Candidate subtrees** are all subtrees $T_i$ of the document with
  - $|T_i| \leq \tau$ AND
  - $T_i$ is not contained in a larger subtree $|T_j| \leq \tau$
Candidate Subtrees

- **Candidate subtrees** are all subtrees $T_i$ of the document with
  - $|T_i| \leq \tau$ AND
  - $T_i$ is not contained in a larger subtree $|T_j| \leq \tau$

- **Pruning**: find candidate subtrees
Simple pruning approach: ($\tau = 6$ in example above)
- add nodes to memory buffer until non-candidate ($|T_i| > \tau$) is added
- subtrees of non-candidate with $|T_i| \leq \tau$ are candidate subtrees
Simple Pruning Approach

- **Simple pruning** approach: ($\tau = 6$ in example above)
  - add nodes to memory buffer until non-candidate ($|T_i| > \tau$) is added
  - subtrees of non-candidate with $|T_i| \leq \tau$ are candidate subtrees
Simple pruning approach: ($\tau = 6$ in example above)
- add nodes to memory buffer until non-candidate ($|T_i| > \tau$) is added
- subtrees of non-candidate with $|T_i| \leq \tau$ are candidate subtrees
Simple pruning approach: ($\tau = 6$ in example above)

- add nodes to memory buffer until non-candidate ($|T_i| > \tau$) is added
- subtrees of non-candidate with $|T_i| \leq \tau$ are candidate subtrees
Simple pruning approach: \((\tau = 6\ \text{in example above})\)

- add nodes to memory buffer until non-candidate \((|T_i| > \tau)\) is added
- subtrees of non-candidate with \(|T_i| \leq \tau\) are candidate subtrees
Simple Pruning Approach

- **Simple pruning** approach: \( \tau = 6 \) in example above
  - add nodes to memory buffer until non-candidate \(|T_i| > \tau\) is added
  - subtrees of non-candidate with \(|T_i| \leq \tau\) are candidate subtrees

- **Problem**: memory buffer can grow very large!
  - must keep subtrees in memory until non-candidate ancestor is read
  - worst case: memory buffer stores \( O(n) \) nodes
    (frequent in data-centric XML!)
Simple Pruning Approach

- **Simple pruning** approach: \((\tau = 6 \text{ in example above})\)
  - add nodes to memory buffer until non-candidate \((|T_i| > \tau)\) is added
  - subtrees of non-candidate with \(|T_i| \leq \tau\) are candidate subtrees
- **Problem**: memory buffer can grow very large!
  - must keep subtrees in memory until non-candidate ancestor is read
  - worst case: memory buffer stores \(O(n)\) nodes
    (frequent in data-centric XML!)
- **Example**: DBLP, \(\tau = 50\)
  - 99% of nodes are still in buffer when root node is read!

\[
\text{article}_5 \quad \text{proceedings}_{18} \quad \text{book}_{21}
\]
\[
\text{auth}_2 \quad \text{title}_4 \quad \text{conf}_7 \quad \text{article}_{12} \quad \text{article}_{17} \quad \text{title}_{20}
\]
\[
\text{John}_1 \quad \text{X}_13 \quad \text{VLDB}_6 \quad \text{auth}_9 \quad \text{title}_{11} \quad \text{auth}_{14} \quad \text{title}_{16} \quad \text{X}_219
\]
\[
\text{Peter}_8 \quad \text{X}_310 \quad \text{Mike}_{13} \quad \text{X}_415
\]
Simple Pruning Approach

Simple pruning approach: \((\tau = 6 \text{ in example above})\)
- add nodes to memory buffer until non-candidate \((|T_i| > \tau)\) is added
- subtrees of non-candidate with \(|T_i| \leq \tau\) are candidate subtrees

Problem: memory buffer can grow very large!
- must keep subtrees in memory until non-candidate ancestor is read
- worst case: memory buffer stores \(O(n)\) nodes
  (frequent in data-centric XML!)

Example: DBLP, \(\tau = 50\)
- 99% of nodes are still in buffer when root node is read!

Simple pruning not feasible for large documents!
Efficient Pruning is Tricky!

- **Problem:** when can we remove a node from the buffer?
  - when we see $|T_i| \leq \tau$, we don’t yet know about parent (postorder!)
  - subtree of parent might be smaller than $\tau$!
Efficient Pruning is Tricky!

- **Problem:** when can we remove a node from the buffer?
  - when we see $|T_i| \leq \tau$, we don’t yet know about parent (postorder!)
  - subtree of parent might be smaller than $\tau$!

- **Our Solution** does not wait for parent
  - prefix ring buffer: fixed size buffer
  - pruning rule: prune based on following nodes
Prefix ring buffer (\(\tau = 6\))

<table>
<thead>
<tr>
<th>e</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>John,1</td>
<td>auth,2</td>
</tr>
</tbody>
</table>

Prefix ring buffer of size \(\tau + 1\) (main memory)
- stores prefix (\(\tau\) nodes in postorder) of the document
Prefix ring buffer \((\tau = 6)\)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>John,1</th>
<th>auth,2</th>
<th>X1,1</th>
<th>title,4</th>
<th>article,5</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Prefix ring buffer of size \(\tau + 1\) (main memory)

- **stores prefix** \((\tau\) nodes in postorder) of the document
- **two operations**
Pruning in Small Memory

prefix ring buffer ($\tau = 6$)

<table>
<thead>
<tr>
<th>e↑</th>
<th>John,1</th>
<th>auth,2</th>
<th>X1,1</th>
<th>title,4</th>
<th>article,5</th>
</tr>
</thead>
</table>

Prefix ring buffer of size $\tau + 1$ (main memory)

- stores prefix ($\tau$ nodes in postorder) of the document
- two operations
  - append new node
Prefix ring buffer ($\tau = 6$)

<table>
<thead>
<tr>
<th>VLDB,1</th>
<th>John,1</th>
<th>auth,2</th>
<th>X1,1</th>
<th>title,4</th>
<th>article,5</th>
</tr>
</thead>
<tbody>
<tr>
<td>e↑</td>
<td>s↑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Prefix ring buffer of size $\tau + 1$ (main memory)

- stores prefix ($\tau$ nodes in postorder) of the document
- two operations
  - append new node
prefix ring buffer \((\tau = 6)\)

<table>
<thead>
<tr>
<th>VLDB,1</th>
<th>John,1</th>
<th>auth,2</th>
<th>X1,1</th>
<th>title,4</th>
<th>article,5</th>
</tr>
</thead>
<tbody>
<tr>
<td>e↑</td>
<td>s↑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Prefix ring buffer of size \(\tau + 1\) (main memory)

- stores prefix \((\tau\) nodes in postorder) of the document
- two operations
  - append new node
  - remove leftmost subtree/node
**Prefix ring buffer** ($\tau = 6$)

<table>
<thead>
<tr>
<th>VLDB, 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

$s \uparrow$  $e \uparrow$

**Prefix ring buffer** of size $\tau + 1$ (main memory)

- stores prefix ($\tau$ nodes in postorder) of the document
- two operations
  - append new node
  - remove leftmost subtree/node
Pruning in Small Memory

prefix ring buffer \((\tau = 6)\)

Prefix ring buffer of size \(\tau + 1\) (main memory)

- stores prefix \((\tau\) nodes in postorder) of the document
- two operations
  - append new node
  - remove leftmost subtree/node

Pruning rule: If leftmost node in full ring buffer is

- leaf: leftmost subtree is candidate subtree
- non-leaf: leftmost node is non-candidate node
Pruning Rule – Intuition

- **Candidate subtree**: leftmost node is a leaf
  - $T_i$: leftmost subtree, starts with leftmost node
  - $T_j$: smallest subtree that contains $T_i$
  - due to postorder: $T_j$ contains all nodes in buffer
  - since $|T_i| \leq \tau$ and $|T_j| > \tau$: $T_i$ is a candidate
Pruning Rule – Intuition

**Candidate subtree**: leftmost node is a leaf

- $T_i$: leftmost subtree, starts with leftmost node
- $T_j$: smallest subtree that contains $T_i$
- due to postorder: $T_j$ contains all nodes in buffer
- since $|T_i| \leq \tau$ and $|T_j| > \tau$: $T_i$ is a candidate

**Non-candidate node**: leftmost node is a non-leaf

- leftmost non-leaf is parent of previously removed nodes
- we remove either candidate subtrees and non-candidate nodes
- in both cases: parent is a non-candidate
Prefix Ring Buffer Pruning – Example

1. fill ring buffer

2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove

3. until queue and buffer empty

\[ \tau = 6 \]

append

postorder queue (input)

John,1 \rightarrow auth,2 \rightarrow X1,1 \rightarrow \ldots

prefix ring buffer (main memory)

\[ \text{candidate subtrees: (output)} \]

Nikolaus Augsten (Bolzano, Italy)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

postorder queue (input)

prefix ring buffer (main memory)

candidate subtrees: (output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

τ = 6

postorder queue (input)

prefix ring buffer (main memory)

candidate subtrees: (output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer

2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove

3. until queue and buffer empty

\( \tau = 6 \)

postorder queue (input)

| title,2 | article,5 | VLDB,1 | … |

prefix ring buffer (main memory)

| John,1 | auth,2 | X1,1 | | | | |

candidate subtrees:
(output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>John,1</th>
<th>auth,2</th>
<th>X1,1</th>
<th>title,2</th>
<th>...</th>
</tr>
</thead>
</table>

candidate subtrees:
(output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>John,1</th>
<th>auth,2</th>
<th>X1,1</th>
<th>title,2</th>
<th>article,5</th>
</tr>
</thead>
</table>

candidate subtrees: (output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

append

<table>
<thead>
<tr>
<th>postorder queue (input)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conf,2</td>
</tr>
<tr>
<td>Peter,1</td>
</tr>
<tr>
<td>auth,2</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>John,1</th>
<th>auth,2</th>
<th>X1,1</th>
<th>title,2</th>
<th>article,5</th>
<th>VLDB,1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>e</td>
</tr>
</tbody>
</table>

candidate subtrees:
(output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\( \tau = 6 \)

postorder queue (input)

prefix ring buffer (main memory)

candidate subtrees:
(output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

τ = 6

prefix ring buffer (main memory)

candidate subtrees: article
  \ auth title
  |    |    John X1

postorder queue (input)

article auth title conf
  \ auth title auth title X2
  |    |    Peter X3 Mike X4

τ = 6
1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

append

postorder queue (input)

prefix ring buffer (main memory)

candidate subtrees: (output)

article
   / \ auth title
   /   |
John X1
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

append

postorder queue (input)

prefix ring buffer (main memory)

candidate subtrees: (output)

article

auth title

John X1
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\( \tau = 6 \)

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>Peter,1</th>
<th>auth,2</th>
<th></th>
<th></th>
<th></th>
<th>VLDB,1</th>
<th>conf,2</th>
</tr>
</thead>
<tbody>
<tr>
<td>e↑</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s↑</td>
<td></td>
</tr>
</tbody>
</table>

candidate subtrees: (output)

\[
\text{article} \\
\text{auth title} \\
\text{John X1}
\]
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

append

postorder queue (input)
```
| title,2 | article,5 | Mike,1 | ...
```

prefix ring buffer (main memory)
```
| Peter,1 | auth,2 | X3,1 | | | VLDB,1 | conf,2 |
```

candidate subtrees: (output)
```
article
  \(\backslash\)
  auth title
    \(\backslash\)
    John X1
```
Prefix Ring Buffer Pruning – Example

1. fill ring buffer

2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove

3. until queue and buffer empty

\[ \tau = 6 \]

append

postorder queue (input)

article,5 → Mike,1 → auth,2 → …

candidate subtrees: (output)

article
  / \ auth title
  |   |
John X1
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

append

postorder queue (input)

article,5 \rightarrow Mike,1 \rightarrow auth,2 \rightarrow …

candidate subtrees: (output)

article
  \ /
auth title
  |   |
John X1

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>Peter,1</th>
<th>auth,2</th>
<th>X3,1</th>
<th>title,2</th>
<th>VLDB,1</th>
<th>conf,2</th>
</tr>
</thead>
</table>

\[ e \uparrow \quad s \uparrow \]
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

τ = 6

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>Peter,1</th>
<th>auth,2</th>
<th>X3,1</th>
<th>title,2</th>
<th></th>
</tr>
</thead>
</table>

candidate subtrees: (output)

```
      article
      |  \\
auth title  
|       \\
      John  X1
```

postorder queue (input)

```
article,5  Mike,1  auth,2  ...
```
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

postorder queue (input)

prefix ring buffer (main memory)

candidate subtrees: (output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

append

postorder queue (input)

\[
\begin{array}{c}
\text{auth,2} \rightarrow \text{X4,1} \rightarrow \text{title,2} \rightarrow \ldots
\end{array}
\]

prefix ring buffer (main memory)

\[
\begin{array}{cccccc}
\text{Peter,1} & \text{auth,2} & \text{X3,1} & \text{title,2} & \text{article,5} & \text{Mike,1}
\end{array}
\]

candidate subtrees: (output)

\[
\begin{array}{l}
\text{article} \quad \text{conf} \\
\text{auth title} \quad \text{VLDB} \\
\text{John} \quad \text{X1}
\end{array}
\]
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\( \tau = 6 \)

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>Peter,1</th>
<th>auth,2</th>
<th>X3,1</th>
<th>title,2</th>
<th>article,5</th>
<th>Mike,1</th>
</tr>
</thead>
</table>

candidate subtrees: (output)

article
  \( \overleftarrow{\text{conf}} \)
  \( \overleftarrow{\text{auth title}} \)
  \( \overleftarrow{\text{VLDB}} \)
  \( \overleftarrow{\text{John X1}} \)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

postorder queue (input)

prefix ring buffer (main memory)

candidate subtrees: (output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

prefix ring buffer (main memory)

candidate subtrees: (output)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

𝜏 = 6

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>e↑</th>
<th>s↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>X4,1</td>
<td>Mike,1</td>
</tr>
</tbody>
</table>

candidate subtrees: (output)

article / \ conf / \ article
auth title / \ \\
John X1 / \ VLDB / \ auth title
\ \\
\ | Peter X3
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

append

postorder queue (input)

article,5 \rightarrow proc,13 \rightarrow X2,1 \rightarrow \ldots

candidate subtrees: (output)

\[
\begin{array}{c}
\text{article} \\
\text{auth title} \\
\text{John X1}
\end{array}
\quad
\begin{array}{c}
\text{conf} \\
\text{VLDB}
\end{array}
\quad
\begin{array}{c}
\text{article} \\
\text{auth title} \\
\text{Peter X3}
\end{array}
\]
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

postorder queue (input):

```
proc,13 → X2,1 → title,2 → ...
```

prefix ring buffer (main memory):

```
<table>
<thead>
<tr>
<th>X4,1</th>
<th>title,2</th>
<th>article,5</th>
</tr>
</thead>
</table>
```

candidate subtrees: (output)

```
article
  / \ conf
auth title / \ article
  |     / \ 
John X1 VLDB auth title
  |     / 
Peter X3 Mike X4
```
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

append

postorder queue (input)

X2,1 \( \rightarrow \) title,2 \( \rightarrow \) book,3 \( \rightarrow \) ...

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>X4,1</th>
<th>title,2</th>
<th>article,5</th>
<th>proc,13</th>
<th>Mike,1</th>
<th>auth,2</th>
</tr>
</thead>
</table>

candidate subtrees: (output)

article

/ \conf

/ \article

/ \auth title

/ \VLDB

/ \auth title

/ \Mike

/ \Peter

/ \X3

John X1

Nikolaus Augsten (Bolzano, Italy)

TASM: Top-k Approx. Subtree Matching

ICDE 2010 20 / 28
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

append
candidate subtrees:
(output)

postorder queue (input)

prefix ring buffer (main memory)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

Prefix ring buffer (main memory)

Candidate subtrees: (output)

article
  \[ \text{auth title} \]
  John X1

conf
  VLDB

article
  \[ \text{auth title} \]
  Peter X3

article
  \[ \text{auth title} \]
  Mike X4

postorder queue (input)

article
  auth title
  John X1

conf
  VLDB

article
  \[ \text{auth title} \]
  Peter X3

article
  \[ \text{auth title} \]
  Mike X4
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

postorder queue (input)

\[ \text{title,2} \rightarrow \text{book,3} \rightarrow \text{dblp,22} \rightarrow \ldots \]

prefix ring buffer (main memory)

\[
\begin{array}{cccc}
\text{proc,13} & \text{X2,1} & \text{X1} & \text{X3} & \text{X4} \\
\text{s} & \uparrow & & & \text{e} \uparrow
\end{array}
\]

candidate subtrees: (output)

\[
\begin{array}{cccc}
\text{article} & \text{conf} & \text{article} & \text{article} \\
\text{auth title} & \text{VLDB} & \text{auth title} & \text{auth title} \\
\text{John} & \text{X1} & \text{Peter} & \text{X3} & \text{Mike} & \text{X4}
\end{array}
\]
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

$\tau = 6$

prefix ring buffer (main memory)

candidate subtrees: (output)

article
  / \ conf
auth title VLDB
  |   |
John X1 Peter X3
  |
Mike X4
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

$\tau = 6$

prefix ring buffer (main memory)

| e↑ | proc,13 | X2,1 | title,2 | book,3 | s↑ |

candidate subtrees: (output)

article
\/
\/
auth title
\/
John
\/
X1
\
conf
\/

article
\/
\/
auth title
\/
VLDB
\/
Peter
\/
X3
\
article
\/
\/
auth title
\/
Mike
\/
X4
\
article
\/
\/

Prefix Ring Buffer Pruning – Example

1. fill ring buffer

2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove

3. until queue and buffer empty

$\tau = 6$

append

postorder queue (input)

(empty) $\rightarrow$ $\rightarrow$ $\rightarrow$ $\cdots$

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>dblp,22</th>
<th></th>
<th>proc,13</th>
<th>X2,1</th>
<th>title,2</th>
<th>book,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>e↑</td>
<td></td>
<td>s↑</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

candidate subtrees: (output)

article
   \ /
  auth title
  |  |  
John X1  VLDB

conf

article
   \ /
  auth title
  |  |  
Peter X3  Mike X4

article
   \ /
  auth title
  |  |  
article
   \ /
  auth title
  |  |  
article
   \ /
  auth title
  |  |  
article
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\( \tau = 6 \)
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

**postorder queue (input)**

```
(EMPTY) -> [ ] -> [ ] -> ...
```

**prefix ring buffer (main memory)**

<table>
<thead>
<tr>
<th>dblp,22</th>
<th></th>
<th>X2,1</th>
<th>title,2</th>
<th>book,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>e↑</td>
<td></td>
<td>s↑</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**candidate subtrees: (output)**

```
article  / \  conf  / \  article  / \  article
  / \      / \      / \      / \      / \      / \\
auth title auth title auth title auth title
  / \      / \      / \      / \\
John X1  VLDB  auth title Mike X4
```
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>dblp,22</th>
<th></th>
<th></th>
<th>X2,1</th>
<th>title,2</th>
<th>book,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>e↑</td>
<td></td>
<td></td>
<td>s↑</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

candidate subtrees: (output)

article  
/ \  
auth title conf
| |  
| |  
John X1 VLDB

article  
/ \  
auth title article
| |  
| |  
Peter X3 Mike X4
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

prefix ring buffer (main memory)

<table>
<thead>
<tr>
<th>dblp,22</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

\[ \text{s} \uparrow \quad \text{e} \uparrow \]

candidate subtrees: (output)

<table>
<thead>
<tr>
<th>article</th>
<th>conf</th>
<th>article</th>
<th>article</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth title</td>
<td>VLDB auth title</td>
<td>auth title</td>
<td>auth title</td>
<td>title</td>
</tr>
<tr>
<td>John X1</td>
<td></td>
<td>Peter X3</td>
<td>Mike X4</td>
<td>X2</td>
</tr>
</tbody>
</table>
Prefix Ring Buffer Pruning – Example

1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

candidate subtrees: (output)

article
   \[ \text{auth title} \]
   John X1

conf
   \[ \text{article} \]
   VLDB X1

article
   \[ \text{auth title} \]
   Peter X3

article
   \[ \text{auth title} \]
   Mike X4

title
   X2

\[
\begin{align*}
\text{prefix ring buffer (main memory)} & : \\
\text{postorder queue (input):} & \\
\tau = 6 & \\
\text{append} & \\
\text{(empty)} & \rightarrow \\
& \rightarrow \\
& \rightarrow \\
& \rightarrow \\
& \rightarrow \\
\end{align*}
\]
1. fill ring buffer
2. check leftmost node
   - leaf: candidate subtree – to result
   - non-leaf: non-candidate – remove
3. until queue and buffer empty

\[ \tau = 6 \]

prefix ring buffer (main memory)

candidate subtrees: (output)
TASM-postorder

1. empty ranking $R$, tightening upper bound $\tau' = \tau$

2. for each candidate subtree $T_i$
   a. if $|R| = k$: update $\tau' = \min(\tau, \max(R) + |Q|)$
   b. compute tree edit distance for all subtrees of $T_i$ within $\tau'$
   c. update ranking $R$
TASM-Postorder

TASM-postorder

1. empty ranking $R$, tightening upper bound $\tau' = \tau$
2. for each candidate subtree $T_i$
   a. if $|R| = k$: update $\tau' = \min(\tau, \max(R) + |Q|)$
   b. compute tree edit distance for all subtrees of $T_i$ within $\tau'$
   c. update ranking $R$

Theorem (TASM-Postorder)

The space complexity of TASM-postorder is independent of the document size:

$$O(m^2 + mk)$$

($m$: query size, $k$: result size)
TASM-Postorder

TASM-postorder

1. empty ranking \( R \), tightening upper bound \( \tau' = \tau \)
2. for each candidate subtree \( T_i \)
   a. if \( |R| = k \): update \( \tau' = \min(\tau, \max(R) + |Q|) \)
   b. compute tree edit distance for all subtrees of \( T_i \) within \( \tau' \)
   c. update ranking \( R \)

Theorem (TASM-Postorder)

The space complexity of TASM-postorder is independent of the document size:

\[ O(m^2 + mk) \]

\( (m: \text{query size}, k: \text{result size}) \)

TASM-postorder scales to very large documents!
Outline

1 Motivation and Problem Definition

2 TASM-Postorder
   - Upper Bound on Subtree Size
   - Prefix Ring Buffer Pruning

3 Experiments

4 Conclusion and Future Work
Pruning Effectiveness
Prefix ring buffer pruning is very effective!
Maximum subtree reduced from 37M to 18 nodes.

- **Dataset**: PSD protein sequences, 37M nodes, 683MB
- **Compute TASM** (\(|Q| = 4, k = 1\))
  - TASM-dynamic (state of the art)
  - TASM-postorder (our solution)
- **Histogram** of computed subtrees

![Histogram of computed subtrees](image)

- **TASM-Dynamic**
- **TASM-Postorder**

- Largest subtree: 37M entire document
- Largest subtree: 18
Experiments

Scalability: TASM-Postorder vs. TASM-Dynamic
Scalability: TASM-Postorder vs. TASM-Dynamic

TASM-postorder much faster than TASM-dynamic.

- **Dataset:** XMark (synthetic XML for benchmark)
- Vary query size and document size
- Compute TASM ($k = 5$)
  - TASM-dynamic (state of the art)
  - TASM-postorder (our solution)
- Measure wall clock **time**

![Graphs showing time vs. query size and document size for TASM-dynamic and TASM-postorder]
Scalability with Result Size $k$
**Experiments**

### Scalability with Result Size $k$

**TASM-postorder** scales well with $k$. Increasing $k$ by 4 orders of magnitude only doubles runtime.

- **Dataset:** XMark (synthetic XML for benchmark)
- **Vary $k$ (size of ranking)**
- **Compute TASM ($|Q|=16$)**
  - TASM-dynamic (state of the art)
  - TASM-postorder (our solution)
- **Measure wall clock time**
Space complexity: TASM-Postorder vs. TASM-Dynamic
Experiments

Space complexity: TASM-Postorder vs. TASM-Dynamic

**TASM-postorder:** space independent of document!

- **Dataset:** XMark (synthetic XML for benchmark)
- **Vary document size**
- **Compute TASM** \( k = 5 \)
  - TASM-dynamic (state of the art)
  - TASM-postorder (our solution)
- **Measure main memory usage**
Outline

1. Motivation and Problem Definition

2. TASM-Postorder
   - Upper Bound on Subtree Size
   - Prefix Ring Buffer Pruning

3. Experiments

4. Conclusion and Future Work
Conclusion

- **Prefix Ring Buffer** for space efficient pruning
- **Dynamic programming does not scale** for database size solutions.
- **Upper bound** $\tau$: limit maximum subtree size for TASM
- **TASM-postorder**: highly scalable TASM algorithm

TASM-postorder makes TASM feasible.
Conclusion

- **Prefix Ring Buffer** for space efficient pruning
- **Dynamic programming does not scale** for database size solutions.
- **Upper bound \( \tau \):** limit maximum subtree size for TASM
- **TASM-postorder:** highly scalable TASM algorithm

**TASM-postorder makes TASM feasible.**

**Future Work** – New research opportunities:
- tune tree edit distance to different applications
- index the document: can we avoid a document scan?
- parallel TASM algorithm: where to split document?
Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann.

An optimal decomposition algorithm for tree edit distance.

K. Zhang and D. Shasha.
Simple fast algorithms for the editing distance between trees and related problems.