Approximate Joins for Data-Centric XML

Nikolaus Augsten\textsuperscript{1} \quad Michael Böhlen\textsuperscript{1} \quad Curtis Dyreson\textsuperscript{2} \quad Johann Gamper\textsuperscript{1}

\textsuperscript{1}Free University of Bozen-Bolzano
Bolzano, Italy
\{augsten,boehlen,gamper\}@inf.unibz.it

\textsuperscript{2}Utah State University
Logan, UT, U.S.A.
curtis.dyreson@usu.edu

April 10, 2008
ICDE, Cancún, Mexico
Outline

1 Motivation

2 Windowed $pq$-Grams for Data-Centric XML
   - Windowed $pq$-Grams
   - Tree Sorting
   - Forming Bases

3 Efficient Approximate Joins with Windowed $pq$-Gram

4 Experiments

5 Related Work

6 Conclusion and Future Work
Motivation

Approximate Join on Music CDs

Song Lyric Store

- album
  - track
    - title artist artist title
      - So Far Mark Roger Breathe
  - track
    - title
      - Harvest

CD Warehouse

- album
  - track
    - price
      - artist title
        - Neil Alabama
  - track
    - price
      - artist title
        - Roger Breathe
        - Mark So Far

Query: Give me all album pairs that represent the same music CDs.
Query: Give me all album pairs that represent the same music CDs.

How similar are two XML items?
**Motivation**

How Similar Are these XMLs?

- **Standard solution** $O(n^3)$: tree edit distance
  - Minimum number of **node edit operations** (insert, delete, rename) that transforms one ordered tree into the other.
**Motivation**

How Similar Are these XMLs?

- **Standard solution** $O(n^3)$: tree edit distance
  Minimum number of **node edit operations** (insert, delete, rename) that transforms one ordered tree into the other.

- **Problem**: permuted subtrees are deleted/re-inserted node by node
Ordered Trees

sibling order matters

\[
\begin{align*}
\text{Ordered Trees} & \quad \begin{array}{c}
\text{c} \\
\text{d} \\
\text{e}
\end{array} \\
\text{a} & \quad \begin{array}{c}
\text{b}
\end{array}
\end{align*}
\]

\[
\begin{align*}
\text{a} & \quad \begin{array}{c}
\text{b} \\
\text{d} \\
\text{e}
\end{array}
\end{align*}
\]
Ordered vs. Unordered Trees

**Ordered Trees**
- sibling order matters

**Unordered Trees**
= data-centric XML
- sibling order ignored

**Notes:**
- Ignore order when comparing unordered trees.
- Order matters in ordered trees, as shown by the different structures.
- The example uses a simple tree structure to illustrate the concept.

*Source:* Nikolaus Augsten (Bolzano, Italy), Approximate Joins for Data-Centric XML, ICDE 2008 – Cancún, Mexico
Ordered vs. Unordered Trees

Ordered Trees
- sibling order matters

Unordered Trees
- \(\text{data-centric XML}\)
  - sibling order ignored

**Edit distance** between unordered trees: \(\text{NP-complete}\)
- all sibling permutations must be considered!
Problem Definition

Find an **effective distance** for the approximate matching of hierarchical data represented as **unordered labeled trees** that is **efficient** for approximate joins.
Find an **effective distance** for the approximate matching of hierarchical data represented as **unordered labeled trees** that is **efficient** for approximate joins.

**Naive approaches** that fail:
- unordered tree edit distance: NP-complete
- allow subtree move: NP-hard
- compute minimum distance between all permutations: $O(n!)$
- sort by label and use ordered tree edit distance: error $O(n)$
Outline

1 Motivation

2 Windowed $pq$-Grams for Data-Centric XML
   • Windowed $pq$-Grams
   • Tree Sorting
   • Forming Bases

3 Efficient Approximate Joins with Windowed $pq$-Gram

4 Experiments

5 Related Work

6 Conclusion and Future Work
Our Solution: Windowed $pq$-Grams

- **Windowed $pq$-Gram**: small subtree with stem and base

\[
\text{stem} \quad p = 2 \\
\text{base} \quad q = 3
\]
Our Solution: Windowed $pq$-Grams

- **Windowed pq-Gram**: small subtree with stem and base

- **Key Idea**: split unordered tree into set of windowed $pq$-grams that is
  - not sensitive to the sibling order
  - sensitive to any other change in the tree

- **Intuition**: similar unordered trees have similar windowed $pq$-grams
Our Solution: Windowed $pq$-Grams

- **Windowed $pq$-Gram**: small subtree with *stem* and *base*

- **Key Idea**: split unordered tree into set of windowed $pq$-grams that is
  - not sensitive to the sibling order
  - sensitive to any other change in the tree

- **Intuition**: similar unordered trees have similar windowed $pq$-grams

- **Systematic computation** of windowed $pq$-grams
  1. sort the children of each node by their label (works OK for $pq$-grams)
  2. simulate permutations with a **window**
  3. split tree into windowed $pq$-grams
Implementation of Windowed $pq$-Grams

- **Set of windowed $pq$-grams:**

  $\begin{align*}
  a & \quad * & * & * & * & * & a & a & a & a & a & a & c & c \\
  b & c & a & a & a & a & b & c & c & c & c & c & d & e \\
  d & e & b & c & b & c & b & c & b & c & b & c & d & e \\
  \end{align*}$
Implementation of Windowed \(pq\)-Grams

- **Set of windowed \(pq\)-grams:**
  \[
  \begin{array}{c}
  a \\
  b \\ c \\
  d \\ e \\
  \end{array}
  \quad \rightarrow \quad
  \begin{array}{cccccccccccccccc}
  * & * & * & * & * & a & a & a & a & a & a & a & c & c \\
  a & a & a & a & a & b & c & c & c & c & c & c & d & e \\
  b & c & b^* & c^* & c^* & b^* & b^* & c^* & c^* & d & e & d^* & e & d^* & e & d^* & e & e & e & * & * & * & * & * \\
  \end{array}
  \\

- **Hashing:** map \(pq\)-gram to integer:

  \[
  \begin{array}{c}
  * \\
  a \\
  b \\ c \\
  \end{array}
  \quad \rightarrow \quad
  (*, a, b, c) \quad (\text{shorthand}) \quad
  \begin{array}{c}
  \text{serialize} \\
  \rightarrow \quad
  *abc \quad \rightarrow \quad
  \text{hash} \quad \rightarrow \quad
  0973
  \end{array}
  \\

  \begin{array}{ccc}
  \text{label} & l & h(l) \\
  * & 0 & \\
  a & 9 & \\
  b & 7 & \\
  c & 3 & \\
  \ldots & \ldots & \\
  \end{array}
  \\

  \textbf{Note:} labels may be strings of arbitrary length!
Implementation of Windowed \( pq \)-Grams

- **Set of windowed \( pq \)-grams:**

  \[
  \begin{array}{c}
  a \\
  b \\
  c \\
  d \\
  e \\
  \end{array}
  \rightarrow
  \begin{array}{c}
  * \\
  a \\
  b \\
  c \\
  d \\
  e \\
  \end{array}
  \]

- **Hashing:** map \( pq \)-gram to integer:

<table>
<thead>
<tr>
<th>label</th>
<th>( h(l) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>9</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

  \[
  \text{serialize: } (*, a, b, c) \rightarrow *abc \quad \text{(shorthand)} \quad \text{hash: } \rightarrow 0973
  \]

  \textbf{Note:} labels may be strings of arbitrary length!

- **\( pq \)-Gram index:** bag of hashed \( pq \)-grams

  \[
  I(T) = \{0973, 0970, 0930, 0937, 0907, 0903, 9700, 9316, 9310, 9360, 9361, 9301, 9306, 3100, 3600\}
  \]

  \textbf{Tree is represented by a bag of integers!}
The **windowed pq-gram distance** between two trees, $T$ and $T'$:

$$\text{dist}^{pq}(T, T') = |\mathcal{I}(T) \cup \mathcal{I}(T')| - 2|\mathcal{I}(T) \cap \mathcal{I}(T')|$$
The windowed $pq$-gram distance between two trees, $T$ and $T'$:

$$\text{dist}^{pq}(T, T') = |\mathcal{I}(T) \cup \mathcal{I}(T')| - 2|\mathcal{I}(T) \cap \mathcal{I}(T')|$$

- **Pseudo-metric** properties hold:
  - **self-identity**: $x = y \Rightarrow \text{dist}^{pq}(x, y) = 0$
The windowed $pq$-gram distance between two trees, $T$ and $T'$:

$$\text{dist}^{pq}(T, T') = |I(T) \cup I(T')| - 2|I(T) \cap I(T')|$$

- **Pseudo-metric** properties hold:
  - self-identity: $x = y \iff \text{dist}^{pq}(x, y) = 0$

  ![Venn diagram](image)

- Different trees may be at distance zero:

  ![Tree diagram](image)
The \textbf{windowed $pq$-gram distance} between two trees, $T$ and $T'$:

$$\text{dist}^{pq}(T, T') = |\mathcal{I}(T) \cup \mathcal{I}(T')| - 2|\mathcal{I}(T) \cap \mathcal{I}(T')|$$

\textbf{Pseudo-metric} properties hold:

✓ **self-identity**: $x = y \iff \text{dist}^{pq}(x, y) = 0$

✓ **symmetry**: $\text{dist}^{pq}(x, y) = \text{dist}^{pq}(y, x)$

Different trees may be at distance zero:

```
   b
  / \
 b   b
 / \ / \b
b   b b
```

\text{I}(T) \cap \text{I}(T')
The windowed $pq$-gram distance between two trees, $T$ and $T'$:

$$\text{dist}^{pq}(T, T') = |I(T) \cup I(T')| - 2|I(T) \cap I(T')|$$

- **Pseudo-metric** properties hold:
  - ✓ self-identity: $x = y \iff \text{dist}^{pq}(x, y) = 0$
  - ✓ symmetry: $\text{dist}^{pq}(x, y) = \text{dist}^{pq}(y, x)$
  - ✓ triangle inequality: $\text{dist}^{pq}(x, z) \leq \text{dist}^{pq}(x, y) + \text{dist}^{pq}(y, z)$

- Different trees may be at distance zero:

\[
\begin{array}{c}
\node{b} \\
\node{b} \\
\node{b} \\
\node{b}
\end{array}
\quad
\begin{array}{c}
\node{b} \\
\node{b} \\
\node{b} \\
\node{b}
\end{array}
\]
The windowed \( pq \)-gram distance between two trees, \( T \) and \( T' \):

\[
\text{dist}^{pq}(T, T') = |\mathcal{I}(T) \cup \mathcal{I}(T')| - 2|\mathcal{I}(T) \cap \mathcal{I}(T')|
\]

- **Pseudo-metric** properties hold:
  - ✓ self-identity: \( x = y \Leftrightarrow \text{dist}^{pq}(x, y) = 0 \)
  - ✓ symmetry: \( \text{dist}^{pq}(x, y) = \text{dist}^{pq}(y, x) \)
  - ✓ triangle inequality: \( \text{dist}^{pq}(x, z) \leq \text{dist}^{pq}(x, y) + \text{dist}^{pq}(y, z) \)

- Different trees may be at distance zero:

\[
\begin{array}{c}
  b \quad b \\
  | \quad | \\
  b \quad b \\
  \quad | \\
  b \quad b
\end{array}
\]

- **Runtime** for the distance computation is \( O(n \log n) \).
Outline

1 Motivation

2 Windowed \(pq\)-Grams for Data-Centric XML
   - Windowed \(pq\)-Grams
   - Tree Sorting
   - Forming Bases

3 Efficient Approximate Joins with Windowed \(pq\)-Gram

4 Experiments

5 Related Work

6 Conclusion and Future Work
Idea:

1. sort the children of each node by their label
2. apply an ordered tree distance
**Idea:**

1. sort the children of each node by their label
2. apply an ordered tree distance

![Diagram](image.png)

- **Edit distance:** tree sorting does not work
- **Windowed pq-Grams:** tree sorting works OK
Edit Distance: Tree Sorting Does Not Work

1. Non-unique sorting:

\[ \begin{align*}
\text{unordered} & \quad \text{edit dist} = 0 \\
\end{align*} \]
Edit Distance: Tree Sorting Does Not Work

1. Non-unique sorting:
**Edit Distance: Tree Sorting Does Not Work**

1. **Non-unique sorting**: edit distance $O(n)$ for identical trees

![Diagram showing non-unique sorting](image)

**Example:***

- Unordered: $\text{edit dist} = 0$
- Ordered: $\text{edit dist} = O(n)$
2. Node renaming:

\( T_2 \)

\( a \)

\( e \)

\( f \)

\( g \)

\( d \)

\( j \)

\( k \)

\( c \)

\( b \)

\( h \)

\( i \)

\( 1 \) rename

\( T_2 \)

\( a \)

\( b \)

\( e \)

\( f \)

\( g \)

\( d \)

\( j \)

\( k \)

\( c \)

\( b \)

\( h \)

\( i \)
2. Node renaming:

\[
\begin{align*}
T_2 &= \begin{array}{c}
\text{a} \\
\text{b} \\
\text{c} \\
\text{d} \\
\text{e} \\
\text{f} \\
\text{g} \\
\text{h} \\
\text{i} \\
\text{k} \\
\text{j}
\end{array}
\end{align*}
\]

\[
\begin{align*}
T_2 &= \begin{array}{c}
\text{a} \\
\text{b} \\
\text{c} \\
\text{d} \\
\text{e} \\
\text{f} \\
\text{g} \\
\text{h} \\
\text{i} \\
\text{k} \\
\text{j}
\end{array}
\end{align*}
\]

\[
\begin{align*}
1 \text{ rename} \\
\text{sort} \quad \rightarrow \quad \text{sort}
\end{align*}
\]

\[
\begin{align*}
\text{dist} = 1
\end{align*}
\]
2. **Node renaming:**

\[ T_2 \]

- \( a \)
  - \( e \)
  - \( f \)
  - \( g \)
  - \( d \)
  - \( j \)
  - \( k \)
- \( b \)
  - \( h \)
  - \( i \)

\[ 1 \text{ rename} \]

- \( a \)
  - \( e \)
  - \( f \)
  - \( g \)
  - \( d \)
  - \( j \)
  - \( k \)
- \( b \)
  - \( h \)
  - \( i \)

\[ \text{sort} \]

- \( a \)
  - \( b \)
  - \( c \)
  - \( d \)
  - \( e \)
  - \( f \)
  - \( g \)
  - \( h \)
  - \( i \)
  - \( j \)
  - \( k \)

\[ \text{dist} = 1 \]

\[ \text{sort} \]

- \( a \)
  - \( b \)
  - \( c \)
  - \( d \)
  - \( e \)
  - \( f \)
  - \( g \)
  - \( h \)
  - \( i \)
  - \( j \)
  - \( k \)

\[ \text{sort} \]

- \( a \)
  - \( b \)
  - \( c \)
  - \( d \)
  - \( e \)
  - \( f \)
  - \( g \)
  - \( h \)
  - \( i \)
  - \( j \)
  - \( k \)

\[ \text{sort} \]

- \( a \)
  - \( b \)
  - \( c \)
  - \( d \)
  - \( e \)
  - \( f \)
  - \( g \)
  - \( h \)
  - \( i \)
  - \( j \)
  - \( x \)
2. **Node renaming**: edit distance depends on node label

$$T_2$$

```
      a
  f
/   \
|     |
|     |
|     |
```

$$T_2$$

```
      b
  f
/   \
|     |
|     |
|     |
```

$$T_2$$

```
      x
  e
/   \
|     |
|     |
|     |
```

- Sort
- $$\text{dist} = 1$$
- $$\text{sort}$$
- $$\text{dist} = O(n)$$
Theorem (Local Effect of Node Reordering)

If $k$ children of a node are reordered, i.e., their subtrees are moved, only $O(k)$ windowed $pq$-grams change.

Proof (idea):

- $pq$-grams consist of a **stem** and a **base**
- **stems** are invariant to the sibling order
- **bases**: only the $O(k)$ $pq$-grams with the reordered nodes in the bases change
Theorem (Local Effect of Node Reordering)

If $k$ children of a node are reordered, i.e., their subtrees are moved, only $O(k)$ windowed $pq$-grams change.

Proof (idea):
- $pq$-grams consist of a stem and a base.
- Stems are invariant to the sibling order.
- Bases: only the $O(k)$ $pq$-grams with the reordered nodes in the bases change.

✓ Non-unique sortings are equivalent: distance is 0 for identical trees.
✓ Node renaming is independent of the node label.
Outline

1. Motivation

2. Windowed $pq$-Grams for Data-Centric XML
   - Windowed $pq$-Grams
   - Tree Sorting
   - Forming Bases

3. Efficient Approximate Joins with Windowed $pq$-Gram

4. Experiments

5. Related Work

6. Conclusion and Future Work
How To Form Bases?

- **Goal** for windowed $pq$-grams:
  - not sensitive to the sibling order
  - sensitive to any other change in the tree

```
  stem
  p = 2

  base
  q = 3
```
How To Form Bases?

**Goal** for windowed $pq$-grams:
- **not sensitive** to the sibling order
- **sensitive** to any other change in the tree

**Stems**: ignore sibling order

```
  a
/ \                          * a a c c
 b  c  →  | | | | | |
/ \   a b c d e
 d  e
```

Diagram:
```
  stem
    p = 2
  base
    q = 3
```
How To Form Bases?

**Goal** for windowed $pq$-grams:

- **not sensitive** to the sibling order
- **sensitive** to any other change in the tree

**Stems:** ignore sibling order

```
  a
 /\  
b  c  * a a c c
 /\    1 1 1 1 1
 d  e  a b c d e
```

**Bases:** do not ignore sibling order!
Requirements for Bases

- **Requirements** for bases:
  - detection of node moves
  - robustness to different sortings
  - balanced node weight
Requirements for Bases

**Requirements** for bases:
- detection of node moves
- robustness to different sortings
- balanced node weight

**Our solution:**
- **windows**: simulate all permutations within a window
- **wrapping**: wrap windows that extend beyond the right border
- **dummies**: extend small sibling sets with dummy nodes
Solution: Windowed $pq$-Gram Bases

Algorithm 1: **Form bases** from a sorted sibling sequence

- **Example:** stem, sorted sibling sequence, window $w = 3$

```
    a
   /|
  b / |
 /  |
/   |
/    d e
```
Solution: Windowed $pq$-Gram Bases

Algorithm 2: **Form bases** from a sorted sibling sequence

1. if sibling sequence $<$ window then extend with dummy nodes;

- **Example:** stem, sorted sibling sequence, window $w = 3$

```
      a
     / \
    b   c
   /   /\
  d   e  *
```
Solution: Windowed \(pq\)-Gram Bases

Algorithm 3: **Form bases** from a sorted sibling sequence

1. if sibling sequence < window then extend with dummy nodes;
2. initialize window: start with leftmost node;

**Example:** stem, sorted sibling sequence, window \(w = 3\)

```
    a
   / \
  b   c
 / \  / \
d  e  *  
```
Solution: **Windowed pq-Gram Bases**

Algorithm 4: **Form bases** from a sorted sibling sequence

1. if sibling sequence < window then extend with dummy nodes;
2. initialize window: start with leftmost node;
3. repeat
4. form bases in window: all q-permutations that contain start node;
7. until processed all window positions

**Example**: stem, sorted sibling sequence, window \( w = 3 \)

```plaintext
     a
    /|
   / |-----------------
  b  c
  /   /  d e *  \  
   \ /\       /   /
    \  \   /     /
       \   /     /
         \ /     /
          \   /   
           \ /   
            d  e
```

Nikolaus Augsten (Bolzano, Italy)
Solution: Windowed $pq$-Gram Bases

Algorithm 5: **Form bases** from a sorted sibling sequence

1. **if** sibling sequence $<$ window **then** extend with dummy nodes;
2. **initialize** window: start with leftmost node;
3. repeat
4. **form bases** in window: all $q$-permutations that contain start node;
7. **until** processed all window positions

**Example:** stem, sorted sibling sequence, window $w = 3$

```
  a
 /\  
 b  c
 
  d e *

\[ \rightarrow \]

  a  a
 / \ / 
 c  c
 
  d e d *
```
Solution: Windowed $pq$-Gram Bases

Algorithm 6: **Form bases** from a sorted sibling sequence

1. if sibling sequence $< \text{window}$ then extend with dummy nodes;
2. initialize window: start with leftmost node;
3. repeat
4. form bases in window: all $q$-permutations that contain start node;
5. shift window to the right by one node;
6. until processed all window positions

**Example:** stem, sorted sibling sequence, window $w = 3$

```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>
```

```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Nikolaus Augsten (Bolzano, Italy) Approximate Joins for Data-Centric XML ICDE 2008 – Cancún, Mexico 19 / 33
Solution: Windowed \( pq \)-Gram Bases

Algorithm 7: Form bases from a sorted sibling sequence

1. if sibling sequence < window then extend with dummy nodes;
2. initialize window: start with leftmost node;
3. repeat
4. form bases in window: all \( q \)-permutations that contain start node;
5. shift window to the right by one node;
6. if window extends the right border then wrap window;
7. until processed all window positions

Example: stem, sorted sibling sequence, window \( w = 3 \)

\[
\begin{array}{c}
\text{a} \\
\text{b} \quad \text{c} \\
\text{d} \quad \text{e} \star
\end{array}
\rightarrow
\begin{array}{c}
\text{a} \\
\text{c} \\
\text{d} \quad \text{e} \quad \text{d} \star
\end{array}
\]
Algorithm 8: **Form bases** from a sorted sibling sequence

1. if `sibling sequence < window` then extend with dummy nodes;
2. initialize window: start with leftmost node;
3. repeat
4. form bases in window: all \( q \)-permutations that contain start node;
5. shift window to the right by one node;
6. if `window extends the right border` then wrap window;
7. until processed all window positions

**Example:** stem, sorted sibling sequence, window \( w = 3 \)

```
   a
  / \       a     a     a     a
 b   c     c     c     c
|   |     |     |     |     |
|   |     |     |     |     |
|   |     |     |     |     |
|   |     |     |     |     |
```

```
      d e  d *  e *  e d
```
Solution: Windowed $pq$-Gram Bases

Algorithm 9: Form bases from a sorted sibling sequence

1. if sibling sequence $<$ window then extend with dummy nodes;
2. initialize window: start with leftmost node;
3. repeat
4. form bases in window: all $q$-permutations that contain start node;
5. shift window to the right by one node;
6. if window extends the right border then wrap window;
7. until processed all window positions

Example: stem, sorted sibling sequence, window $w = 3$
Solution: Windowed $pq$-Gram Bases

Algorithm 10: **Form bases** from a sorted sibling sequence

1. if sibling sequence $< \text{window}$ then extend with dummy nodes;
2. initialize window: start with leftmost node;
3. repeat
4. form bases in window: all $q$-permutations that contain start node;
5. shift window to the right by one node;
6. if window extends the right border then wrap window;
7. until processed all window positions

**Example:** stem, sorted sibling sequence, window $w = 3$

```
  a
 /|
 b c
 /  
 d e*
```

```
  a  a  a  a  a  a
 /|
 c  c  c  c  c  c
 /  
 d  e  d  e  e  d  e
```
Theorem (Optimal Windowed \(pq\)-Grams)

For trees with fanout \(f\), windowed \(pq\)-grams with base size \(q = 2\) and window size \(w = \frac{f+1}{2}\) have the following properties:
Theorem (Optimal Windowed pq-Grams)

For trees with fanout $f$, windowed pq-grams with base size $q = 2$ and window size $w = \frac{f+1}{2}$ have the following properties:

1. **Detection of node moves:**
   - base recall $\rho = 1$ (all sibling pairs are encoded)
   - base precision $\pi = 1$ (each pair is encoded only once)
Theorem (Optimal Windowed $pq$-Grams)

For trees with fanout $f$, windowed $pq$-grams with base size $q = 2$ and window size $w = \frac{f + 1}{2}$ have the following properties:

1. **Detection of node moves:**
   - **base recall** $\rho = 1$ (all sibling pairs are encoded)
   - **base precision** $\pi = 1$ (each pair is encoded only once)

2. **Robustness to different sortings:** ($k$ edit operations)
   - **base error** $\epsilon \leq \frac{2k}{f}$
Theorem (Optimal Windowed $pq$-Grams)

For trees with fanout $f$, windowed $pq$-grams with base size $q = 2$ and window size $w = \frac{f+1}{2}$ have the following properties:

1. **Detection of node moves:**
   - **Base recall** $\rho = 1$ (all sibling pairs are encoded)
   - **Base precision** $\pi = 1$ (each pair is encoded only once)

2. **Robustness to different sortings:** ($k$ edit operations)
   - **Base error** $\epsilon \leq \frac{2k}{f}$

3. **Balanced node weight:**
   - Each non-root node appears in exactly $2w - 2$ bases.
Illustration: Detection of Node Moves

- **Single Node:** each node forms a base of size $q = 1$
**Illustration: Detection of Node Moves**

- **Single Node:** each node forms a base of size $q = 1$

![Diagram of node moves]

**Goal:** bases must change
**Single Node:** each node forms a base of size $q = 1$

Goal: bases must change

| Single Node: | c, d, e | no bases change | c, d, e |
Illustration: Detection of Node Moves

**Single Node:** each node forms a base of size $q = 1$

---

Goal: bases must change

- **✗ Single Node:** c, d, e  no bases change  c, d, e

---

Diagram:

```
        a
       / \
      b   b
     / \  /
    c   d e
```

1 node move

```
        a
       / \
      b   b
     /     /
    c     d
```

---
Illustration: Detection of Node Moves

- **Single Node:** each node forms a base of size $q = 1$
- **Window:** $q \geq 2$ nodes of a window form a base

Goal: bases must change

<table>
<thead>
<tr>
<th>Single Node:</th>
<th>Window:</th>
</tr>
</thead>
<tbody>
<tr>
<td>X c, d, e no bases change c, d, e</td>
<td>cd, c*, d*, dc, <em>c, <em>d, e</em>, ... 33% bases change c</em>, c*, **, *c, *c, **, de, ...</td>
</tr>
</tbody>
</table>
Illustration: Detection of Node Moves

- **Single Node**: each node forms a base of size $q = 1$
- **Window**: $q \geq 2$ nodes of a window form a base

![Diagram of node moves](image)

**Goal**: bases must change

- **× Single Node**: $c, d, e$  no bases change  $c, d, e$
- **✓ Window**: $cd, c^*, d^*, dc, *c, *d, e^*, \ldots$  33% bases change  $c^*, c^*, **, *c, *c, **, de, \ldots$

**Windowed $pq$-grams detect node moves.**
Illustration: Robustness to Different Sortings

- **Consecutive siblings** form a base (no permutation)
Illustration: Robustness to Different Sortings

- **Consecutive siblings** form a base (no permutation)

```
  x a b d  1 rename  x a c d
    Sorting A                      Sorting A
      x a b d                      x a c d
        Sorting B                  Sorting B
          x a b d                  x a c d
```

**Goal:** Same number of bases change for both sortings.
Illustration: Robustness to Different Sortings

- **Consecutive siblings** form a base (no permutation)

<table>
<thead>
<tr>
<th>Goal:</th>
<th>Same number of bases change for both sortings.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consecutive:</td>
<td>Sorting A</td>
</tr>
<tr>
<td>Sort A</td>
<td>ab bc</td>
</tr>
<tr>
<td>Sort B</td>
<td>ad db</td>
</tr>
</tbody>
</table>
Illustration: Robustness to Different Sortings

- **Consecutive siblings** form a base (no permutation)

Goal: Same number of bases change for both sortings.

<table>
<thead>
<tr>
<th>Consecutive:</th>
<th>Sort A</th>
<th>Sort B</th>
<th>100% bases change</th>
<th>50% bases change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort A</td>
<td>ab bc</td>
<td>ad db</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sort B</td>
<td>ac cd</td>
<td>ad dc</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Illustration: Robustness to Different Sortings

- **Consecutive siblings** form a base (no permutation)
- **Window**: all sibling permutations within the window form bases

\[
\begin{array}{ccc}
\times & a & d \\
\times & b & d \\
\times & a & b & d \\
\end{array} & \xrightarrow{1 \text{ rename}} & \begin{array}{ccc}
\times & a & d \\
\times & c & d \\
\times & a & c & d \\
\end{array}
\]

**Sorting A**

\[
\begin{array}{ccc}
\times & a & b \\
\times & a & d \\
\times & a & c & d \\
\end{array}
\]

**Sorting B**

\[
\begin{array}{ccc}
\times & a & c \\
\times & a & d \\
\times & a & d & c \\
\end{array}
\]

**Goal**: Same number of bases change for both sortings.

<table>
<thead>
<tr>
<th><strong>Consecutive</strong></th>
<th>Sort A</th>
<th>ab bc</th>
<th>100% bases change</th>
<th>Sort B</th>
<th>ad db</th>
<th>50% bases change</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Window</strong></td>
<td>Sort A</td>
<td>ad ab db...</td>
<td>33% bases change</td>
<td>Sort B</td>
<td>ad ac dc...</td>
<td>33% bases change</td>
</tr>
<tr>
<td></td>
<td>Sort B</td>
<td>ad ab db...</td>
<td>33% bases change</td>
<td></td>
<td>ad ac dc...</td>
<td>33% bases change</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Nikolaus Augsten (Bolzano, Italy) Approximate Joins for Data-Centric XML ICDE 2008 – Cancún, Mexico
Illustration: Robustness to Different Sortings

- **Consecutive siblings** form a base (no permutation)
- **Window**: all sibling permutations within the window form bases

<table>
<thead>
<tr>
<th></th>
<th><strong>Sort A</strong></th>
<th><strong>Sort B</strong></th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consecutive</strong></td>
<td>ab bc</td>
<td>ad db</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Window</strong></td>
<td>ad ab db...</td>
<td>ad ac dc...</td>
<td>33%</td>
</tr>
</tbody>
</table>

**Goal**: Same number of bases change for both sortings.

Windowed pq-grams: Robust to different sortings.
Illustration: Balancing the Node Weight

- **Permutations**: all permutations of size $q$ form a base
Illustration: Balancing the Node Weight

- **Permutations**: all permutations of size $q$ form a base

![Diagram showing permutations and renames]

**Goal**: Same number of bases change for both renames.
**Permutations:** all permutations of size $q$ form a base

**Goal:** Same number of bases change for both renames.

**Permutations:** 60/137 bases change       6/137 bases change
Illustration: Balancing the Node Weight

**Permutations:** all permutations of size $q$ form a base

- **Goal:** Same number of bases change for both renames.

- **Permutations:** $\frac{60}{137}$ bases change vs $\frac{6}{137}$ bases change
Illustration: Balancing the Node Weight

- **Permutations**: all permutations of size $q$ form a base
- **Window**: only permutations within window form a base

Goal: Same number of bases change for both renames.

<table>
<thead>
<tr>
<th></th>
<th>Permutations</th>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>60/137 bases change</td>
<td>12/51 bases change</td>
</tr>
<tr>
<td>✗ Permutations:</td>
<td>6/137 bases change</td>
<td>12/51 bases change</td>
</tr>
</tbody>
</table>
Illustration: Balancing the Node Weight

- **Permutations**: all permutations of size $q$ form a base
- **Window**: only permutations within window form a base

Goal: Same number of bases change for both renames.

<table>
<thead>
<tr>
<th></th>
<th># of Bases Change</th>
<th># of Bases Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permutations</td>
<td>60/137</td>
<td>6/137</td>
</tr>
<tr>
<td>Window</td>
<td>12/51</td>
<td>12/51</td>
</tr>
</tbody>
</table>

Windowed $pq$-grams: Node weight is independent of sibling number.
Outline

1 Motivation

2 Windowed $pq$-Grams for Data-Centric XML
   - Windowed $pq$-Grams
   - Tree Sorting
   - Forming Bases

3 Efficient Approximate Joins with Windowed $pq$-Gram

4 Experiments

5 Related Work

6 Conclusion and Future Work
Approximate Join

\[
F
\begin{array}{c|c}
\text{tid} & \text{tree} \\
\hline
T_1 & x \ y \ w \\
& v \ z \\
T_2 & a \ b \ c \ b \\
T_3 & a \ e \ b \ h \\
\end{array}
\]

\[
F'
\begin{array}{c|c}
\text{tree} & \text{tid} \\
\hline
a \ b \ c \ d \ e \\
T_1' \\
T_2' & d \ a \ h \ i \\
T_3' & x \ y \ w \\
& w \ z \\
\end{array}
\]
**Approximate Join**

- **Simple approach:** distance join
  - 1. compute **distance** between all pairs of trees
Approximate Join

Simple approach: distance join

1. compute distance between all pairs of trees
2. return document pairs within threshold
Simple approach: distance join
1. compute distance between all pairs of trees
2. return document pairs within threshold

Very expensive: $N^2$ distance computations!
Usual Join Optimization Does not Apply

- **Distance join**: expensive
  - nested loop join: evaluate distance function between every input pair

- **Equality join**: efficient
  - implementation as sort-merge or hash join
Usual Join Optimization Does not Apply

- **Distance join**: expensive
  - nested loop join: evaluate distance function between every input pair

- **Equality join**: efficient
  - implementation as sort-merge or hash join

- **Sort-merge and hash join**:  
  - *first step*: treat each *join attribute in isolation* (sort/hash)  
  - *second step*: evaluate equality function

- **Sort-merge and hash not applicable to distance join**:  
  - there is **no sorting** that groups similar trees  
  - there is **no hash function** that partitions similar trees into buckets
Efficient Approximate Joins with Windowed $pq$-Gram

Usual Join Optimization Does not Apply

- **Distance join**: expensive
  - nested loop join: evaluate distance function between every input pair
- **Equality join**: efficient
  - implementation as sort-merge or hash join

**Sort-merge and hash join:**
- *first step*: treat each join attribute in isolation (sort/hash)
- *second step*: evaluate equality function

Sort-merge and hash **not applicable to distance join**:
- there is **no sorting** that groups similar trees
- there is **no hash function** that partitions similar trees into buckets

**Solution**: reduce **distance join to equality join** on $pq$-grams
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: $N^2$ intersections between integer bags
  
  \[
  \begin{align*}
  \{1, 7\}_a & \quad \{1, 7\}_d \\
  \{1, 0\}_b & \quad \{5, 5\}_e \\
  \{4, 6\}_c & \quad \{0, 8\}_f
  \end{align*}
  \]
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: $N^2$ intersections between integer bags

  \[
  \begin{align*}
  \{1, 7\}_a & \quad \{1, 7\}_d \quad |a \cap d| = 2 \\
  \{1, 0\}_b & \quad \{5, 5\}_e \quad |a \cap e| = 0 \\
  \{4, 6\}_c & \quad \{0, 8\}_f \quad |a \cap f| = 0
  \end{align*}
  \]
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: $N^2$ intersections between integer bags

  - $\{1, 7\}_a \cap \{1, 7\}_d = 2$
  - $\{1, 7\}_a \cap \{0, 8\}_f = 0$
  - $\{1, 7\}_a \cap \{5, 5\}_e = 0$

  - $\{1, 0\}_b \cap \{1, 7\}_d = 1$
  - $\{1, 0\}_b \cap \{0, 8\}_f = 1$
  - $\{1, 0\}_b \cap \{5, 5\}_e = 0$

  - $\{4, 6\}_c \cap \{1, 7\}_d = 0$
  - $\{4, 6\}_c \cap \{0, 8\}_f = 0$
  - $\{4, 6\}_c \cap \{5, 5\}_e = 0$
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: $N^2$ intersections between integer bags
  
  |  |  |
  |---|---|---|
  | $\{1, 7\}_a$ | $\{1, 7\}_d$ | $|a \cap d| = 2$ |
  | $\{1, 0\}_b$ | $\{5, 5\}_e$ | $|b \cap d| = 1$ |
  | $\{4, 6\}_c$ | $\{0, 8\}_f$ | $|c \cap d| = 0$ |

- **Optimized pq-gram join**: empty intersections are never computed!
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: $N^2$ intersections between integer bags
  \[
  \{1, 7\}_a \cap \{1, 7\}_d = 2 \quad |a \cap d| = 2 \\
  \{1, 0\}_b \cap \{5, 5\}_e = 0 \quad |b \cap e| = 0 \\
  \{4, 6\}_c \cap \{0, 8\}_f = 0 \quad |c \cap f| = 0
  \]

- **Optimized pq-gram join**: empty intersections are never computed!
  1. union
  \[
  \{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\} \quad \{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}
  \]
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: $N^2$ intersections between integer bags

  \[
  \{1, 7\}_a \cap \{1, 7\}_d | a \cap d | = 2 \\
  \{1, 0\}_b \cap \{5, 5\}_e | b \cap e | = 1 \\
  \{4, 6\}_c \cap \{0, 8\}_f | c \cap f | = 0
  \]

- **Optimized pq-gram join**: empty intersections are never computed!

  1. union

     \[
     \{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\} \cup \{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}
     \]

  2. sort

     \[
     0_b, 0_f, 1_a, 1_d, 1_b, 5_e, 4_c, 5_e, 6_c, 7_d, 7_a, 8_f
     \]
Reduction of a Distance Join to an Equality Join

- **Distance join** between trees: $N^2$ intersections between integer bags
  \[
  \{1, 7\}_a \quad \{1, 7\}_d \\
  \{1, 0\}_b \quad \{5, 5\}_e \\
  \{4, 6\}_c \quad \{0, 8\}_f \\
  \]
  \[
  |a \cap d| = 2 \quad |a \cap e| = 0 \quad |a \cap f| = 0 \\
  |b \cap d| = 1 \quad |b \cap e| = 0 \quad |b \cap f| = 1 \\
  |c \cap d| = 0 \quad |c \cap e| = 0 \quad |c \cap f| = 0 \\
  \]

- **Optimized pq-gram join**: Empty intersections are never computed!
  1. union
  \[
  \{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\} \quad \{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\} \\
  \]
  2. sort
  3. merge-join
    \[
    0_b \quad 0_f \\
    1_a \quad 1_d \\
    1_b \quad 5_e \\
    4_c \quad 5_e \\
    6_c \quad 7_d \\
    7_a \quad 8_f \\
    \]
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: $N^2$ intersections between integer bags
  \[
  \begin{align*}
  \{1, 7\}_a & \cap \{1, 7\}_d = 2 & a \cap d &= 2 \\
  \{1, 0\}_b & \cap \{5, 5\}_e = 1 & b \cap e &= 1 \\
  \{4, 6\}_c & \cap \{0, 8\}_f = 0 & c \cap f &= 0
  \end{align*}
  \]

- **Optimized pq-gram join**: empty intersections are never computed!
  1. union
     \[
     \begin{align*}
     \{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\} & \cup \{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}
     \end{align*}
     \]
  2. sort
  3. merge-join
     \[
     \begin{align*}
     0_b & \cap 0_f = 0 \\
     1_a & \cap 1_d = 1 \\
     1_b & \cap 5_e = 1 \\
     4_c & \cap 5_e = 1 \\
     6_c & \cap 7_d = 1 \\
     7_a & \cap 8_f = 1
     \end{align*}
     \]
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: \( N^2 \) intersections between integer bags

\[
\begin{array}{ccc}
\{1, 7\}_a & \{1, 7\}_d & |a \cap d| = 2 \\
\{1, 0\}_b & \{5, 5\}_e & |b \cap d| = 1 \\
\{4, 6\}_c & \{0, 8\}_f & |c \cap d| = 0
\end{array}
\]

- **Optimized pq-gram join**: empty intersections are never computed!

1. union

\[
\begin{array}{ccc}
\{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\} & \{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}
\end{array}
\]

2. sort

3. merge-join

\[
\begin{array}{c}
\begin{array}{c}
0_b \\
1_a \\
1_b \\
4_c \\
6_c \\
7_a
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
0_f \\
1_d \\
5_e \\
5_e \\
7_d \\
8_f
\end{array}
\end{array}
\]

\[
\begin{array}{c}
|b \cap f| \\
|a \cap d|
\end{array}
\]
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: \( N^2 \) intersections between integer bags

  \[
  \begin{align*}
  \{1, 7\} & \cap \{1, 7\} = 2 \\
  \{1, 0\} & \cap \{5, 5\} = 1 \\
  \{4, 6\} & \cap \{0, 8\} = 0
  \end{align*}
  \]

- **Optimized \( pq \)-gram join**: empty intersections are never computed!

  1. union
     \[
     \{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\} \cup \{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}
     \]
  2. sort
  3. merge-join

\[
\begin{array}{c}
1_a & 1_b & 4_c & 6_c & 7_a & 0_b & 0_f & 1_d & 5_e & 5_e & 7_d & 8_f \\
\end{array}
\]
Reducing a Distance Join to an Equality Join

- **Distance join** between trees: \( N^2 \) intersections between integer bags

\[
\begin{align*}
\{1, 7\} \cap \{1, 7\} &= 2 \\
\{1, 0\} \cap \{5, 5\} &= 1 \\
\{4, 6\} \cap \{0, 8\} &= 0
\end{align*}
\]

- **Optimized \( pq \)-gram join**: empty intersections are never computed!

  1. union

\[
\begin{align*}
\{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\} \\
\{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}
\end{align*}
\]

  2. sort

  3. merge-join
Outline

1. Motivation
2. Windowed $pq$-Grams for Data-Centric XML
   - Windowed $pq$-Grams
   - Tree Sorting
   - Forming Bases
3. Efficient Approximate Joins with Windowed $pq$-Gram
4. Experiments
5. Related Work
6. Conclusion and Future Work
Experiments

Effectiveness of the Windowed $pq$-Gram Join
Experiments

Effectiveness of the Windowed $pq$-Gram Join

**Experiment:** match DBLP articles
- add noise to articles (missing elements and spelling mistakes)
- approximate join between original and noisy data
- measure precision and recall for different thresholds

**Windowed $pq$-grams are effective for data-centric XML**
Experiments

Effectiveness of the Windowed $pq$-Gram Join

**Experiment:** match DBLP articles
- add noise to articles (missing elements and spelling mistakes)
- approximate join between original and noisy data
- measure precision and recall for different thresholds

**Datasets:**
- **DBLP:** articles
depth 1.9, 15 nodes (max 1494 nodes)
- **SwissProt:** protein descriptions
depth 3.5, 104 nodes (max 2640 nodes)
- **Treebank:** tagged English sentences
depth 6.9 (max depth 30), 43 nodes

Windowed $pq$-grams are effective for data-centric XML
Efficiency of the Optimized $pq$-Gram Join
Efficiency of the Optimized $pq$-Gram Join

Optimized $pq$-gram join: very efficient

- compute nested-loop join between trees
- compute optimized $pq$-gram join between trees
- measure wallclock time
Outline

1. Motivation

2. Windowed \( pq \)-Grams for Data-Centric XML
   - Windowed \( pq \)-Grams
   - Tree Sorting
   - Forming Bases

3. Efficient Approximate Joins with Windowed \( pq \)-Gram

4. Experiments

5. Related Work

6. Conclusion and Future Work
Distances between Unordered Trees

**Edit Distances between Unordered Trees**
- [Zhang et al., 1992]: proof for NP-completeness
- [Kailing et al., 2004]: lower bound for a restricted edit distance
- [Chawathe and Garcia-Molina, 1997]: $O(n^3)$ heuristics
- Our solution: $O(n \log n)$ approximation

**Approximate Join**
- [Gravano et al., 2001]: efficient approximate join for strings
Windowed \( pq \)-grams for unordered trees:

- \( O(n \log n) \) approximation of NP-complete edit distance
- **Key problem:** all permutations must be considered
- **Our approach:** sort trees and simulate permutations with window
- **Sorting:** works for \( pq \)-grams, but not for edit distance
- **Window technique** guarantees core properties
  - detection of node moves
  - robustness to different sortings
  - balanced node weight
- **Efficient approximate join:** reduces distance join to equality join
Windowed \( pq \)-grams for unordered trees:

- \( O(n \log n) \) approximation of NP-complete edit distance
- **Key problem:** all permutations must be considered
- **Our approach:** sort trees and simulate permutations with window
- **Sorting:** works for \( pq \)-grams, but not for edit distance
- **Window technique** guarantees core properties
  - detection of node moves
  - robustness to different sortings
  - balanced node weight
- **Efficient approximate join:** reduces distance join to equality join

**Future work:**

- incremental updates of the windowed \( pq \)-gram index
- include approximate string matching into XML distance


Kaizhong Zhang, Richard Statman, and Dennis Shasha.
On the editing distance between unordered labeled trees.