Approximate Joins for Data-Centric XML

Nikolaus Augsten\textsuperscript{1} Michael Böhlen\textsuperscript{1} Curtis Dyreson\textsuperscript{2} 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
Query: Give me all album pairs that represent the same music CDs.

How similar are two XML items?
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 vs. Unordered Trees

**Ordered Trees**
- Sibling order matters

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

**Unordered Trees**
- Sibling order ignored

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

= data-centric XML

**Edit distance** between unordered trees: NP-complete

→ All sibling permutations must be considered!
Motivation

Problem Definition

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**
  
- **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:**
  
  \[ \begin{array}{cccccccccccc}
  a & b & c & d & e \\
  * & * & * & * & * & a & a & a & a & a & a & c & c \\
  a & a & a & a & a & b & c & c & c & c & c & d & e \\
  b & c & b & * & c & * & c & b & * & b & * & c & * & d & e & d & e & * & e & d & * & d & * & e & * & * & * & * \\
  \end{array} \]

- **Hashing:** map pq-gram to integer:

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

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

  **Note:** labels may be strings of arbitrary length!

- **pq-Gram index:** bag of hashed pq-grams

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

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

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

- **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:
  - ![Tree Diagram]

- **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

**Edit distance:** tree sorting does not work

**Windowed pq-Grams:** tree sorting works OK
1. **Non-unique sorting**: edit distance $O(n)$ for identical trees

![Diagram showing non-unique sorting with edit distance $O(n)$ for identical trees.](image)
2. **Node renaming**: edit distance depends on node label

![Diagram showing node renaming and edit distance](image)
Windowed *pq*-Grams for Data-Centric XML

✓ **Windowed *pq*-Grams: Tree Sorting Works OK**

**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

- **Stems:** ignore sibling order

- **Bases:** do not ignore sibling order!
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
**Algorithm 1: 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 \( w = 3 \)

\[
\begin{array}{c}
\text{a} \\
\text{b} \ \text{c} \\
\text{d} \ \text{e} \ *
\end{array} \quad \rightarrow \quad \begin{array}{cccccccc}
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} \\
\text{c} & \text{c} & \text{c} & \text{c} & \text{c} & \text{c} & \text{c} \\
\text{d} & \text{e} & \text{d} & \text{e} & \text{e} & \text{d} & \text{d} & \text{e} & \text{e}
\end{array}
\]
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 $q = 1$
- **Window:** $q \geq 2$ nodes of a window form a base

Goal: bases must change

<table>
<thead>
<tr>
<th>Single Node:</th>
<th>c, d, e</th>
<th>no bases change</th>
<th>c, d, e</th>
</tr>
</thead>
</table>

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

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

Windowed \( pq \)-Grams: Robust to different sortings.

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

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

Nikolaus Augsten (Bolzano, Italy) Approximate Joins for Data-Centric XML ICDE 2008 – Cancún, Mexico 22 / 33
**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>✗</td>
<td>6/137 bases change</td>
<td>12/51 bases change</td>
</tr>
</tbody>
</table>

**Windowed $pq$-grams:** Node weight is independent of sibling number.
Efficient Approximate Joins with Windowed \(pq\)-Gram

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
Efficient Approximate Joins with Windowed $pq$-Gram

Approximate Join

$F$  
| $tid$ | tree | $F'$     
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>$x$</td>
<td>$y$ $w$ $v$ $z$</td>
</tr>
<tr>
<td>$T_2$</td>
<td>$a$ $b$ $c$ $b$</td>
<td>$d$ $a$ $h$ $i$</td>
</tr>
<tr>
<td>$T_3$</td>
<td>$a$ $e$ $b$ $h$</td>
<td>$x$ $y$ $w$ $z$</td>
</tr>
</tbody>
</table>

$F$ threshold=2

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

Very expensive: $N^2$ distance computations!
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
Distance join between trees: $N^2$ intersections between integer bags

- $\{1, 7\}_a \cap \{1, 7\}_d = 2$
- $\{1, 7\}_a \cap \{5, 5\}_e = 0$
- $\{1, 7\}_a \cap \{0, 8\}_f = 0$
- $\{1, 0\}_b \cap \{5, 5\}_e = 1$
- $\{1, 0\}_b \cap \{0, 8\}_f = 0$
- $\{4, 6\}_c \cap \{0, 8\}_f = 0$

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

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

**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
Experiments

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
Related 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

Future work:

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