Approximate Joins for Data-Centric XML

Nikolaus Augsten[1] Michael Böhlen[1] Curtis Dyreson[2] Johann Gamper[1]

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

[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

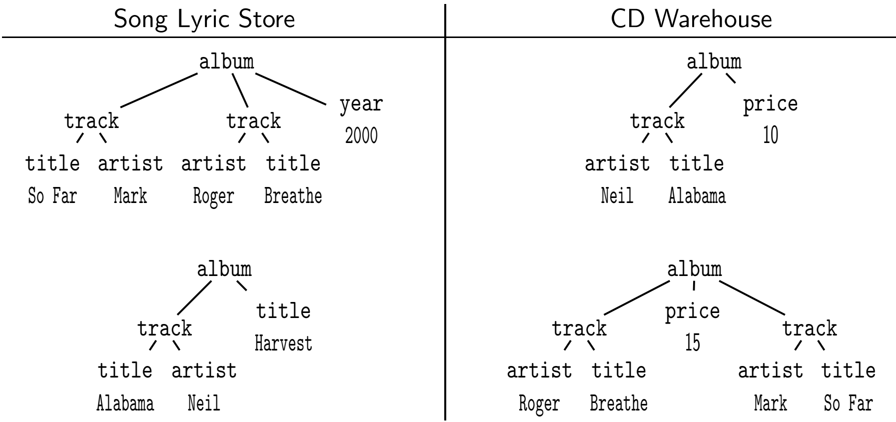
3 Efficient Approximate Joins with Windowed *pq*-Gram

4 Experiments

5 Related Work

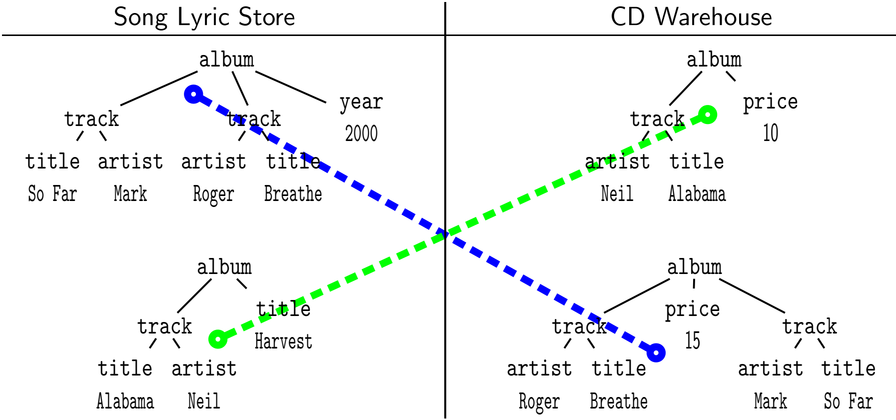
6 Conclusion and Future Work

Approximate Join on Music CDs



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

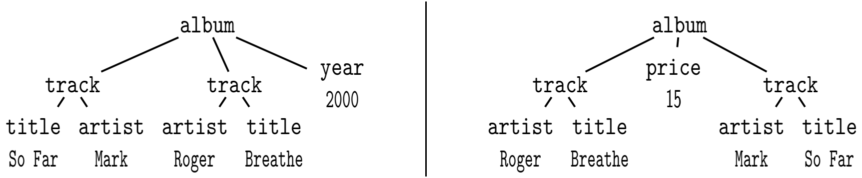
Approximate Join on Music CDs



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

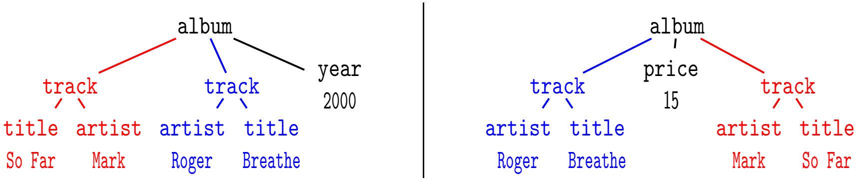
How similar are two XML items?

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.

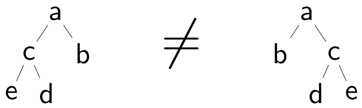
How Similar Are these XMLs?



- **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

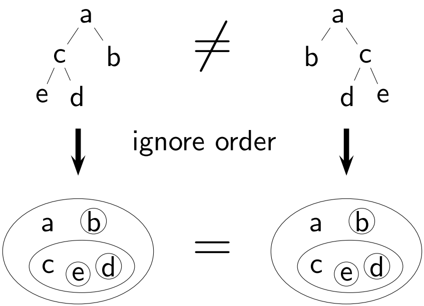
Ordered vs. Unordered Trees

Ordered Trees
sibling order matters



Ordered vs. Unordered Trees

Ordered Trees
sibling order matters

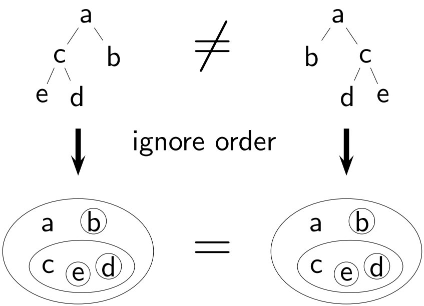


Unordered Trees
= data-centric XML
sibling order ignored

Ordered vs. Unordered Trees

Ordered Trees
sibling order matters

Unordered Trees
= data-centric XML
sibling order ignored



- **Edit distance** between unordered trees: **NP-complete**
  $\longrightarrow$ 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**.

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

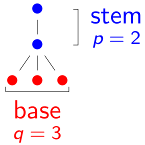
4 Experiments

5 Related Work

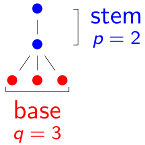
6 Conclusion and Future Work

Our Solution: Windowed *pq*-Grams

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

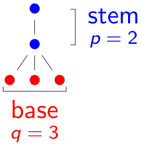


Our Solution: Windowed *pq*-Grams



- **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

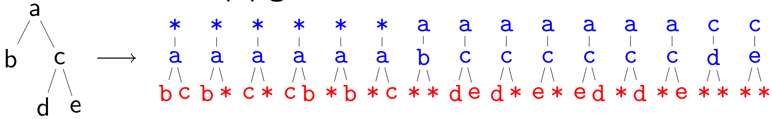
Our Solution: Windowed *pq*-Grams



- **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

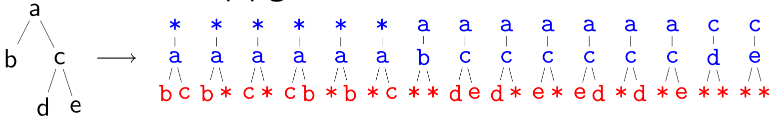
Implementation of Windowed *pq*-Grams

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

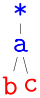


Implementation of Windowed *pq*-Grams

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



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



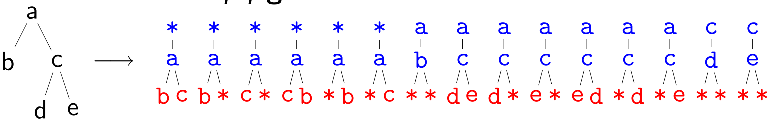
$\xrightarrow{\textit{serialize}}$ (*, a, b, c) $\xrightarrow{\textit{(shorthand)}}$ *abc $\xrightarrow{\textit{hash}}$ 0973

label l	$h(l)$
*	0
a	9
b	7
c	3
...	...

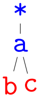
Note: labels may be strings of arbitrary length!

Implementation of Windowed *pq*-Grams

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



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



$\xrightarrow{serialize}$ (*, a, b, c) $\xrightarrow{(shorthand)}$ *abc $\xrightarrow{hash}$ 0973

label l	$h(l)$
*	0
a	9
b	7
c	3
...	...

Note: labels may be strings of arbitrary length!

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

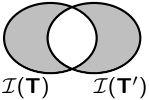
$$\mathcal{I}(\mathbf{T}) = \{0973, 0970, 0930, 0937, 0907, 0903, 9700, 9316, 9310, 9360, 9361, 9301, 9306, 3100, 3600\}$$

Tree is represented by a bag of integers!

The Windowed *pq*-Gram Distance

- The **windowed** *pq*-**gram distance** between two trees, **T** and **T**′:

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

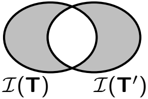


The Windowed *pq*-Gram Distance

- The **windowed** *pq*-**gram distance** between two trees, **T** and **T**′:

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

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

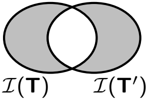


The Windowed *pq*-Gram Distance

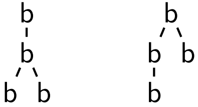
- The **windowed *pq*-gram distance** between two trees, $\mathbf{T}$ and $\mathbf{T}'$:

$$\mathrm{dist}^{pq}(\mathbf{T}, \mathbf{T}') = |\mathcal{I}(\mathbf{T}) \uplus \mathcal{I}(\mathbf{T}')| - 2|\mathcal{I}(\mathbf{T}) \cap \mathcal{I}(\mathbf{T}')|$$

- **Pseudo-metric** properties hold:
  - ✓ self-identity: $x = y \not\Leftarrow \Rightarrow \mathrm{dist}^{pq}(x, y) = 0$



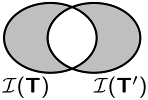
- Different trees may be at distance zero:



The Windowed *pq*-Gram Distance

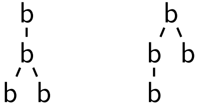
- The **windowed *pq*-gram distance** between two trees, **T** and **T**′:

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



- **Pseudo-metric** properties hold:
  - ✓ self-identity: $x = y \not\Leftarrow\Rightarrow \text{dist}^{pq}(x,y) = 0$
  - ✓ symmetry: $\text{dist}^{pq}(x,y) = \text{dist}^{pq}(y,x)$

- Different trees may be at distance zero:

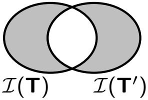


The Windowed *pq*-Gram Distance

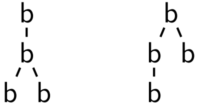
- The **windowed *pq*-gram distance** between two trees, $\mathbf{T}$ and $\mathbf{T}'$:

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

- **Pseudo-metric** properties hold:
  - ✓ self-identity: $x = y \not\Leftarrow \Rightarrow \text{dist}^{pq}(x, y) = 0$
  - ✓ symmetry: $\text{dist}^{pq}(x, y) = \text{dist}^{pq}(y, x)$
  - ✓ triangle inequality: $\text{dist}^{pq}(x, z) \leq \text{dist}^{pq}(x, y) + \text{dist}^{pq}(y, z)$



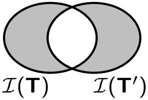
- Different trees may be at distance zero:



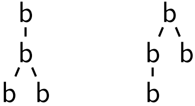
The Windowed *pq*-Gram Distance

- The **windowed** *pq*-**gram distance** between two trees, **T** and **T**′:

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



- **Pseudo-metric** properties hold:
  - ✓ self-identity: $x = y \not\Leftarrow \Rightarrow \text{dist}^{pq}(x,y) = 0$
  - ✓ 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:



- **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

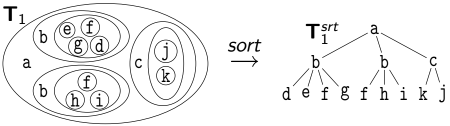
4 Experiments

5 Related Work

6 Conclusion and Future Work

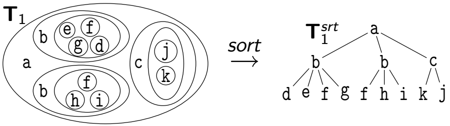
Sorting the Tree?

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



Sorting the Tree?

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

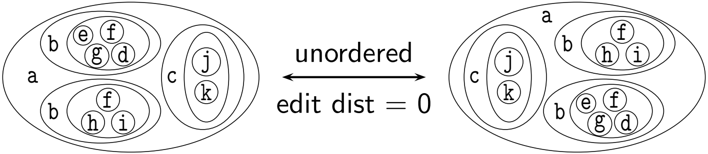


✘ **Edit distance**: tree sorting does not work

✓ **Windowed *pq*-Grams**: tree sorting works OK

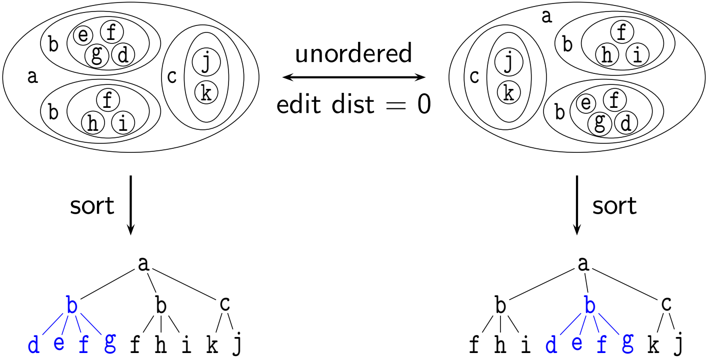
✘ Edit Distance: Tree Sorting Does Not Work

1. **Non-unique sorting**:



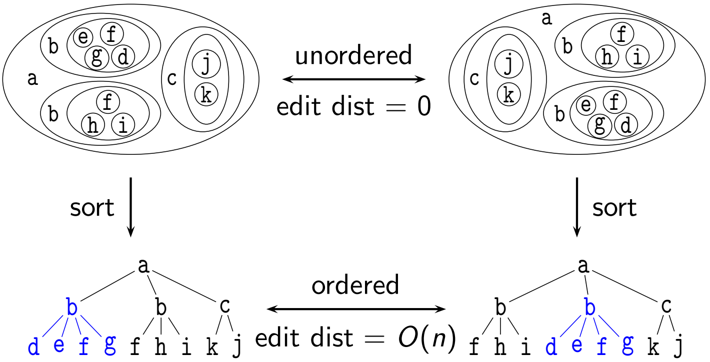
✘ Edit Distance: Tree Sorting Does Not Work

1. **Non-unique sorting**:



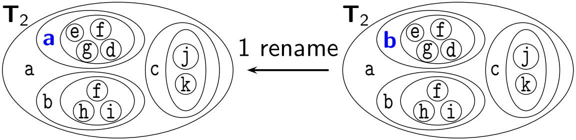
✘ Edit Distance: Tree Sorting Does Not Work

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



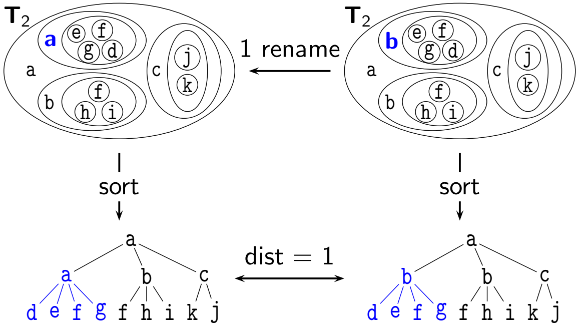
✘ Edit Distance: Tree Sorting Does Not Work

2. **Node renaming**:



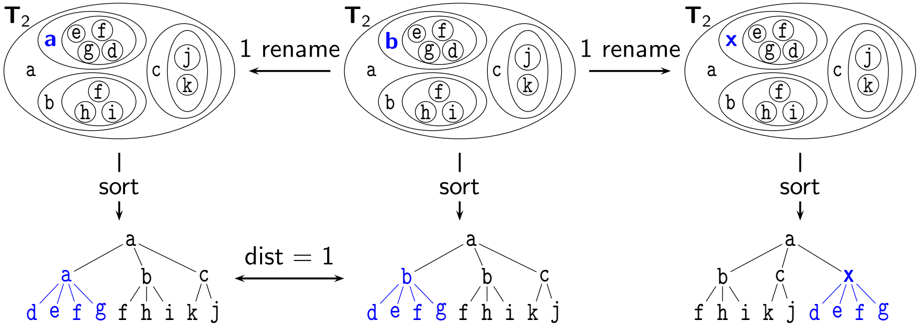
✘ Edit Distance: Tree Sorting Does Not Work

2. **Node renaming**:



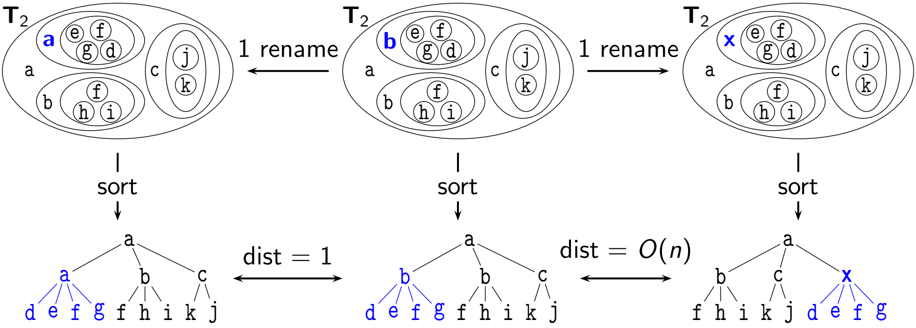
✘ Edit Distance: Tree Sorting Does Not Work

2. **Node renaming**:



✘ Edit Distance: Tree Sorting Does Not Work

2. **Node renaming**: edit distance depends on node label

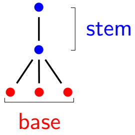


✓ Windowed *pq*-Grams: Tree Sorting Works OK

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

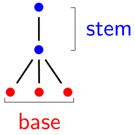


✓ Windowed *pq*-Grams: Tree Sorting Works OK

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

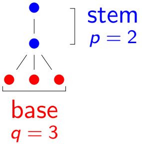
✓ **Node renaming** is independent of the node label

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

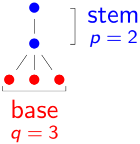
How To Form Bases?

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

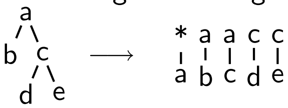


How To Form Bases?

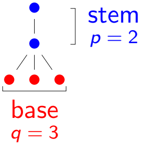
- **Goal** for windowed *pq*-grams:
  - **not sensitive** to the sibling order
  - **sensitive** to any other change in the tree



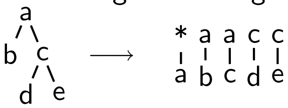
- **Stems**: ignore sibling order



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

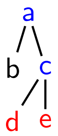
Requirements for Bases

- **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 $w = 3$



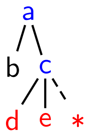
Solution: Windowed *pq*-Gram Bases

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 $w = 3$

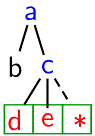


Solution: Windowed *pq*-Gram Bases

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 $w = 3$



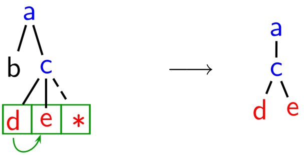
Solution: Windowed *pq*-Gram Bases

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

```
if sibling sequence < window then extend with dummy nodes;
initialize window: start with leftmost node;
repeat
    form bases in window: all q-permutations that contain start node;

until processed all window positions
```

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



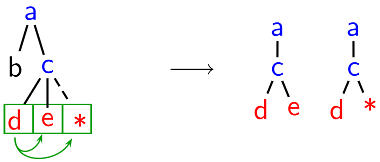
Solution: Windowed *pq*-Gram Bases

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

```
if sibling sequence < window then extend with dummy nodes;
initialize window: start with leftmost node;
repeat
    form bases in window: all q-permutations that contain start node;

until processed all window positions
```

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



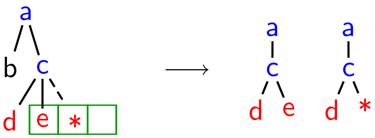
Solution: Windowed *pq*-Gram Bases

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

if *sibling sequence* < *window* then extend with dummy nodes;
initialize window: start with leftmost node;
repeat
    form bases in window: all *q*-permutations that contain start node;
    shift window to the right by one node;

until *processed all window positions*

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

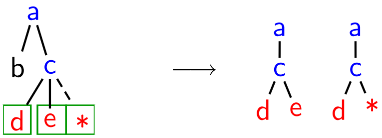


Solution: Windowed *pq*-Gram Bases

Algorithm 7: Form bases from a sorted sibling sequence

```
if sibling sequence < window then extend with dummy nodes;
initialize window: start with leftmost node;
repeat
    form bases in window: all q-permutations that contain start node;
    shift window to the right by one node;
    if window extends the right border then wrap window;
until processed all window positions
```

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

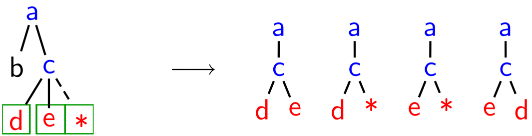


Solution: Windowed *pq*-Gram Bases

Algorithm 8: **Form bases** from a sorted sibling sequence

```
if sibling sequence < window then extend with dummy nodes;
initialize window: start with leftmost node;
repeat
    form bases in window: all q-permutations that contain start node;
    shift window to the right by one node;
    if window extends the right border then wrap window;
until processed all window positions
```

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

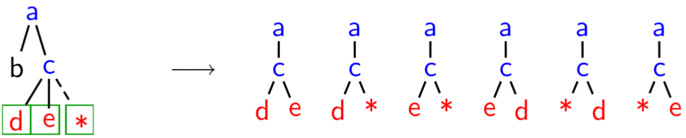


Solution: Windowed *pq*-Gram Bases

Algorithm 9: Form bases from a sorted sibling sequence

```
if sibling sequence < window then extend with dummy nodes;
initialize window: start with leftmost node;
repeat
    form bases in window: all q-permutations that contain start node;
    shift window to the right by one node;
    if window extends the right border then wrap window;
until processed all window positions
```

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

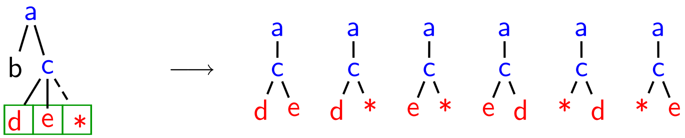


Solution: Windowed *pq*-Gram Bases

Algorithm 10: Form bases from a sorted sibling sequence

```
if sibling sequence < window then extend with dummy nodes;
initialize window: start with leftmost node;
repeat
    form bases in window: all q-permutations that contain start node;
    shift window to the right by one node;
    if window extends the right border then wrap window;
until processed all window positions
```

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



Optimal Windowed *pq*-Grams

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:

Optimal Windowed *pq*-Grams

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)*

Optimal Windowed *pq*-Grams

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)*

$$\textit{base error } \epsilon \leq \frac{2k}{f}$$

Optimal Windowed *pq*-Grams

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)*

   $$\text{base error } \epsilon \leq \frac{2k}{f}$$

3. **Balanced node weight:**

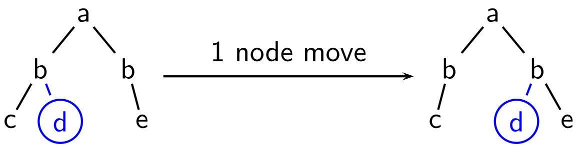
   Each non-root node appears in exactly $2w - 2$ bases.

Illustration: Detection of Node Moves

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

Illustration: Detection of Node Moves

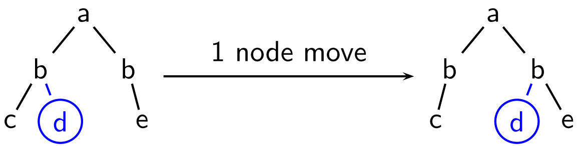
- **Single Node:** each node forms a base of size $q = 1$



Goal: bases must change

Illustration: Detection of Node Moves

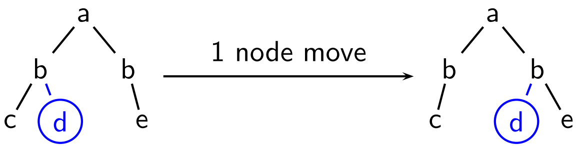
- **Single Node:** each node forms a base of size $q = 1$



Goal:		bases must change	
Single Node:	c, d, e	no bases change	c, d, e

Illustration: Detection of Node Moves

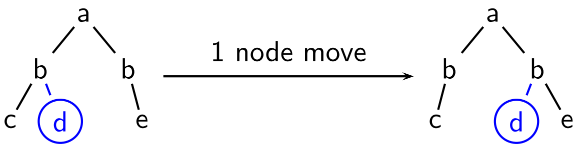
- **Single Node:** each node forms a base of size $q = 1$



Goal:		bases must change	
✘ **Single Node:**	c, d, e	no bases change	c, d, e

Illustration: Detection of Node Moves

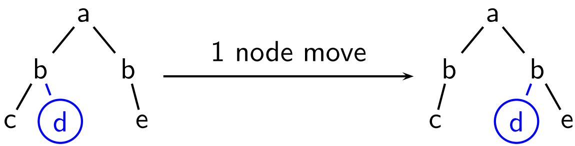
- **Single Node:** each node forms a base of size $q = 1$
- **Window:** $q \geq 2$ nodes of a window form a base



Goal:		bases must change	
✘ **Single Node:**	c, d, e	no bases change	c, d, e
Window:	cd, c*, d*, dc, *c, *d, e*, . . .	33% bases change	c*, c*, **, *c, *c, **, de, . . .

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	
✘ **Single Node:**	c, d, e	no bases change	c, d, e
✓ **Window:**	cd, c*, d*, dc, *c, *d, e*, . . .	33% bases change	c*, c*, **, *c, *c, **, de, . . .

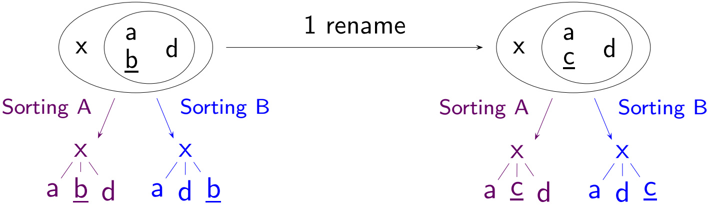
Windowed *pq*-grams detect node moves.

Illustration: Robustness to Different Sortings

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

Illustration: Robustness to Different Sortings

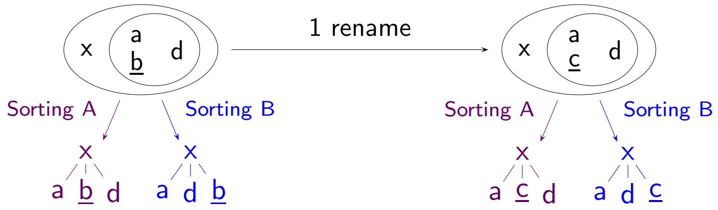
- **Consecutive siblings** form a base (no permutation)



Goal: Same number of bases change for both sortings.

Illustration: Robustness to Different Sortings

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

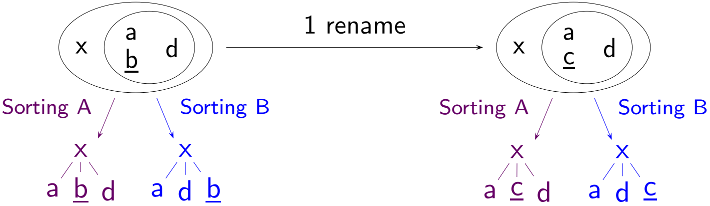


Goal: Same number of bases change for both sortings.

Consecutive:	Sort A	ab bc	100% bases change	ac cd
	Sort B	ad db	50% bases change	ad dc

Illustration: Robustness to Different Sortings

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

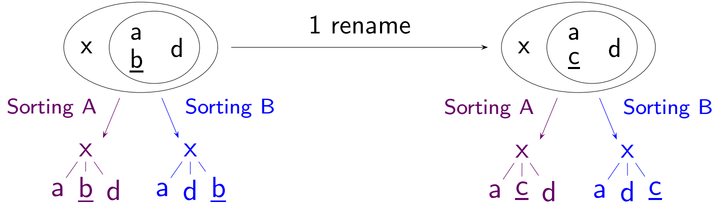


Goal: Same number of bases change for both sortings.

✘ **Consecutive:**	Sort A	ab bc	100% bases change	ac cd
	Sort B	ad db	50% bases change	ad dc

Illustration: Robustness to Different Sortings

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

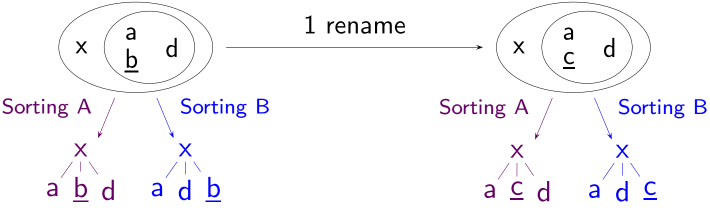


Goal: Same number of bases change for both sortings.

✘ **Consecutive:**	Sort A	ab bc	100% bases change	ac cd
	Sort B	ad db	50% bases change	ad dc
Window:	Sort A	ad ab db. . .	33% bases change	ad ac dc. . .
	Sort B	ad ab db. . .	33% bases change	ad ac dc. . .

Illustration: Robustness to Different Sortings

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



Goal: Same number of bases change for both sortings.

✘ **Consecutive:**	Sort A	ab bc	100% bases change	ac cd
	Sort B	ad db	50% bases change	ad dc
✓ **Window:**	Sort A	ad ab db. . .	33% bases change	ad ac dc. . .
	Sort B	ad ab db. . .	33% bases change	ad ac dc. . .

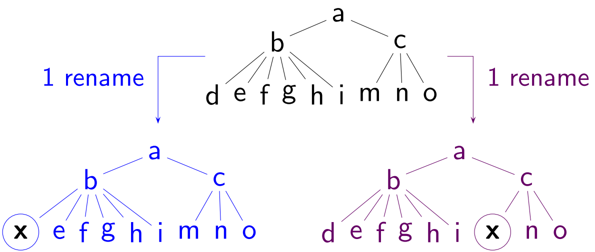
Windowed *pq*-grams: Robust to different sortings.

Illustration: Balancing the Node Weight

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

Illustration: Balancing the Node Weight

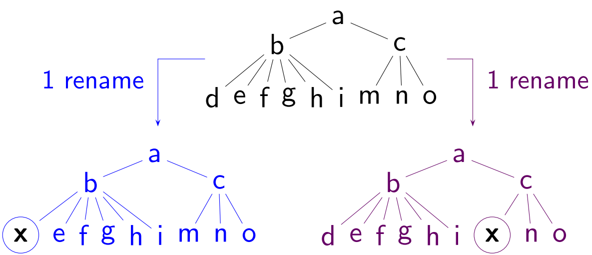
- **Permutations**: all permutations of size q form a base



Goal: Same number of bases change for both renames.

Illustration: Balancing the Node Weight

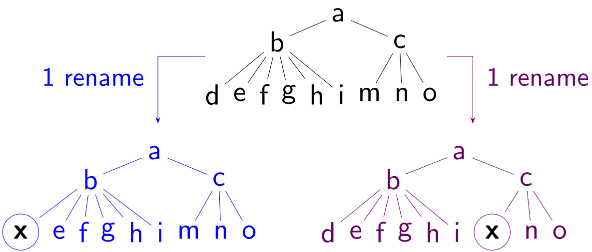
- **Permutations**: all permutations of size q form a base



Goal:	Same number of bases change for both renames.	
Permutations:	60/137 bases change	6/137 bases change

Illustration: Balancing the Node Weight

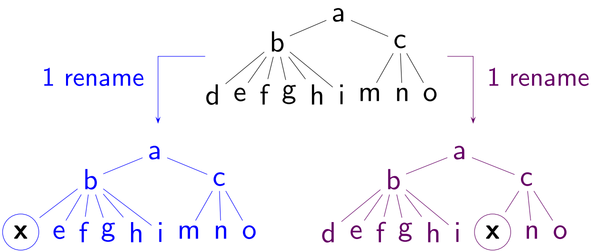
- **Permutations**: all permutations of size q form a base



Goal:	Same number of bases change for both renames.	
✗ **Permutations:**	60/137 bases change	6/137 bases change

Illustration: Balancing the Node Weight

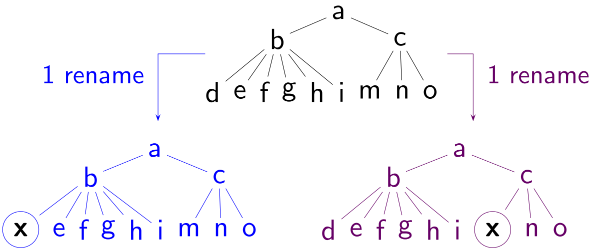
- **Permutations**: all permutations of size q form a base
- **Window:** only permutations within window form a base



Goal:	Same number of bases change for both renames.	
✘ **Permutations:**	60/137 bases change	6/137 bases change
Window:	12/51 bases change	12/51 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.	
✘ **Permutations:**	60/137 bases change	6/137 bases change
✓ **Window:**	12/51 bases change	12/51 bases change

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

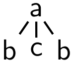
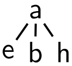
4. Experiments

5. Related Work

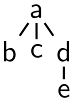
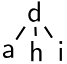
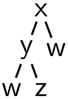
6. Conclusion and Future Work

Approximate Join

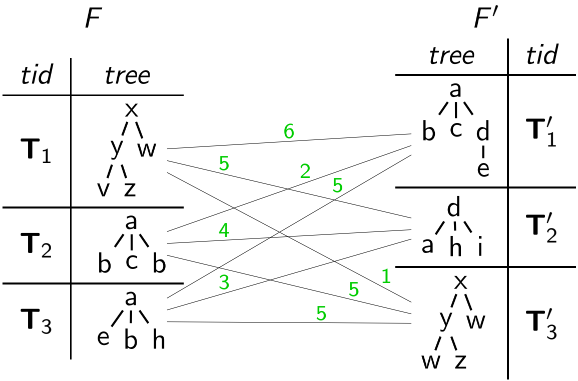
F

tid	*tree*
$\mathbf{T}_1$	
$\mathbf{T}_2$	
$\mathbf{T}_3$	

F'

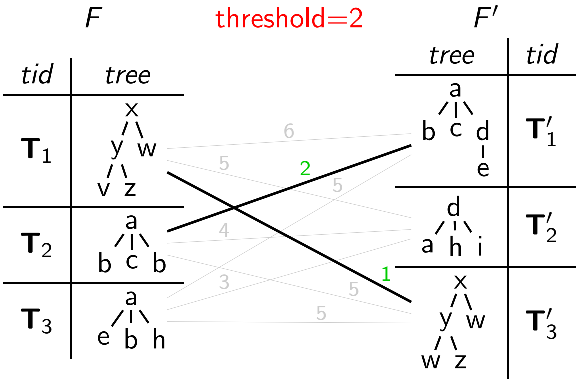
tree	*tid*
	$\mathbf{T}'_1$
	$\mathbf{T}'_2$
	$\mathbf{T}'_3$

Approximate Join



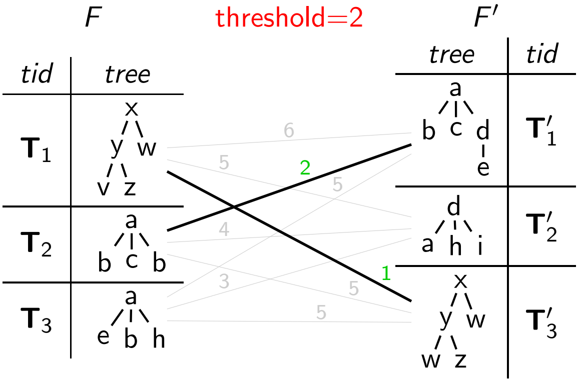
- **Simple approach:** distance join
  1. compute distance between all pairs of trees

Approximate Join



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

Approximate Join



- **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

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

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

Reducing a Distance Join to an Equality Join

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

$\{1,7\}_a$ $\{1,7\}_d$

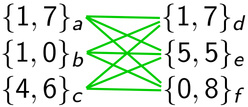
$\{1,0\}_b$ $\{5,5\}_e$

$\{4,6\}_c$ $\{0,8\}_f$

$|a \cap d| = 2 \quad |a \cap e| = 0 \quad |a \cap f| = 0$

Reducing a Distance Join to an Equality Join

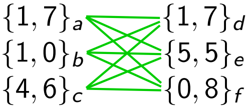
- **Distance join** between trees: N^2 intersections between integer bags



$$|a \cap d| = 2 \quad |a \cap e| = 0 \quad |a \cap f| = 0$$
$$|b \cap d| = 1 \quad |b \cap e| = 0 \quad |b \cap f| = 1$$
$$|c \cap d| = 0 \quad |c \cap e| = 0 \quad |c \cap f| = 0$$

Reducing a Distance Join to an Equality Join

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



$|a \cap d| = 2 \quad |a \cap e| = 0 \quad |a \cap f| = 0$

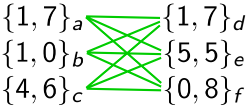
$|b \cap d| = 1 \quad |b \cap e| = 0 \quad |b \cap f| = 1$

$|c \cap d| = 0 \quad |c \cap e| = 0 \quad |c \cap f| = 0$

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

Reducing a Distance Join to an Equality Join

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



$|a \cap d| = 2 \quad |a \cap e| = 0 \quad |a \cap f| = 0$

$|b \cap d| = 1 \quad |b \cap e| = 0 \quad |b \cap f| = 1$

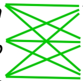
$|c \cap d| = 0 \quad |c \cap e| = 0 \quad |c \cap f| = 0$

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

     $\{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\}$ $\quad$ $\{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}$

Reducing a Distance Join to an Equality Join

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

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

$$|a \cap d| = 2 \quad |a \cap e| = 0 \quad |a \cap f| = 0$$
$$|b \cap d| = 1 \quad |b \cap e| = 0 \quad |b \cap f| = 1$$
$$|c \cap d| = 0 \quad |c \cap e| = 0 \quad |c \cap f| = 0$$

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

1. union

   $\{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\}$ $\qquad$ $\{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}$

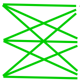
2. sort

0_b	0_f
1_a	1_d
1_b	5_e
4_c	5_e
6_c	7_d
7_a	8_f

Reducing a Distance Join to an Equality Join

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

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



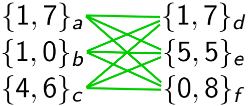
$\|a \cap d\| = 2$	$\|a \cap e\| = 0$	$\|a \cap f\| = 0$
$\|b \cap d\| = 1$	$\|b \cap e\| = 0$	$\|b \cap f\| = 1$
$\|c \cap d\| = 0$	$\|c \cap e\| = 0$	$\|c \cap f\| = 0$

- **Optimized *pq*-gram join**: empty intersections are never computed!
  1. union
     $\{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\}$ $\{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}$
  2. sort
  3. merge-join

0_b	0_f
1_a	1_d
1_b	5_e
4_c	5_e
6_c	7_d
7_a	8_f

Reducing a Distance Join to an Equality Join

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



$|a \cap d| = 2 \quad |a \cap e| = 0 \quad |a \cap f| = 0$

$|b \cap d| = 1 \quad |b \cap e| = 0 \quad |b \cap f| = 1$

$|c \cap d| = 0 \quad |c \cap e| = 0 \quad |c \cap f| = 0$

- **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	0_f
1_a	1_d
1_b	5_e
4_c	5_e
6_c	7_d
7_a	8_f

$|b \cap f|$

Reducing a Distance Join to an Equality Join

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

$\{1,7\}_a \quad \{1,7\}_d$

$\{1,0\}_b \quad \{5,5\}_e$

$\{4,6\}_c \quad \{0,8\}_f$

$\lvert a \cap d\rvert = 2$	$\lvert a \cap e\rvert = 0$	$\lvert a \cap f\rvert = 0$
$\lvert b \cap d\rvert = 1$	$\lvert b \cap e\rvert = 0$	$\lvert b \cap f\rvert = 1$
$\lvert c \cap d\rvert = 0$	$\lvert c \cap e\rvert = 0$	$\lvert c \cap f\rvert = 0$

- **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 — 0_f
   1_a — 1_d
   1_b $\quad$ 5_e
   4_c $\quad$ 5_e
   6_c $\quad$ 7_d
   7_a $\quad$ 8_f

$\lvert b \cap f\rvert$ |

$\lvert a \cap d\rvert$ |

Reducing a Distance Join to an Equality Join

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

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

$|a \cap d| = 2$ $|a \cap e| = 0$ $|a \cap f| = 0$
$|b \cap d| = 1$ $|b \cap e| = 0$ $|b \cap f| = 1$
$|c \cap d| = 0$ $|c \cap e| = 0$ $|c \cap f| = 0$

- **Optimized *pq*-gram join**: empty intersections are never computed!
  1. union
     $\{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\}$ $\{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}$
  2. sort
  3. merge-join
     0_b 0_f
     1_a 1_d
     1_b 5_e
     4_c 5_e
     6_c 7_d
     7_a 8_f

     $|b \cap f|$
     $|a \cap d|$
     $|b \cap d|$

Reducing a Distance Join to an Equality Join

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

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

$|a \cap d| = 2$ $|a \cap e| = 0$ $|a \cap f| = 0$
$|b \cap d| = 1$ $|b \cap e| = 0$ $|b \cap f| = 1$
$|c \cap d| = 0$ $|c \cap e| = 0$ $|c \cap f| = 0$

- **Optimized *pq*-gram join**: empty intersections are never computed!
  1. union
     $\{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\}$ $\{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}$
  2. sort
  3. merge-join

0_b — 0_f
1_a — 1_d
1_b 5_e
4_c 5_e
6_c 7_d
7_a 8_f

$|b \cap f|$ I
$|a \cap d|$ II
$|b \cap d|$ I

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

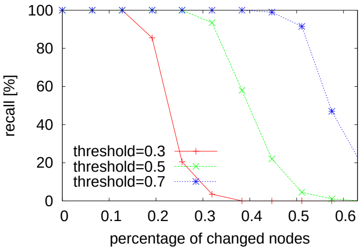
4 Experiments

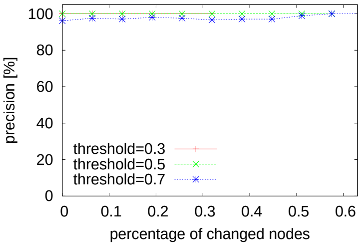
5 Related Work

6 Conclusion and Future Work

Effectiveness of the Windowed *pq*-Gram Join

Effectiveness of the Windowed *pq*-Gram Join



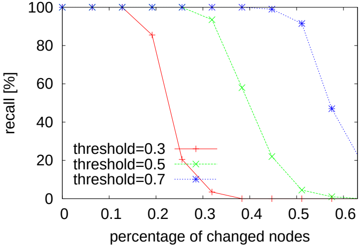


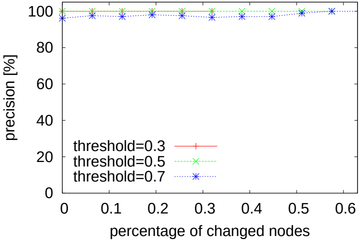
Experiment: match **DBLP** articles

- 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

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:

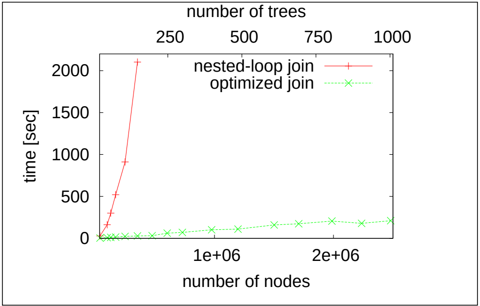
- **DBLP:** articles depth 1.9, 15 nodes (max 1494 nodes)
- **SwissProt:** protein descriptions depth 3.5, 104 nodes (max 2640 nodes)
- **Treebank:** tagged English sentences depth 6.9 (max depth 30), 43 nodes

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**

Conclusion and Future Work

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

Conclusion and Future Work

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

- Sudarshan S. Chawathe and Hector Garcia-Molina.
  Meaningful change detection in structured data.
  In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 26–37, Tucson, Arizona, United States, May 1997. ACM Press.

- Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava.
  Approximate string joins in a database (almost) for free.
  In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 491–500, Roma, Italy, September 2001. Morgan Kaufmann Publishers Inc.

- Karin Kailing, Hans-Peter Kriegel, Stefan Schönauer, and Thomas Seidl.
  Efficient similarity search for hierarchical data in large databases.
  In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, volume 2992 of *Lecture Notes in Computer*

Science, pages 676–693, Heraklion, Crete, Greece, March 2004. Springer.

Kaizhong Zhang, Richard Statman, and Dennis Shasha.
On the editing distance between unordered labeled trees.
Information Processing Letters, 42(3):133–139, 1992.