

# Similarity Search

## Windowed $pq$ -Grams for Unordered Trees

Nikolaus Augsten

nikolaus.augsten@sbg.ac.at  
Department of Computer Sciences  
University of Salzburg



<http://dbresearch.uni-salzburg.at>

WS 2019/20

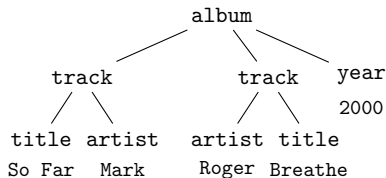
Version February 10, 2020

# Outline

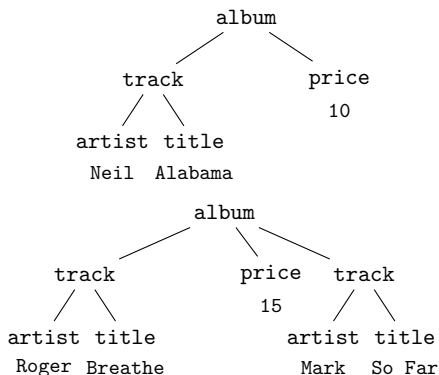
- 1 Last Week
  - The  $pq$ -Gram Distance
  - Fanout Weighting and Lower Bound
  - Experiments
- 2 Windowed  $pq$ -Grams for Unordered Trees
  - **Motivation**
  - Windowed  $pq$ -Grams for Data-Centric XML
  - Efficient Similarity Joins with Windowed  $pq$ -Grams
  - Experiments
- 3 Conclusion

# Similarity Join on Music CDs

## Song Lyric Store

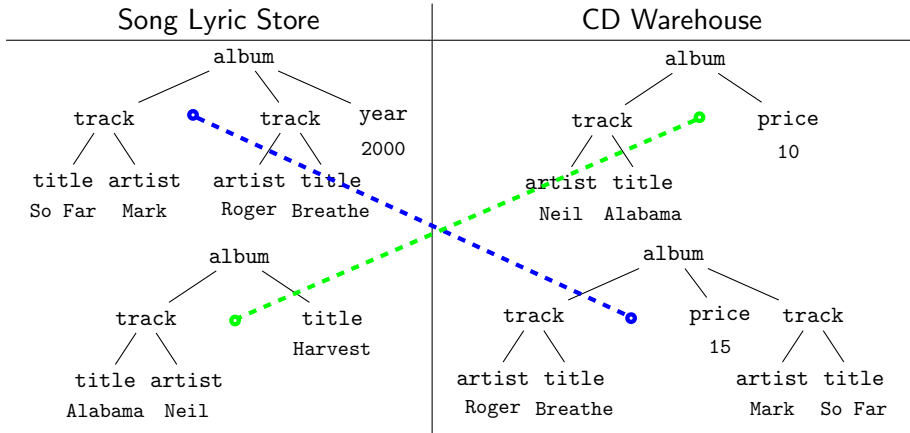


## CD Warehouse



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

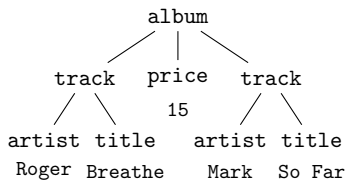
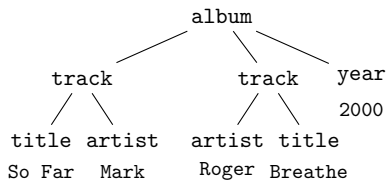
# Similarity Join on Music CDs



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

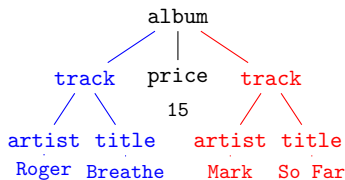
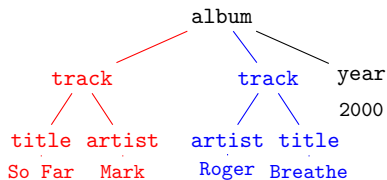
How similar are two XML items?

# 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.
- **Problem:** permuted subtrees are deleted/re-inserted node by node

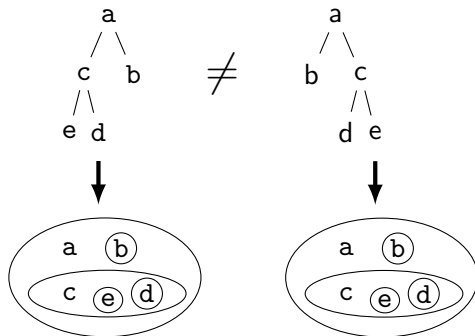
# How Similar Are these XMLs?



- 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



- Edit distance between unordered trees: NP-complete  
→ all sibling permutations must be considered!

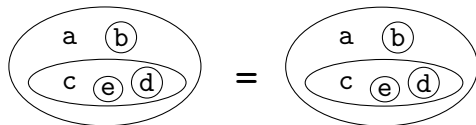
# Ordered vs. Unordered Trees

**Ordered Trees**  
sibling order matters



↓ ignore order ↓

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

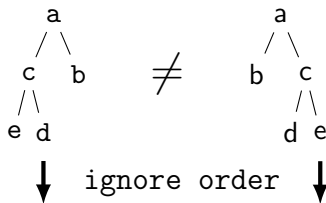


- Edit distance between unordered trees: NP-complete  
→ all sibling permutations must be considered!

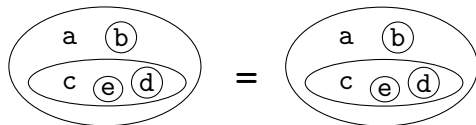


# Ordered vs. Unordered Trees

**Ordered Trees**  
sibling order matters



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



- **Edit distance** between unordered trees: **NP-complete**  
→ all sibling permutations must be considered!

# Problem Definition

Find an effective distance for unordered labeled trees that is efficient for similarity joins.

Naive approaches that fail:

- unordered tree edit distance: NP-complete
- allow subtree move: NP-hard
- compute minimum distance between all permutations: runtime  $O(n!)$ , error  $O(n)$
- sort by label and use ordered tree edit distance: error  $O(n)$

# Problem Definition

Find an **effective distance** for **unordered labeled trees** that is **efficient** for **similarity joins**.

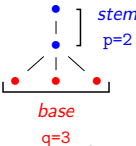
**Naive approaches** that fail:

- unordered tree edit distance: NP-complete
- allow subtree move: NP-hard
- compute minimum distance between all permutations: runtime  $O(n!)$ , error  $O(n)$
- sort by label and use ordered tree edit distance: error  $O(n)$

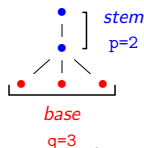
# Outline

- 1 Last Week
  - The  $pq$ -Gram Distance
  - Fanout Weighting and Lower Bound
  - Experiments
- 2 Windowed  $pq$ -Grams for Unordered Trees
  - Motivation
  - Windowed  $pq$ -Grams for Data-Centric XML
  - Efficient Similarity Joins with Windowed  $pq$ -Grams
  - Experiments
- 3 Conclusion

# Solution: Windowed $pq$ -Grams [ABDG08]

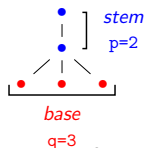
- Windowed  $pq$ -Gram: small subtree with **stem** and **base**

- 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

# Solution: Windowed $pq$ -Grams [ABDG08]



- **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

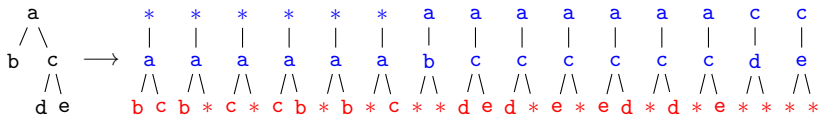
# Solution: Windowed $pq$ -Grams [ABDG08]



- **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:



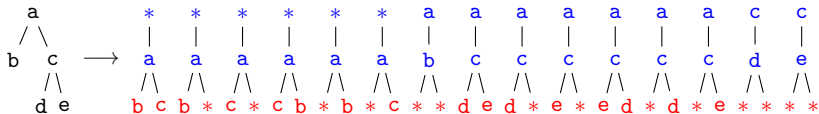
- Hashing: map  $pq$ -gram to integer:

- Windowed  $pq$ -gram index: bag of hashed  $pq$ -grams

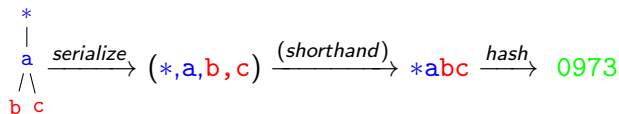


# Implementation of Windowed $pq$ -Grams

- Set of windowed  $pq$ -grams:



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



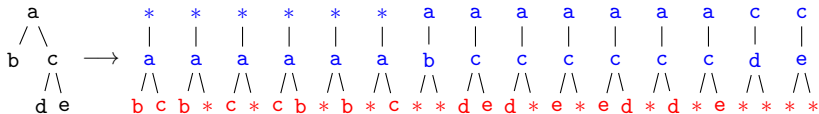
label $l$	$h(l)$
*	0
a	9
b	7
c	3
...	...

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

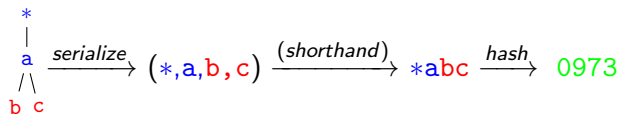
- Windowed  $pq$ -gram index: bag of hashed  $pq$ -grams

# Implementation of Windowed pq-Grams

- Set of windowed pq-grams:



- Hashing: map pq-gram to integer:



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

Note: labels may be strings of arbitrary length!

- Windowed pq-gram index: bag of hashed pq-grams

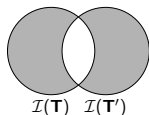
$$\mathcal{I}(\mathbf{T}) = \{0973, 0970, 0930, 0937, 0907, 0903, 9700, 9316, 9310, 9360, 9361, 9301, 9306, 3100, 3600\}$$

Tree is represented by a bag of integers!

# The Windowed $pq$ -Gram Distance

- The windowed  $pq$ -gram distance between two trees,  $\mathbf{T}$  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:  $x = y \Rightarrow \text{dist}^{pq}(x, y) = 0$
- ✓ symmetry:  $\text{dist}^{pq}(x, y) = \text{dist}^{pq}(y, x)$
- ✓ triangle inequality:  $\text{dist}^{pq}(x, z) \leq \text{dist}^{pq}(x, y) + \text{dist}^{pq}(y, z)$

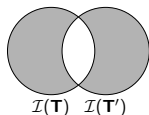
- Different trees may be at distance zero:
- Runtime for the distance computation is  $O(n \log n)$ .



# 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:  $x = y \Rightarrow \text{dist}^{pq}(x, y) = 0$
- ✓ symmetry:  $\text{dist}^{pq}(x, y) = \text{dist}^{pq}(y, x)$
- ✓ triangle inequality:  $\text{dist}^{pq}(x, z) \leq \text{dist}^{pq}(x, y) + \text{dist}^{pq}(y, z)$

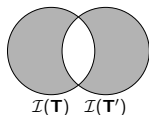


- Different trees may be at distance zero:
- Runtime for the distance computation is  $O(n \log n)$ .

# The Windowed $pq$ -Gram Distance

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

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

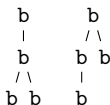


- Pseudo-metric** properties hold:

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

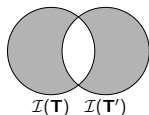
- Runtime for the distance computation is  $O(n \log n)$ .



# The Windowed $pq$ -Gram Distance

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

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

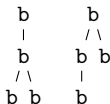


- Pseudo-metric properties hold:

- ✓ self-identity:  $x = y \not\Rightarrow \text{dist}^{pq}(x, y) = 0$
- ✓ 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:

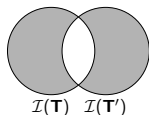
- Runtime for the distance computation is  $O(n \log n)$ .



# The Windowed $pq$ -Gram Distance

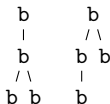
- The **windowed  $pq$ -gram distance** between two trees,  $\mathbf{T}$  and  $\mathbf{T}'$ :

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



- Pseudo-metric** properties hold:

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

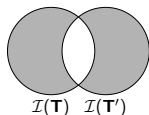


- Different trees may be at distance zero:
- Runtime for the distance computation is  $O(n \log n)$ .

# The Windowed $pq$ -Gram Distance

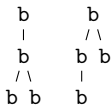
- The **windowed  $pq$ -gram distance** between two trees,  $\mathbf{T}$  and  $\mathbf{T}'$ :

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



- Pseudo-metric** properties hold:

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



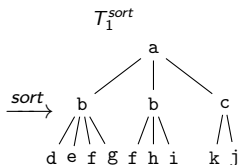
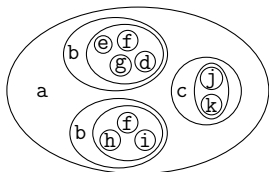
- Different trees may be at distance zero:
- Runtime** for the distance computation is  $O(n \log n)$ .



# Sorting the Tree?

- Idea:

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



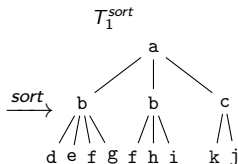
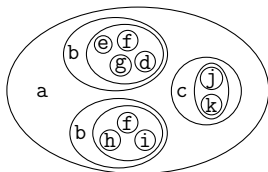
✗ Edit distance: tree sorting does not work

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

# Sorting the Tree?

- Idea:

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

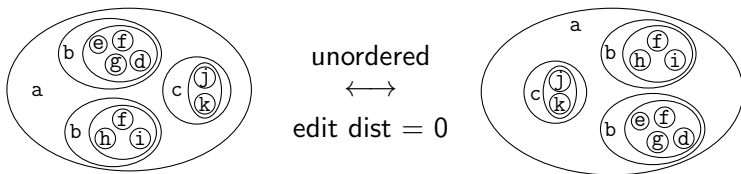


✗ Edit distance: tree sorting does not work

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

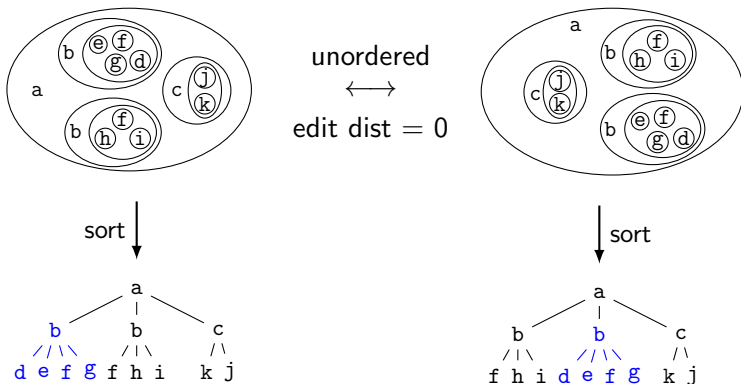
# ✗ Edit Distance: Tree Sorting Does Not Work

## 1. Non-unique sorting:



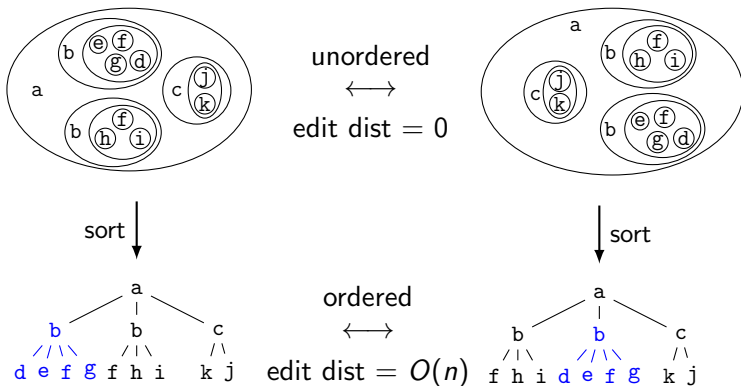
# ✗ Edit Distance: Tree Sorting Does Not Work

## 1. Non-unique sorting:



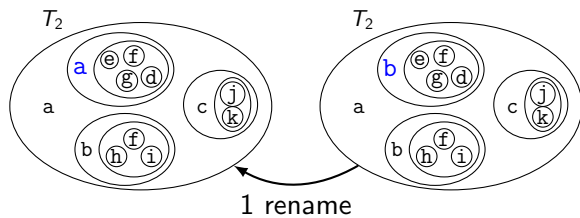
# ✗ Edit Distance: Tree Sorting Does Not Work

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



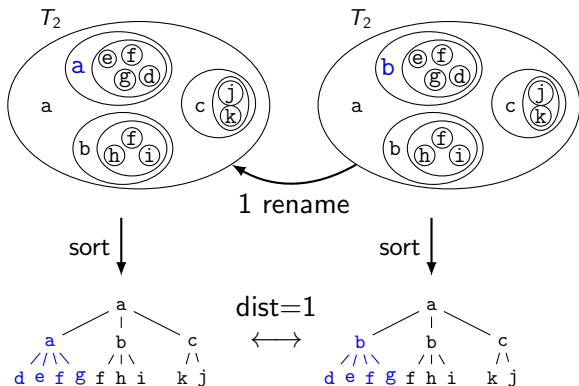
# ✗ Edit Distance: Tree Sorting Does Not Work

## 2. Node renaming:



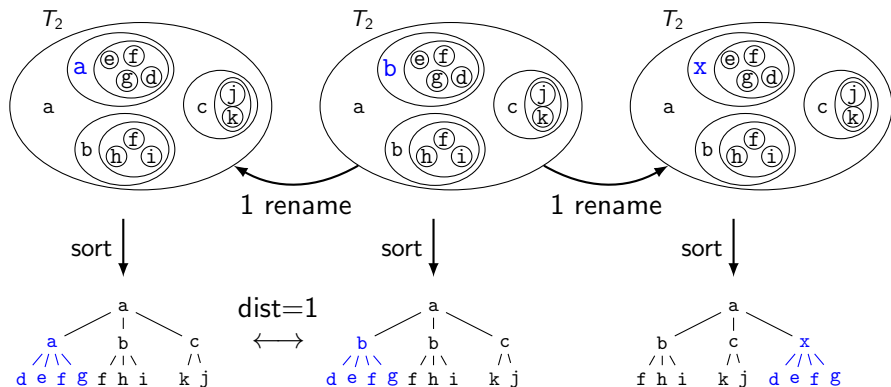
# ✗ Edit Distance: Tree Sorting Does Not Work

## 2. Node renaming:



# ✗ Edit Distance: Tree Sorting Does Not Work

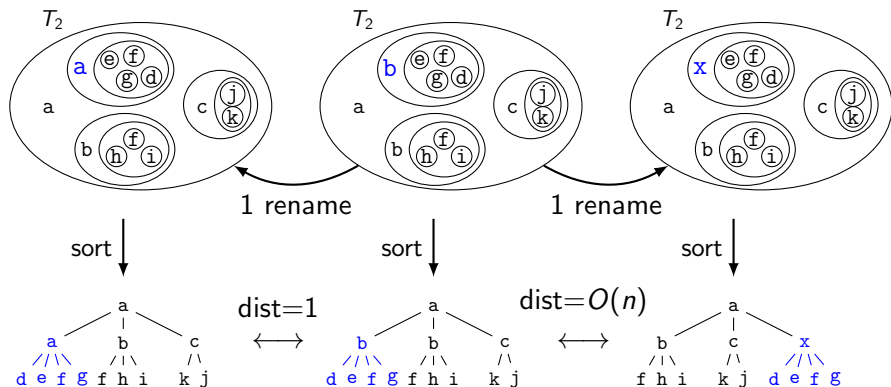
## 2. Node renaming:





# ✗ Edit Distance: Tree Sorting Does Not Work

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



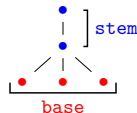
# ✓ Windowed $pq$ -Grams: Tree Sorting Works OK

## 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



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

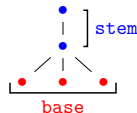
# ✓ Windowed $pq$ -Grams: Tree Sorting Works OK

## 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):

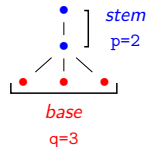
- $pq$ -grams consist of a **stem** and a **base**
- **stems** are invariant to the sibling order
- **bases**: only the  $O(k)$   $pq$ -grams with the reordered nodes in the bases change



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

# How To Form Bases?

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



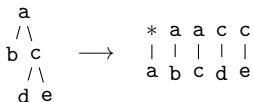
- Stems: ignore sibling order



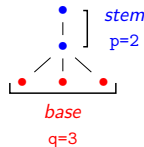
- Bases: do not ignore sibling order!

# 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

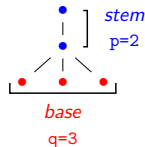


- **Bases**: do not ignore sibling order!

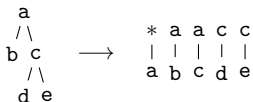


# How To Form Bases?

- **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
- 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

# Requirements for Bases

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



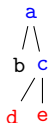
# Solution: Windowed $pq$ -Gram Bases

---

**Algorithm 1: Form bases** from a sorted (non-empty) sibling sequence

---

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



# Solution: Windowed $pq$ -Gram Bases

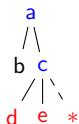
---

**Algorithm 1: Form bases** from a sorted (non-empty) sibling sequence

---

- 1 if *sibling sequence*  $<$  *window* then extend with dummy nodes;
- 2

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



# Solution: Windowed $pq$ -Gram Bases

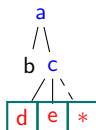
---

**Algorithm 1:** Form bases from a sorted (non-empty) sibling sequence

---

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

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



# Solution: Windowed $pq$ -Gram Bases

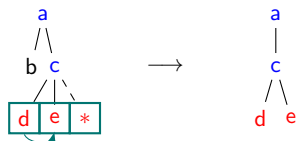
---

**Algorithm 1: Form bases** from a sorted (non-empty) sibling sequence

---

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

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

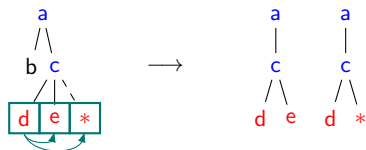


# Solution: Windowed $pq$ -Gram Bases

**Algorithm 1: Form bases** from a sorted (non-empty) sibling sequence

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

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



# Solution: Windowed $pq$ -Gram Bases

**Algorithm 1: Form bases** from a sorted (non-empty) sibling sequence

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

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

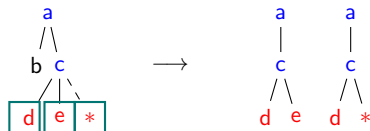


# Solution: Windowed $pq$ -Gram Bases

**Algorithm 1: Form bases** from a sorted (non-empty) sibling sequence

- 1 if *sibling sequence* < *window* then extend with dummy nodes;
- 2
- 3 initialize *window*: start with leftmost node;
- 4 repeat
- 5     form bases in window: all  $q$ -permutations that contain start node;
- 6     shift *window* to the right by one node;
- 7     if *window* extends the right border then wrap *window*;
- 8
- 9 until *processed all window positions*

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

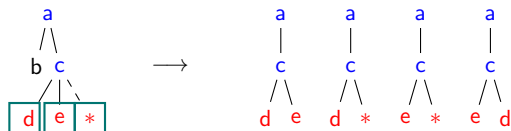


# Solution: Windowed $pq$ -Gram Bases

**Algorithm 1: Form bases** from a sorted (non-empty) sibling sequence

- 1 if *sibling sequence* < *window* then extend with dummy nodes;
- 2
- 3 initialize *window*: start with leftmost node;
- 4 repeat
- 5     form bases in window: all  $q$ -permutations that contain start node;
- 6     shift *window* to the right by one node;
- 7     if *window extends the right border* then wrap *window*;
- 8
- 9 until *processed all window positions*

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



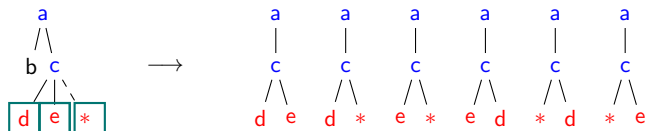


# Solution: Windowed pq-Gram Bases

**Algorithm 1: Form bases** from a sorted (non-empty) sibling sequence

- 1 if *sibling sequence* < *window* then extend with dummy nodes;
- 2
- 3 initialize *window*: start with leftmost node;
- 4 repeat
- 5     form bases in window: all  $q$ -permutations that contain start node;
- 6     shift *window* to the right by one node;
- 7     if *window* extends the right border then wrap *window*;
- 8
- 9 until *processed all window positions*

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

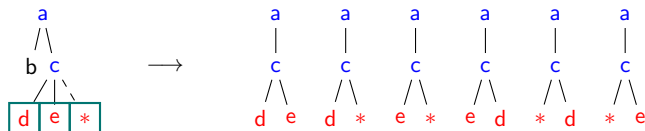


# Solution: Windowed pq-Gram Bases

**Algorithm 1: Form bases** from a sorted (non-empty) sibling sequence

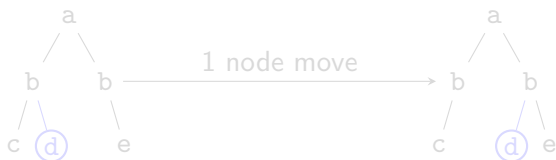
- 1 if *sibling sequence* < *window* then extend with dummy nodes;
- 2
- 3 initialize *window*: start with leftmost node;
- 4 repeat
- 5     form bases in window: all  $q$ -permutations that contain start node;
- 6     shift *window* to the right by one node;
- 7     if *window* extends the right border then wrap *window*;
- 8
- 9 until *processed all window positions*

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



# 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

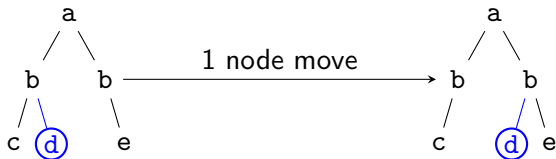


**Goal:**

bases must change

# 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



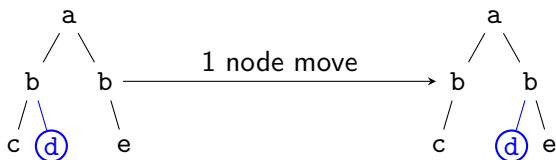
**Goal:**

bases must change

---

# 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:**

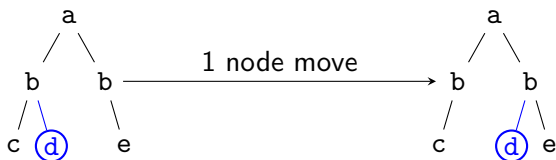
c, d, e

no bases change

c, d, e

# 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

**X Single Node:**

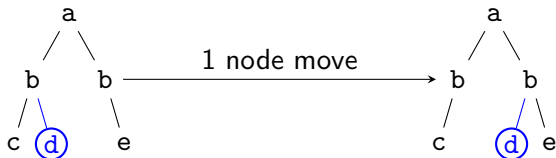
c, d, e

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:**

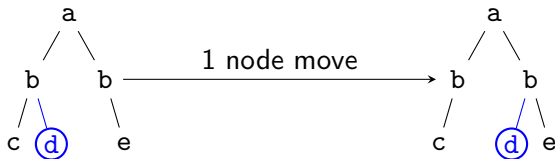
cd, c\*, d\*, dc,  
\*c, \*d, e\*, ...

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

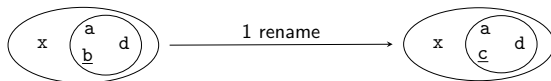
c\*, c\*, \*\*, \*c,  
\*c, \*\*, de, ...

Windowed  $pq$ -grams detect node moves.



# Illustration: Robustness to Different Sortings

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

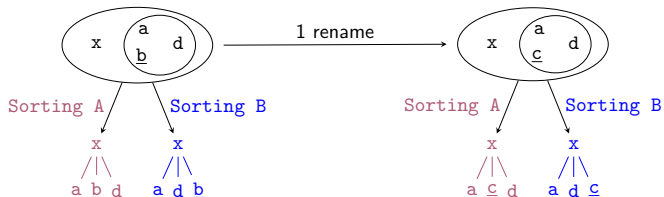


**Goal:** Same number of bases change for both sortings.

---

# Illustration: Robustness to Different Sortings

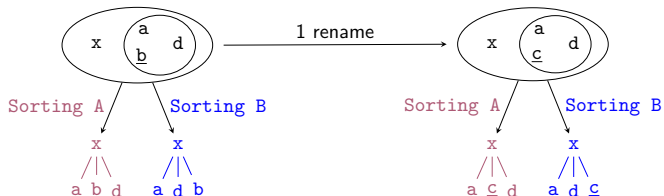
- Consecutive siblings form a base (no permutation)
- Window: all sibling permutations within the window form bases



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

# Illustration: Robustness to Different Sortings

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

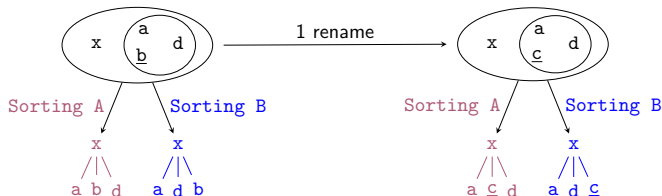


**Goal:** Same number of bases change for both sortings.

<b>Consecutive:</b>	Sort A	<u>a</u> <u>b</u> <u>b</u> <u>d</u>	100% bases change	<u>a</u> <u>c</u> <u>c</u> <u>d</u>
	Sort B	a <u>d</u> <u>d</u> <u>b</u>	50% bases change	a <u>d</u> <u>d</u> <u>c</u>

# Illustration: Robustness to Different Sortings

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

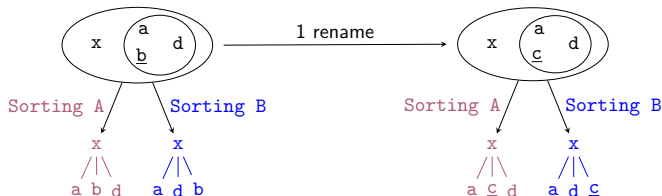


**Goal:** Same number of bases change for both sortings.

<b>✗ Consecutive:</b>	Sort A	<u>a</u> <u>b</u> <u>b</u> <u>d</u>	100% bases change	<u>a</u> <u>c</u> <u>c</u> <u>d</u>
	Sort B	a <u>d</u> <u>d</u> <u>b</u>	50% bases change	a <u>d</u> <u>d</u> <u>c</u>

# Illustration: Robustness to Different Sortings

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

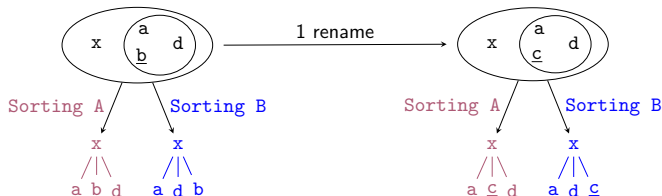


**Goal:** Same number of bases change for both sortings.

<b>✗ Consecutive:</b>	Sort A	a <u>b</u> <u>b</u> d	100% bases change	a <u>c</u> <u>c</u> d
	Sort B	a <u>d</u> <u>d</u> b	50% bases change	a <u>d</u> <u>d</u> c
<b>Window:</b>	Sort A	a <u>d</u> a <u>b</u> <u>b</u> d...	33% bases change	a <u>d</u> a <u>c</u> <u>c</u> d...
	Sort B	a <u>d</u> a <u>b</u> <u>b</u> d...	33% bases change	a <u>d</u> a <u>c</u> <u>c</u> d...

# Illustration: Robustness to Different Sortings

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



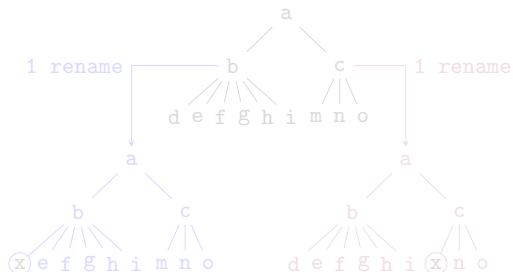
**Goal:** Same number of bases change for both sortings.

<b>✗ Consecutive:</b>	Sort A	<u>a</u> <u>b</u> <u>a</u> <u>d</u>	100% bases change	<u>a</u> <u>c</u> <u>a</u> <u>d</u>
	Sort B	<u>a</u> <u>d</u> <u>d</u> <u>b</u>	50% bases change	<u>a</u> <u>d</u> <u>d</u> <u>c</u>
<b>✓ Window:</b>	Sort A	<u>a</u> <u>d</u> <u>a</u> <u>b</u> <u>d</u> <u>b</u> ...	33% bases change	<u>a</u> <u>d</u> <u>a</u> <u>c</u> <u>d</u> <u>c</u> ...
	Sort B	<u>a</u> <u>d</u> <u>a</u> <u>b</u> <u>d</u> <u>b</u> ...	33% bases change	<u>a</u> <u>d</u> <u>a</u> <u>c</u> <u>d</u> <u>c</u> ...

Windowed pq-grams: Robust to different sortings.

# Illustration: Balancing the Node Weight

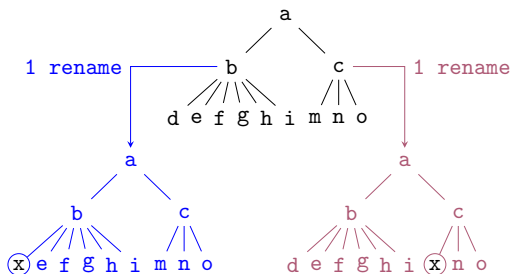
- **Permutations:** all permutations of size  $q$  form a base
- **Window:** only permutations within window form a base



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

# Illustration: Balancing the Node Weight

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

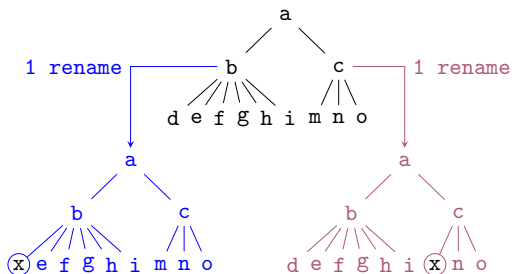


**Goal:** Same number of bases change for both renames.



# Illustration: Balancing the Node Weight

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

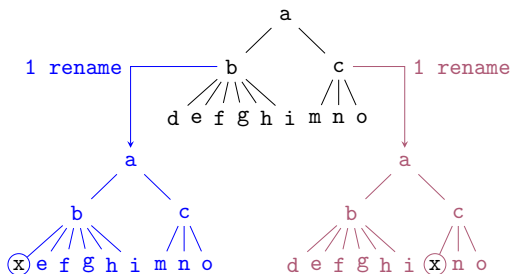


**Goal:** Same number of bases change for both renames.

**Permutations:** 60/137 bases change      6/137 bases change

# Illustration: Balancing the Node Weight

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

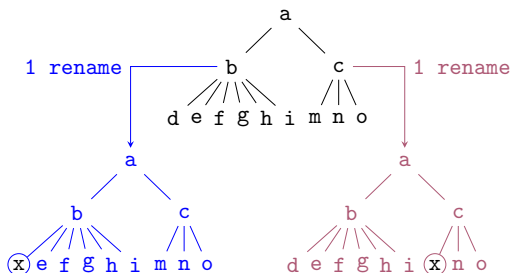


**Goal:** Same number of bases change for both renames.

**X 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



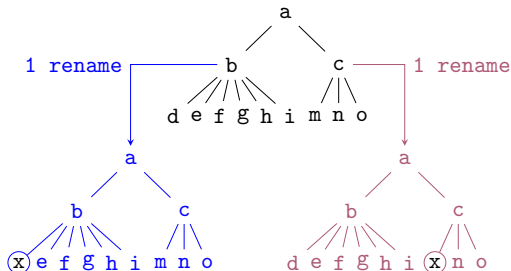
**Goal:** Same number of bases change for both renames.

**✗ 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

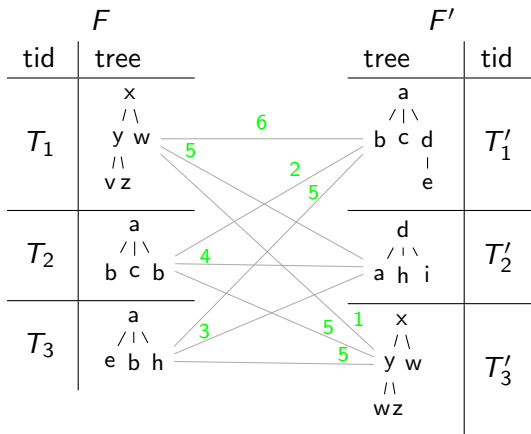
- 1 Last Week
  - The  $pq$ -Gram Distance
  - Fanout Weighting and Lower Bound
  - Experiments
- 2 Windowed  $pq$ -Grams for Unordered Trees
  - Motivation
  - Windowed  $pq$ -Grams for Data-Centric XML
  - **Efficient Similarity Joins with Windowed  $pq$ -Grams**
  - Experiments
- 3 Conclusion

# Similarity Join

$F$		$F'$	
tid	tree	tree	tid
$T_1$	<pre>       x      /\     y w      /\     v z           </pre>	<pre>       a      /\     b c d                     e           </pre>	$T'_1$
$T_2$	<pre>       a      /\     b c b           </pre>	<pre>       d      /\     a h i           </pre>	$T'_2$
$T_3$	<pre>       a      /\     e b h           </pre>	<pre>       x      /\     y w      /\     w z           </pre>	$T'_3$

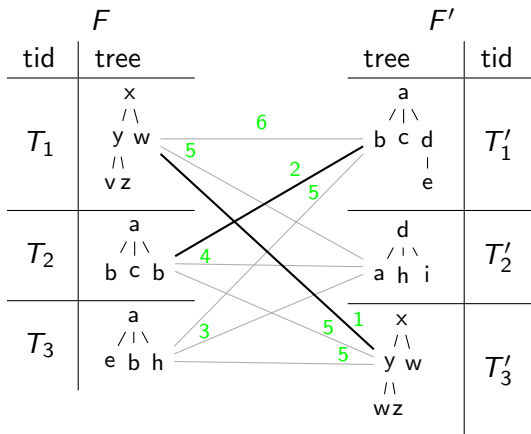
- Simple approach: distance join
  1. compute distance between all pairs of trees
  2. return document pairs within **threshold**
- Very expensive:  $N^2$  distance computations!

# Similarity 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!

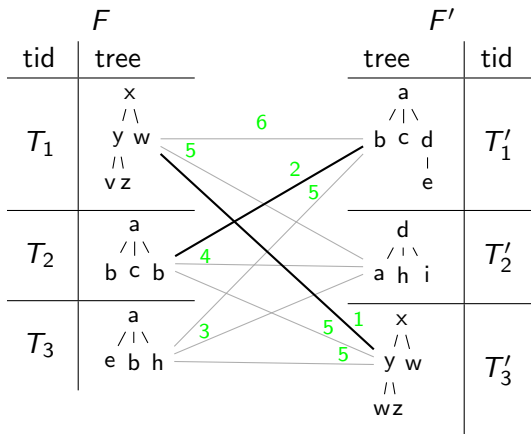
# Similarity Join



- **Simple approach:** distance join
  1. compute **distance** between all pairs of trees
  2. return document pairs within **threshold**
- **Very expensive:**  $N^2$  distance computations!



# Similarity 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
- **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

# 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

# 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

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

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

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

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

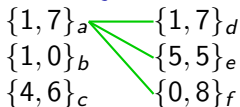
1. union

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

2. sort

# Reducing a Distance Join to an Equality Join

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



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

- 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

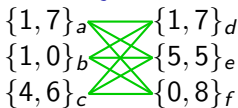
# Reducing a Distance Join to an Equality Join

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

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

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

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



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

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

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

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

1. union

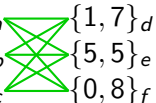
$$\{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\}$$

$$\{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}$$

2. sort

# 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$	$ a \cap e  = 0$	$ a \cap f  = 0$
$\{1, 0\}_b$		$\{5, 5\}_e$	$ b \cap d  = 1$	$ b \cap e  = 0$	$ b \cap f  = 1$
$\{4, 6\}_c$		$\{0, 8\}_f$	$ c \cap d  = 0$	$ c \cap e  = 0$	$ c \cap f  = 0$

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

- union

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

- sort



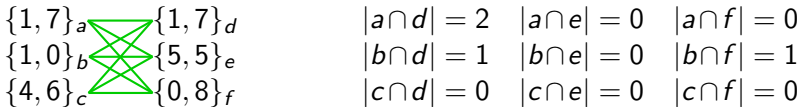
# Reducing a Distance Join to an Equality Join

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

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

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

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



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

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

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

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

1. union

$$\{1_a, 7_a, 1_b, 0_b, 4_c, 6_c\}$$

$$\{1_d, 7_d, 5_e, 5_e, 0_f, 8_f\}$$

2. sort

# 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$	$ a \cap e  = 0$	$ a \cap f  = 0$
$\{1, 0\}_b$	$\{5, 5\}_e$	$ b \cap d  = 1$	$ b \cap e  = 0$	$ b \cap f  = 1$
$\{4, 6\}_c$	$\{0, 8\}_f$	$ c \cap d  = 0$	$ c \cap e  = 0$	$ c \cap f  = 0$

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

- union

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

- 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

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

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

- union

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

- sort

- merge-join

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

$$|b \cap f| \quad |$$

# 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$	$ a \cap e  = 0$	$ a \cap f  = 0$
$\{1, 0\}_b$	$\{5, 5\}_e$	$ b \cap d  = 1$	$ b \cap e  = 0$	$ b \cap f  = 1$
$\{4, 6\}_c$	$\{0, 8\}_f$	$ c \cap d  = 0$	$ c \cap e  = 0$	$ c \cap f  = 0$

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

- union

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

- sort

- 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 $	

# 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$	$ a \cap e  = 0$	$ a \cap f  = 0$
$\{1, 0\}_b$	$\{5, 5\}_e$	$ b \cap d  = 1$	$ b \cap e  = 0$	$ b \cap f  = 1$
$\{4, 6\}_c$	$\{0, 8\}_f$	$ c \cap d  = 0$	$ c \cap e  = 0$	$ c \cap f  = 0$

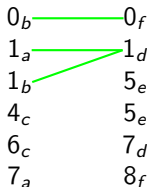
- Optimized pq-gram join: empty intersections are never computed!

- union

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

- sort

- merge-join



$$\begin{array}{l} |b \cap f| \quad | \\ |a \cap d| \quad || \\ |b \cap d| \quad | \end{array}$$

# 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$	$ a \cap e  = 0$	$ a \cap f  = 0$
$\{1, 0\}_b$		$\{5, 5\}_e$	$ b \cap d  = 1$	$ b \cap e  = 0$	$ b \cap f  = 1$
$\{4, 6\}_c$		$\{0, 8\}_f$	$ c \cap d  = 0$	$ c \cap e  = 0$	$ c \cap f  = 0$

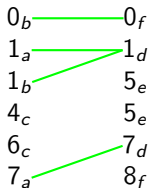
- Optimized pq-gram join: empty intersections are never computed!

- union

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

- sort

- merge-join

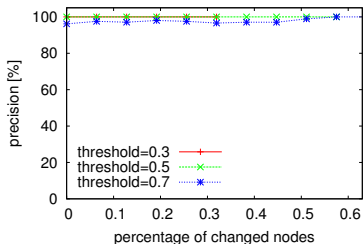
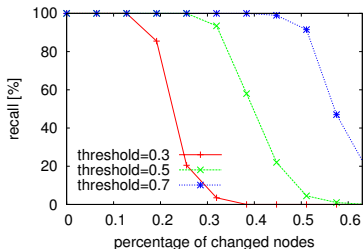


$$\begin{array}{l} |b \cap f| \quad | \\ |a \cap d| \quad || \\ |b \cap d| \quad | \end{array}$$

# Outline

- 1 Last Week
  - The  $pq$ -Gram Distance
  - Fanout Weighting and Lower Bound
  - Experiments
- 2 Windowed  $pq$ -Grams for Unordered Trees
  - Motivation
  - Windowed  $pq$ -Grams for Data-Centric XML
  - Efficient Similarity Joins with Windowed  $pq$ -Grams
  - Experiments
- 3 Conclusion

# Effectiveness of the Windowed $pq$ -Gram Join



## Experiment: match DBLP articles

- add noise to articles (missing elements and spelling mistakes)
- similarity join between original and noisy data
- measure precision and recall for different thresholds

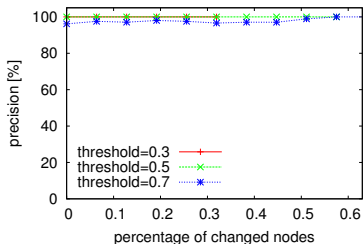
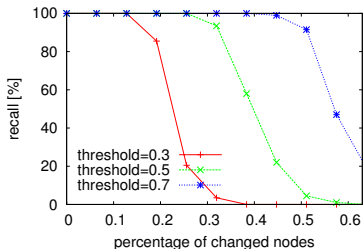
## Datasets:

- 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



# Effectiveness of the Windowed $pq$ -Gram Join



## Experiment: match **DBLP** articles

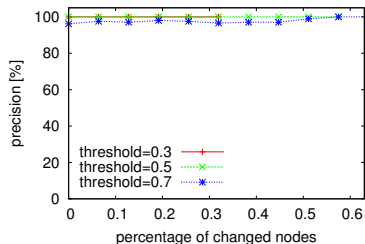
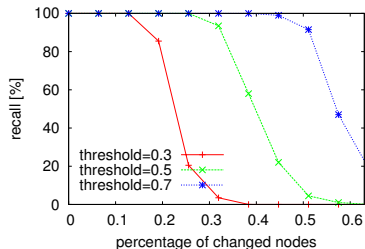
- add noise to articles (missing elements and spelling mistakes)
- similarity join between original and noisy data
- measure precision and recall for different thresholds

## 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**

# Effectiveness of the Windowed $pq$ -Gram Join



## Experiment: match **DBLP** articles

- add noise to articles (missing elements and spelling mistakes)
- similarity join between original and noisy data
- measure precision and recall for different thresholds

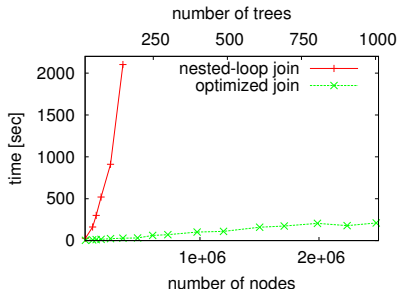
## Datasets:

- **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

Optimized  $pq$ -gram join: very efficient



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

# Distances between Unordered Trees

## Edit Distances between Unordered Trees

- [ZSS92]: proof for NP-completeness
- [KKSS04]: lower bound for a restricted edit distance
- [CGM97]:  $O(n^3)$  heuristics
- Our solution:  $O(n \log n)$  heuristics

## Similarity Join

- [GIJ<sup>+</sup>01]: efficient similarity join for strings

# Windowed pq-Grams

Windowed  $pq$ -grams for unordered trees:

- $O(n \log n)$  heuristics for NP-complete edit distance
- **Key problem:** all permutations must be considered
- **Heuristics:** 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 similarity join:** reduces distance join to equality join



Nikolaus Augsten, Michael Böhlen, Curtis Dyreson, and Johann Gamper.

Approximate joins for data-centric XML.

In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 814–823, Cancún, Mexico, April 2008. IEEE Computer Society.



Nikolaus Augsten, Michael Böhlen, and Johann Gamper.

Approximate matching of hierarchical data using *pq*-grams.

In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 301–312, Trondheim, Norway, September 2005. ACM Press.



Nikolaus Augsten, Michael Böhlen, and Johann Gamper.

The *pq*-gram distance between ordered labeled trees.

*ACM Transactions on Database Systems (TODS)*, 35(1):1–36, 2010.



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.