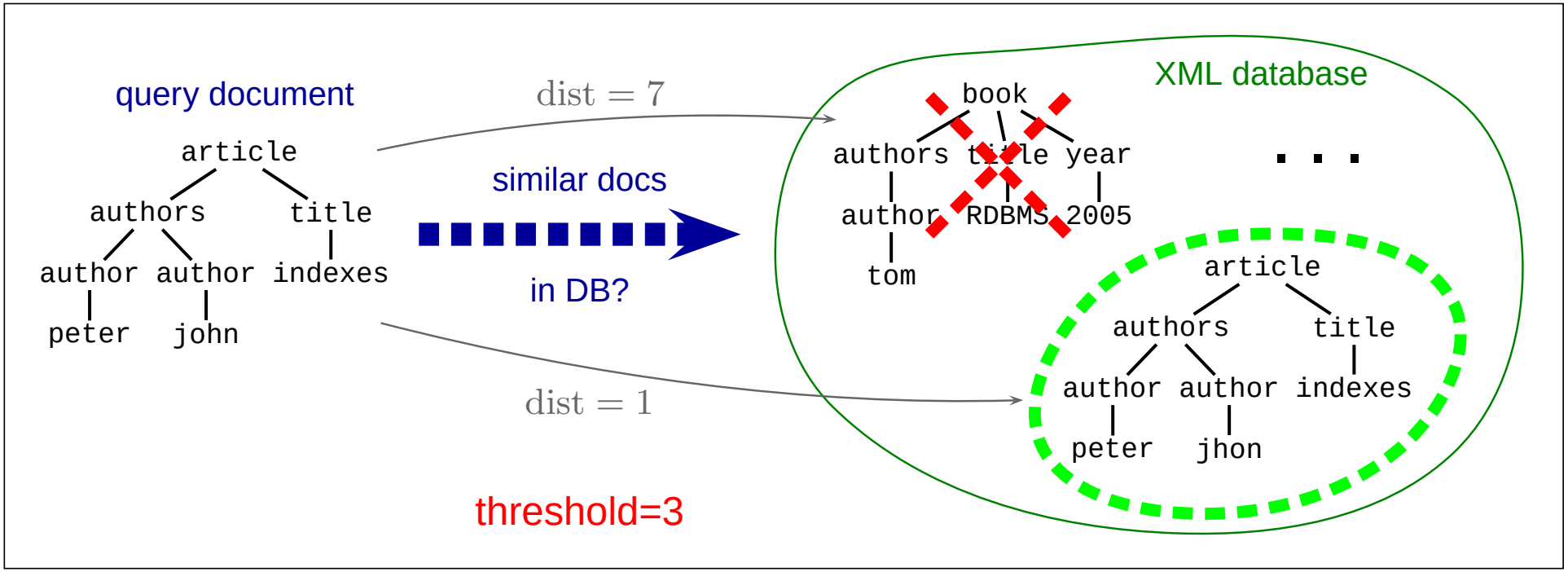
An Incrementally Maintainable Index for Approximate Lookups in Hierarchical Data

Nikolaus Augsten[a], Michael Böhlen, Johann Gamper
DIS - Center for Database and Information Systems
Free University of Bozen-Bolzano, Italy
www.inf.unibz.it

[a] Supported by the Municipality of Bozen-Bolzano.

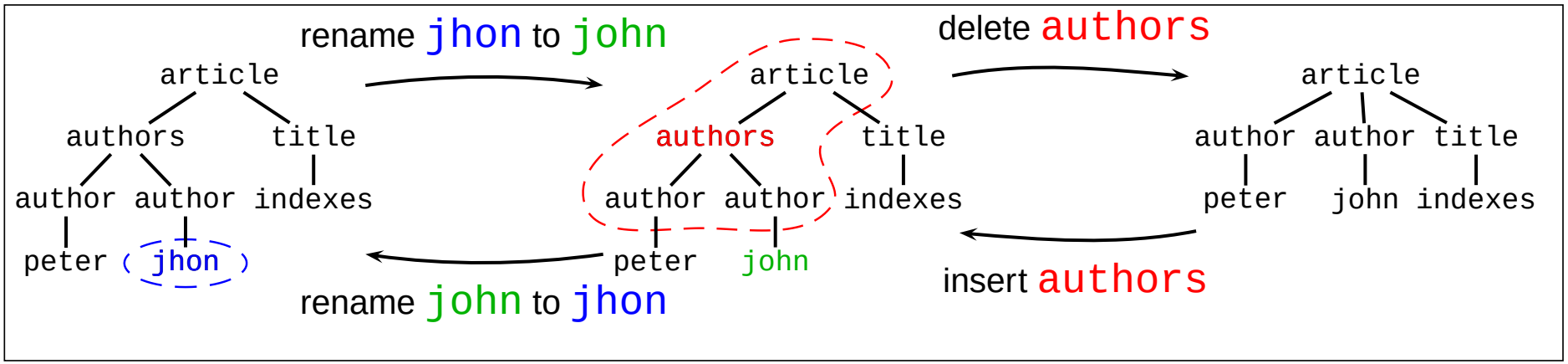
Approximate Lookups in XML Database



- **Simple approach:**
  1. compute **distance** between the query document and each document in DB
  2. choose documents within threshold
- Very **expensive**
  - scan whole DB
  - compute distance to each XML document
- **Index** for approximate lookups

Need for Incremental Updates

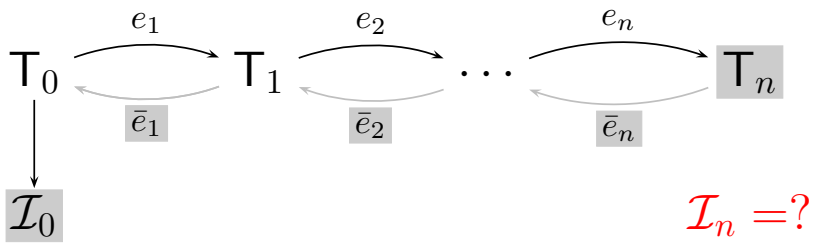
- **Documents** in database **change**
  - changes are **node edit operations**: rename, delete, insert
  - **inverse** edit operation **undoes** the edit operation



- **Index update** required
- **Problem:** computation **from scratch expensive** (> 3 hours for DBLP)
- Only **small subtrees change. . .**
- **Solution:** incrementally **update changing parts** in index

Problem Definition

Scenario:

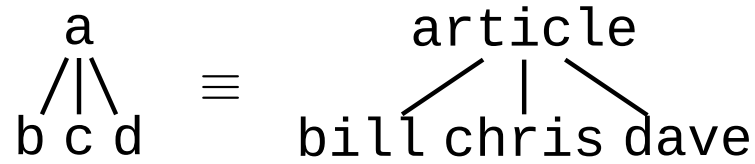


- Example:
  - index for DBLP
  - 1000 updates
  - incrementally update index!

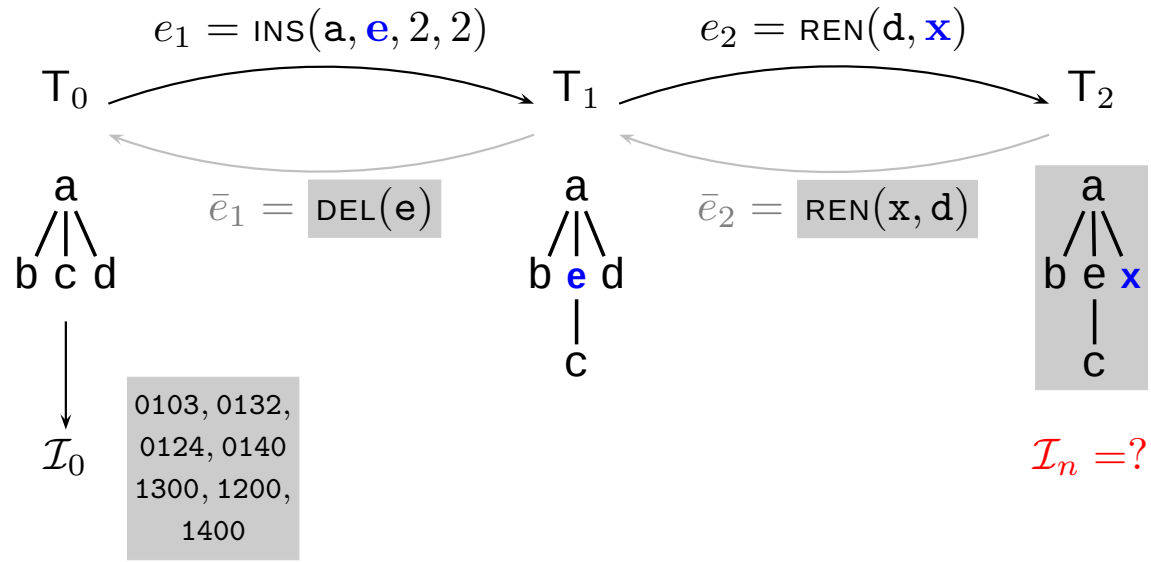
- **Update index** based on
  - old index $\mathcal{I}_0$
  - log of inverse edit operations $(\bar{e}_1, \ldots, \bar{e}_n)$
  - resulting tree T_n
- Do **not** compute $\mathcal{I}_n$ **from scratch**.
- Do **not** compute **intermediate trees**.

Running Example

Example tree:

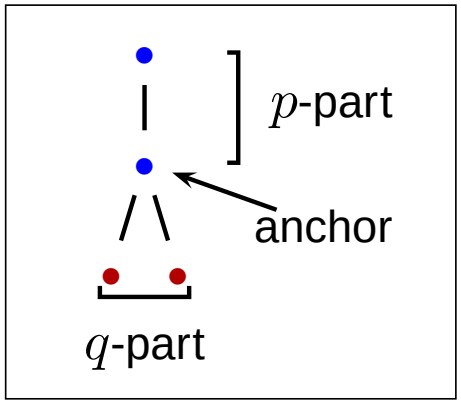


Example Scenario:

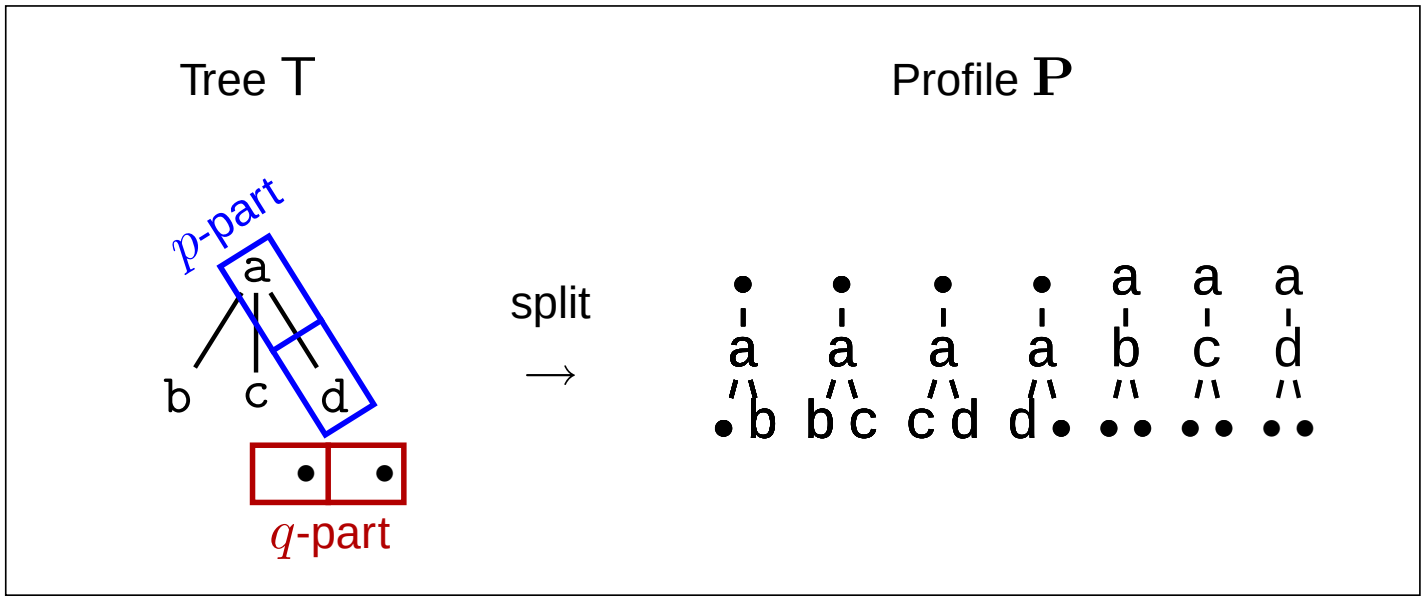


Splitting Tree into pq-Grams

☞ pq-Gram pattern:

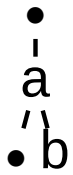


(1) Splitting the tree into 2,2-grams:



Serialization and Hashing

(2) Preorder serialization: pq-gram $\longrightarrow$ serialized pq-gram



serialize $\longrightarrow$ $(\bullet, a, \bullet, b)$ *(shorthand)* $\longrightarrow$ •a•b

(3) Hashing: serialized pq-gram $\longrightarrow$ hash string

☞ Fingerprint hash function:

Hash Function $\lambda()$:

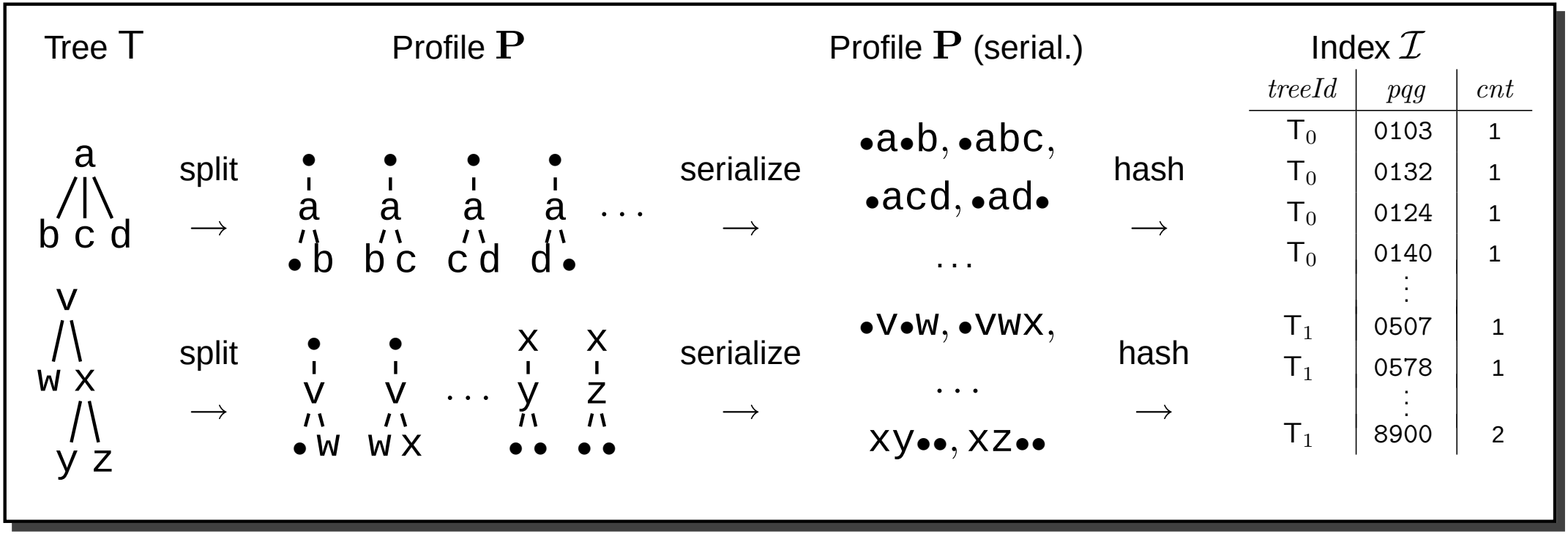
node n	λ(n)
•	0
a	1
b	3
c	2
d	4

☞ Note: •a•b stands for $(\bullet, \texttt{article}, \bullet, \texttt{bill})$

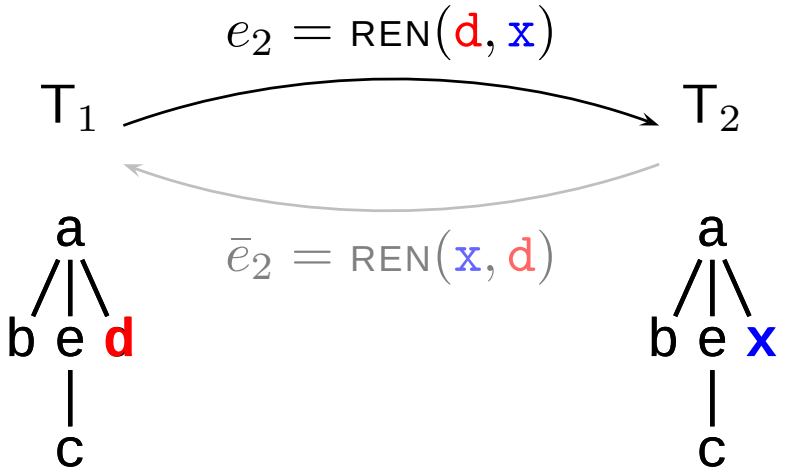
➠ **string tuple** is mapped **to integer**

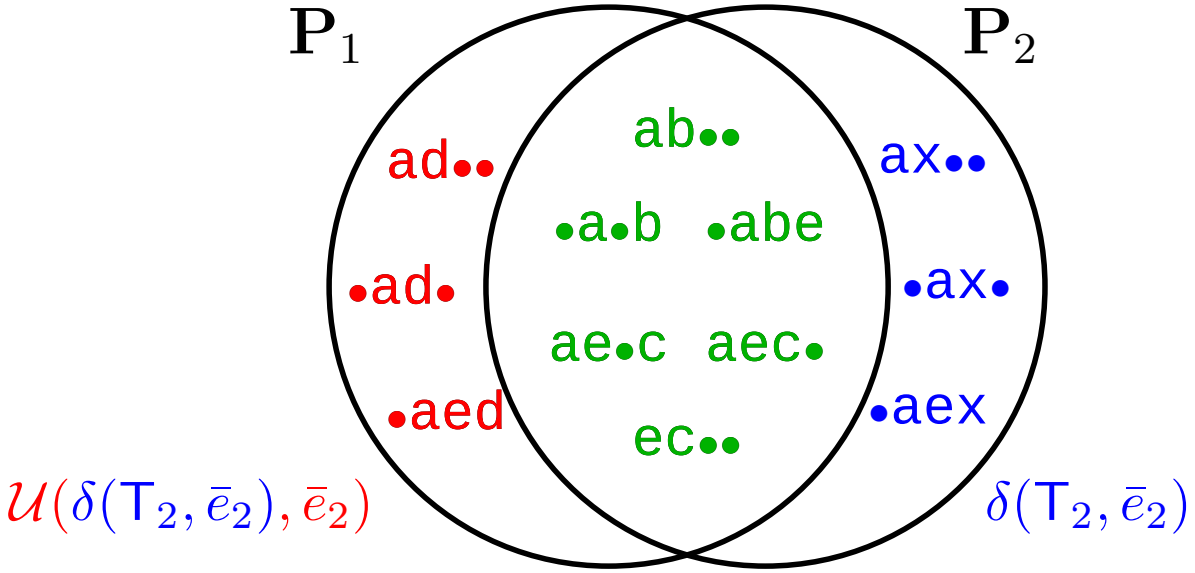
$$(\bullet, \texttt{article}, \bullet, \texttt{bill}) \longrightarrow 0103$$
$$(\bullet, \texttt{article}, \texttt{bill}, \texttt{chris}) \longrightarrow 0132$$
$$\ldots$$

Overview


Tree T
Profile P
Profile P (serial.)
Index I
split
serialize
hash
a
b c d
v
w x
y z
•a•b, •abc,
•acd, •ad•
...
•v•w, •vwx,
...
xy••, xz••
treeId | pqg | cnt
T0 | 0103 | 1
T0 | 0132 | 1
T0 | 0124 | 1
T0 | 0140 | 1
T1 | 0507 | 1
T1 | 0578 | 1
T1 | 8900 | 2

Single Edit Step





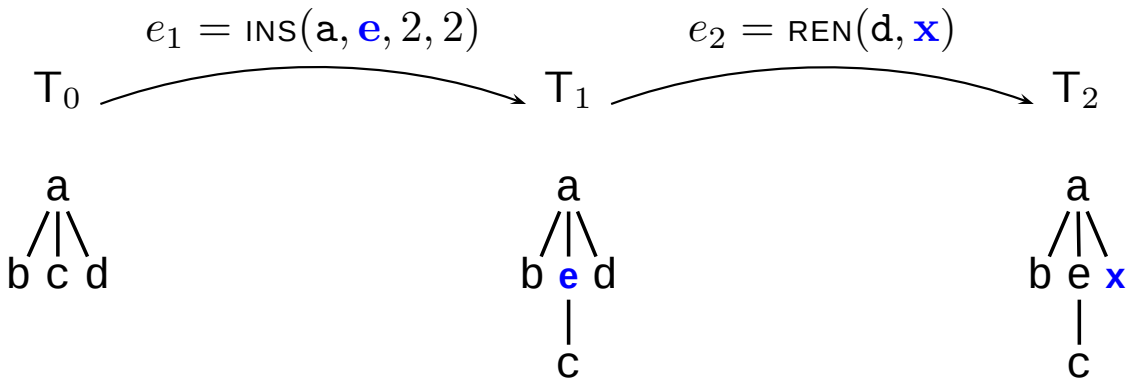
Profile Update Function (old pq-grams)

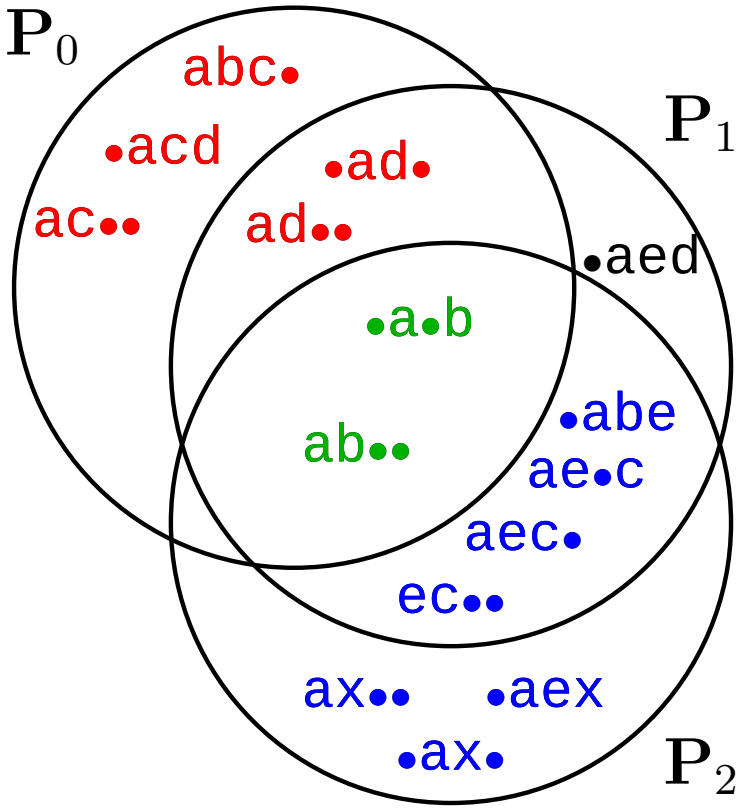
$$\mathcal{U}(\delta(\mathsf{T}_2, \bar{e}_2), \bar{e}_2) = \mathbf{P}_1 \setminus \mathbf{P}_2$$

Delta Function (new pq-grams)

$$\delta(\mathsf{T}_2, \bar{e}_2) = \begin{cases} \mathbf{P}_2 \setminus \mathbf{P}_1 & \text{if } \bar{e}_2 \text{ is defined on } \mathsf{T}_2 \\ \emptyset & \text{otherwise} \end{cases}$$

Multiple Edit Steps





☞ Invariant pq-Grams (no need for update)

$$\mathbf{C}_n = \mathbf{P}_0 \cap \cdots \cap \mathbf{P}_n$$

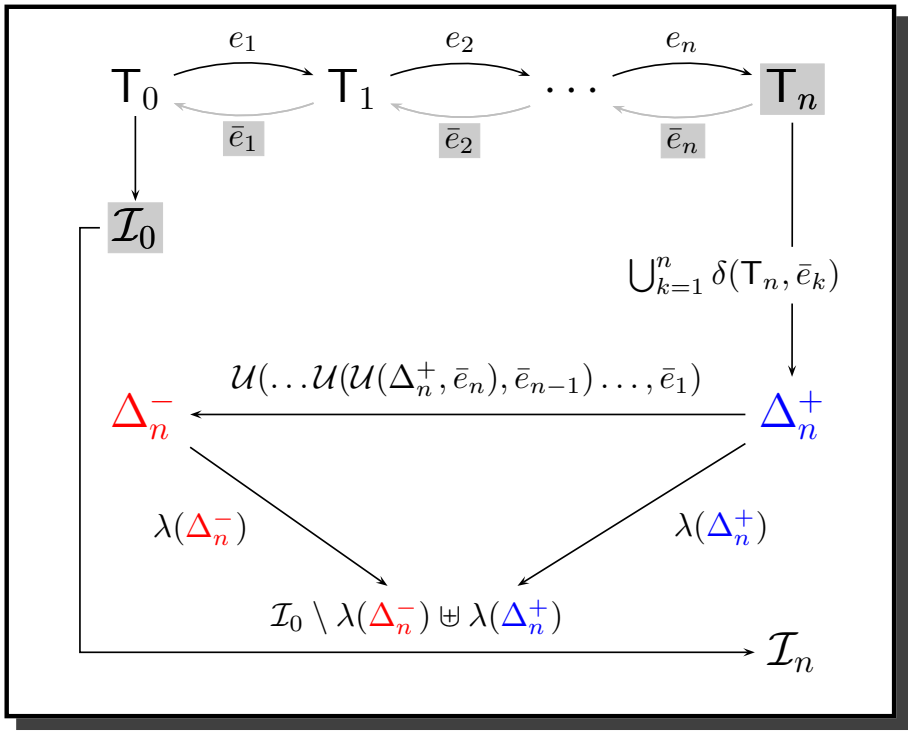
☞ Old pq-Grams (deleted from old index $\mathcal{I}_0$)

$$\Delta_n^- = \mathbf{P}_0 \setminus \mathbf{C}_n$$

☞ New pq-Grams (inserted into new index $\mathcal{I}_n$)

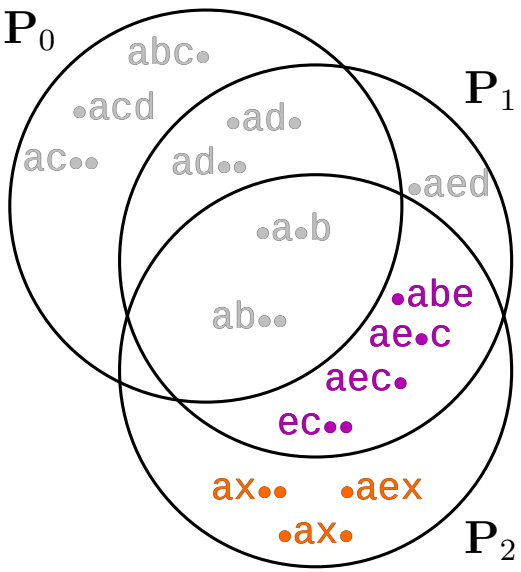
$$\Delta_n^+ = \mathbf{P}_n \setminus \mathbf{C}_n$$

Solution (Overview)



- **Input:**
  1. old index $\mathcal{I}_0$
  2. log of inverse edit operations $(\bar{e}_n, \ldots, \bar{e}_1)$
  3. resulting tree T_n
- **Output:** new index $\mathcal{I}_n$
- **Solution:** 3 Steps
  1. compute **new pq-grams**:
     $\Delta_n^+ = \delta(\mathsf{T}_n, \bar{e}_1) \cup \cdots \cup \delta(\mathsf{T}_n, \bar{e}_n)$
  2. compute **old pq-grams**:
     $\Delta_n^- = \mathcal{U}(\ldots\mathcal{U}(\mathcal{U}(\Delta_n^+, \bar{e}_n), \bar{e}_{n-1})\ldots, \bar{e}_1)$
  3. **update index**:
     $\mathcal{I}_n = \mathcal{I}_0 \setminus \lambda(\Delta_n^-) \uplus \lambda(\Delta_n^+)$

Example of Δ^+ Computation



$$\Delta_2^+ = \underbrace{[\delta(\mathsf{T}_1, \bar{e}_1) \cap \mathbf{P}_2]}_{\mathsf{T}_1 \text{ not given!}} \cup \underbrace{\delta(\mathsf{T}_2, \bar{e}_2)}_{\text{OK}}$$

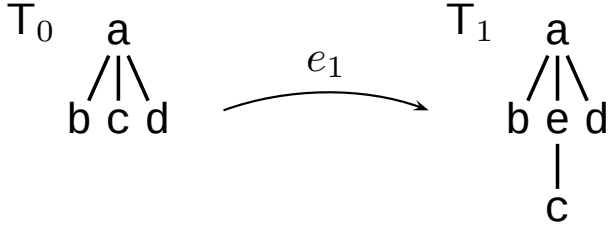
☞ **Compute δ on wrong tree** (T_2) and fix it

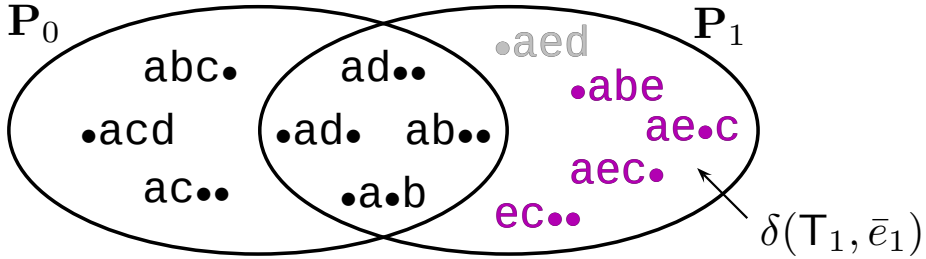
$$\delta(\mathsf{T}_1, \bar{e}_1) \cap \mathbf{P}_2 = \delta(\mathsf{T}_2, \bar{e}_1) \setminus \delta(\mathsf{T}_2, \bar{e}_2)$$

☞ $$\Delta_2^+ = \delta(\mathsf{T}_2, \bar{e}_1) \setminus \delta(\mathsf{T}_2, \bar{e}_2) \cup \delta(\mathsf{T}_2, \bar{e}_2)$$

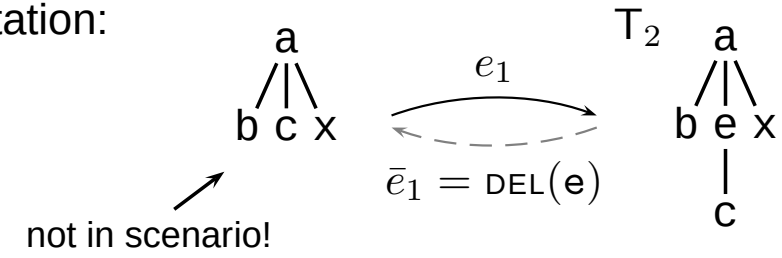
$$\boxed{\Delta_2^+ = \delta(\mathsf{T}_2, \bar{e}_1) \cup \delta(\mathsf{T}_2, \bar{e}_2)}$$

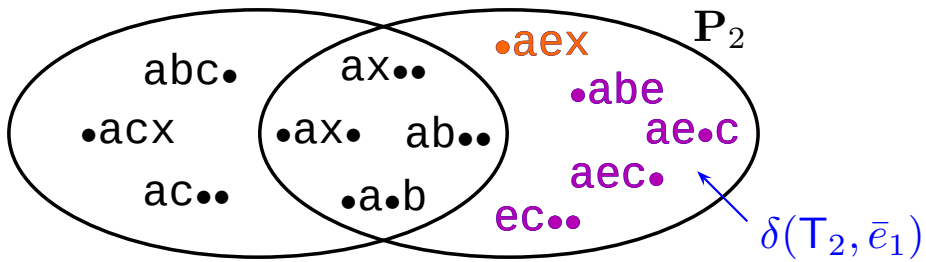
Reality:



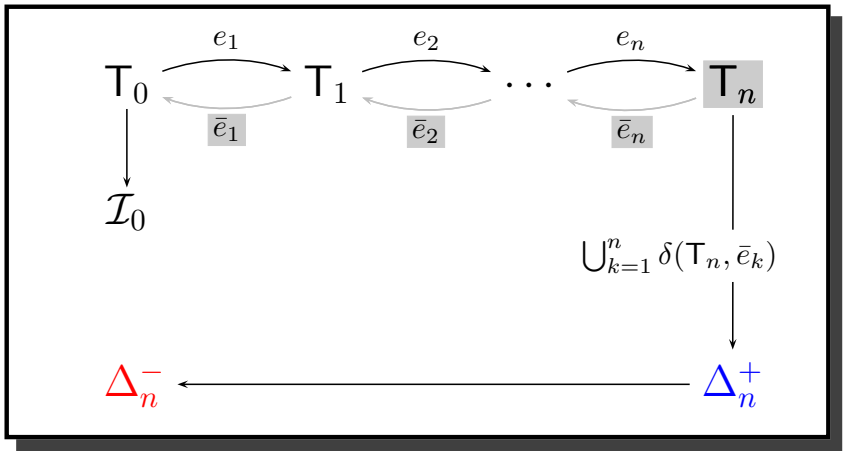


Computation:





Computing Δ^+



☞ new pq-grams can be computed using:

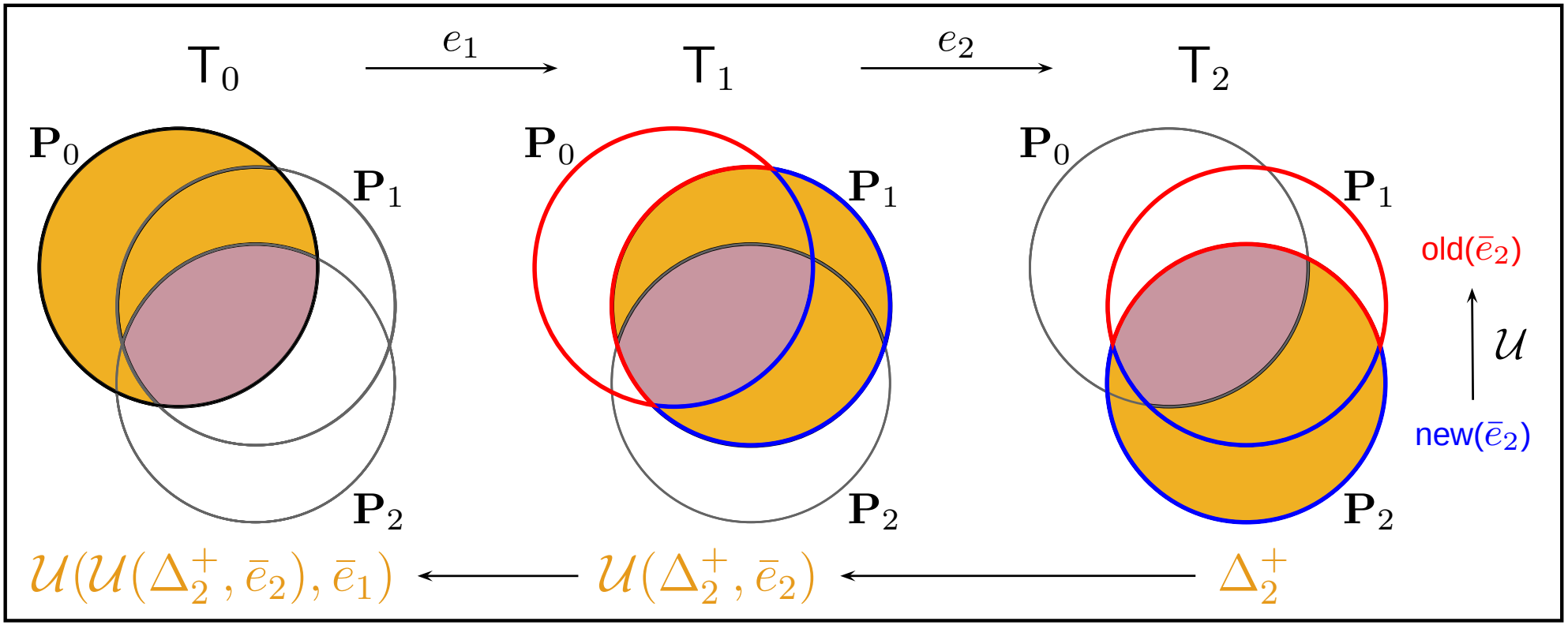
- ⇨ the resulting tree T_n
- ⇨ the log $(\bar{e}_1, \ldots, \bar{e}_n)$

Theorem 1 *The set of new pq-grams can be computed on T_n as*

$$\Delta_n^+ = \bigcup_{k=1}^{n} \delta(\mathsf{T}_n, \bar{e}_k)$$

☞ **Next step:** compute old pq-grams Δ_n^-

Example of Δ^- Computation



☞ **Profile update function $\mathcal{U}$:** transforms new pq-grams into old pq-grams.

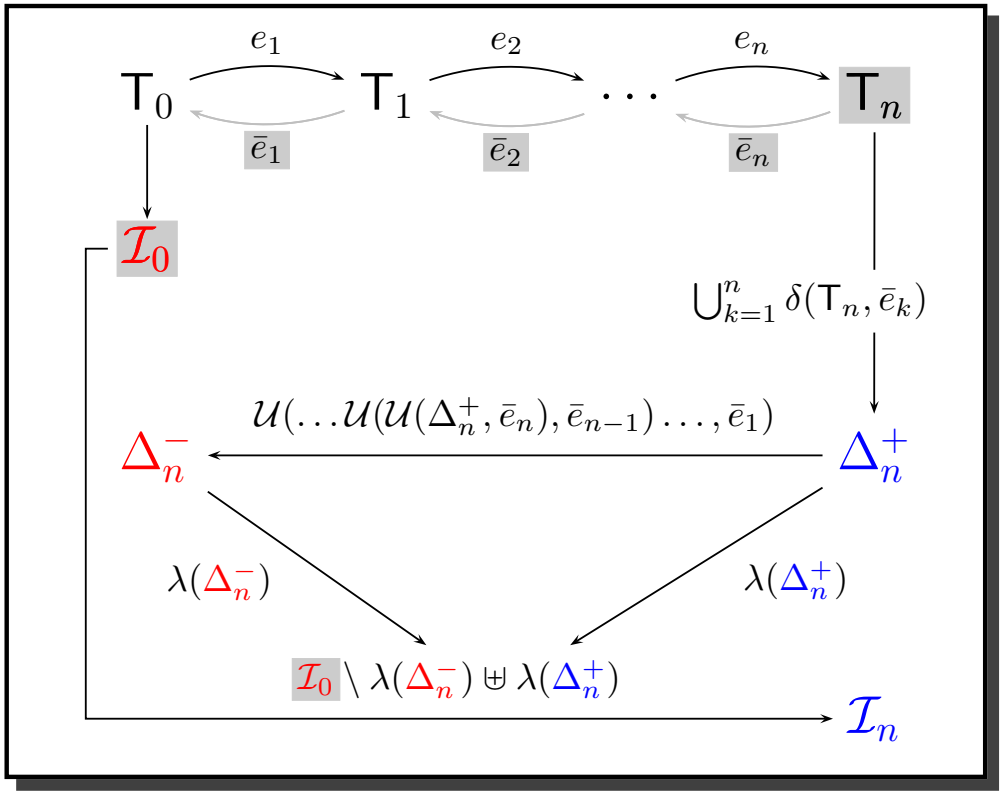
$$\mathcal{U}(\mathcal{U}(\Delta_2^+, \bar{e}_2), \bar{e}_1) = \mathbf{\Delta_2^-}$$

Computing Δ^-

Theorem 2 *The old pq-grams Δ_n^- are computed by recursively applying the* ***update function*** $\mathcal{U}$ *to the new pq-grams Δ_n^+:*

$$\Delta_n^- = \mathcal{U}(\ldots\mathcal{U}(\mathcal{U}(\Delta_n^+, \bar{e}_n), \bar{e}_{n-1})\ldots, \bar{e}_1).$$

Hash pq-Grams and Update Old Index



- **Hash** old (Δ_n^+) and new (Δ_n^-) pq-grams
- **Update** old index $\mathcal{I}_0$

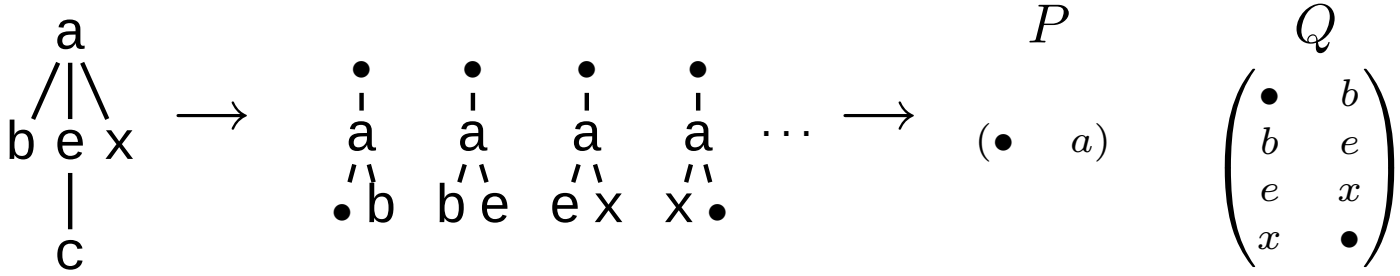
Algorithm

Algorithm 1: $\texttt{updateIndex}(\mathtt{I}_0, \mathsf{T}_n, log)$

1 $\Delta_n^+ = \emptyset$;
2 **foreach** $\bar{e}_i \in log$ **do** $\Delta_n^+ \leftarrow \Delta_n^+ \cup \delta(\mathsf{T}_n, \bar{e}_i)$;
3 $\Delta_n^- \leftarrow \Delta_n^+$;
4 **for** $\bar{e}_i \leftarrow \bar{e}_n$ **downto** $\bar{e}_1$ **do** $\Delta_n^- \leftarrow \mathcal{U}(\Delta_n^-, \bar{e}_i)$;
5 $\mathtt{I}^+ \leftarrow \lambda(\Delta_n^+)$;
6 $\mathtt{I}^- \leftarrow \lambda(\Delta_n^-)$;
7 $\mathtt{I}_n \leftarrow \mathtt{I}_0 \setminus \mathtt{I}^- \cup \mathtt{I}^+$;
8 **return** $\mathtt{I}_n$;

Implementation

☞ Store p-part and q-part in tables P and Q



Formulas for Updating Δ_n^-

Insert node n as the k-th child of node v: $\text{INS}(\mathsf{n}, \mathsf{v}, k, m)$

$$\delta(\mathsf{T}_j, \bar{e}) = P(\mathsf{v}) \circ Q^{k..m}(\mathsf{v}) \cup P(\mathsf{x}) \circ Q(\mathsf{x}) \qquad \forall \mathsf{x} \in \text{desc}_{p-2}(\mathsf{c}_k, \ldots, \mathsf{c}_m)$$

$$\mathcal{U}(\delta(\mathsf{T}_j, \bar{e}), \bar{e}) = P(\mathsf{v}) \circ [Q^{k..m}(\mathsf{v}) /\!\!/ D(\mathsf{n})] \cup P^{+\mathsf{n},0}(\mathsf{v}) \circ \qquad \forall \mathsf{x} \in \text{desc}_{p-2}(\mathsf{c}_k, \ldots, \mathsf{c}_m),\ d = \text{dist}(\mathsf{c}_i, \mathsf{x}) + 1$$

$$[D(\bullet) /\!\!/ Q^{k..m}(\mathsf{v})] \cup P^{+\mathsf{n},d}(\mathsf{x}) \circ Q(\mathsf{x}) \qquad \mathsf{c}_i : i\text{-th child of } \mathsf{v}$$

Delete node n, $\text{DEL}(\mathsf{n})$:

$$\delta(\mathsf{T}_j, \bar{e}) = P(\mathsf{v}) \circ Q^{k..k}(\mathsf{v}) \cup P(\mathsf{x}) \circ Q(\mathsf{x}) \qquad \forall \mathsf{x} \in \text{desc}_{p-1}(\mathsf{n})$$

$$\mathcal{U}(\delta(\mathsf{T}_j, \bar{e}), \bar{e}) = P(\mathsf{v}) \circ [Q^{k..k}(\mathsf{v}) /\!\!/ Q(\mathsf{n})] \cup P^{-\mathsf{n}}(\mathsf{x}) \circ Q(\mathsf{x}) \qquad \forall \mathsf{x} \in \text{desc}_{p-1}(\mathsf{n}) \setminus \{\mathsf{n}\}$$

$\mathsf{v} : \mathsf{n}$ is the k-th child of v

Rename node n to l': $\text{REN}(\mathsf{n}, \mathtt{l}')$

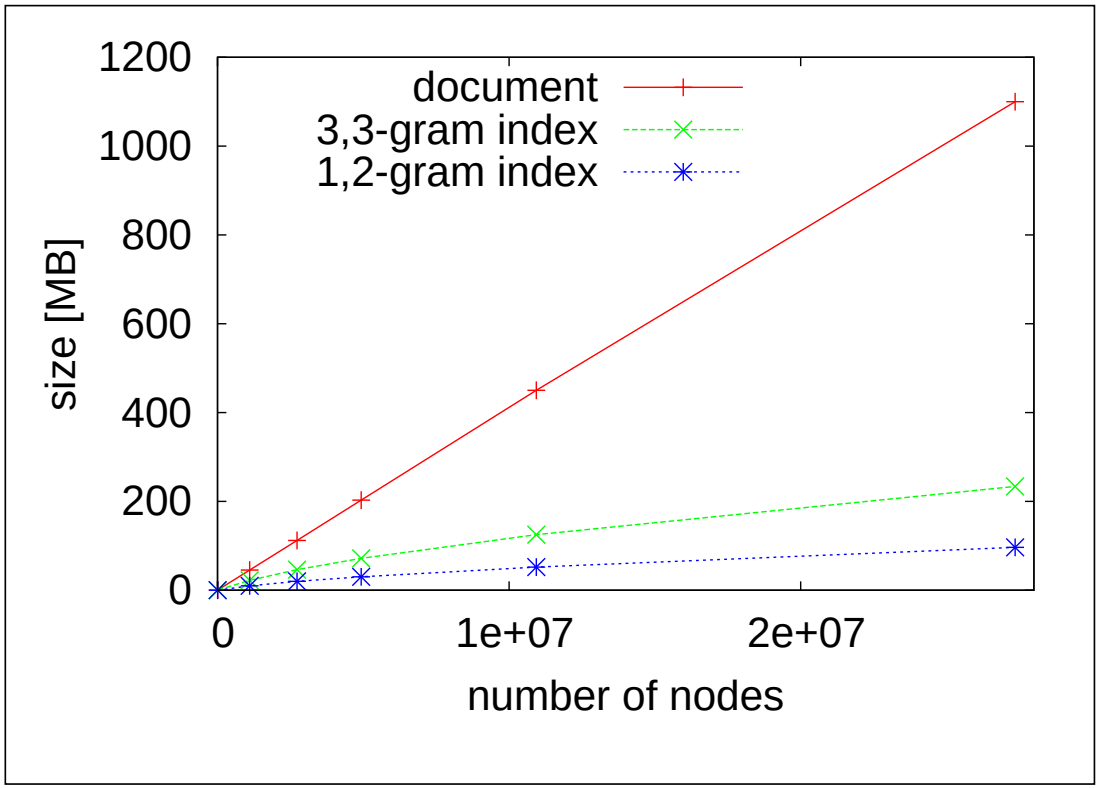
$$\delta(\mathsf{T}_j, \bar{e}) = P(\mathsf{v}) \circ Q^{k..k}(\mathsf{v}) \cup P(\mathsf{x}) \circ Q(\mathsf{x}) \qquad \forall \mathsf{x} \in \text{desc}_{p-1}(\mathsf{n})$$

$$\mathcal{U}(\delta(\mathsf{T}_j, \bar{e}), \bar{e}) = P(\mathsf{v}) \circ [Q^{k..k}(\mathsf{v}) /\!\!/ D(\mathsf{m})] \cup P^{\mathsf{n}/\mathsf{m}}(\mathsf{x}) \circ Q(\mathsf{x}) \qquad \forall \mathsf{x} \in \text{desc}_{p-1}(\mathsf{n})$$

$\mathsf{m} = (\text{id}(\mathsf{n}), l')$ $\mathsf{v} : \mathsf{n}$ is the k-th child of v

Index Size

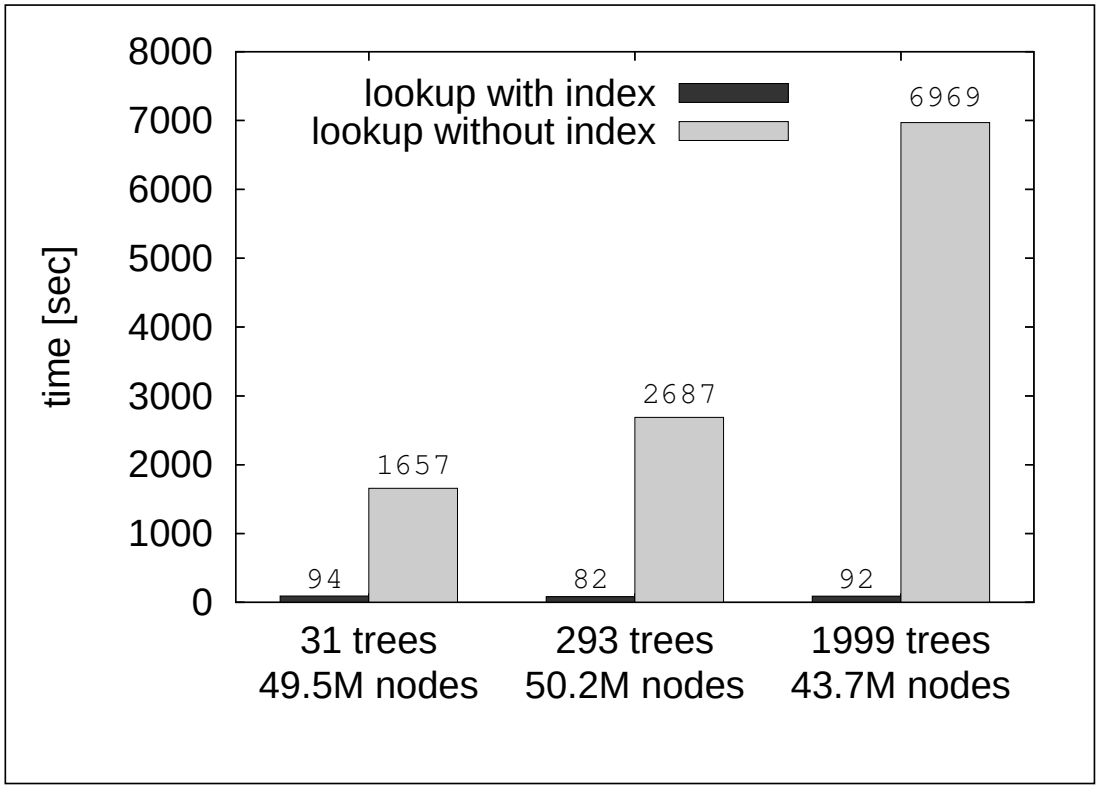
Index size linear in tree size



- Experiment: **synthetic** data (XMark)
  - vary document size
  - compute index
  - compare index size with tree size

Lookup Efficiency

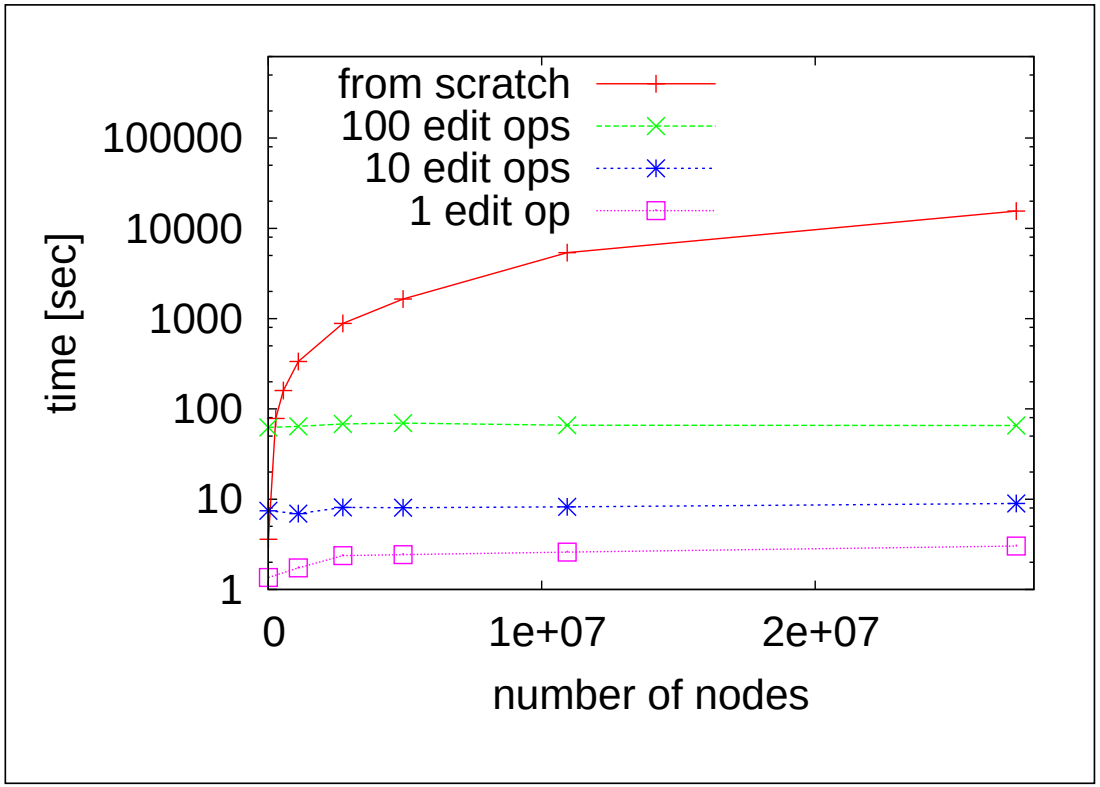
Index greatly increases efficiency for lookup



- Experiment: **synthetic** data (XMark)
  - lookup with/without index
  - different document sets of similar size
  - measure wall clock time

Incremental Update vs. Recomputation

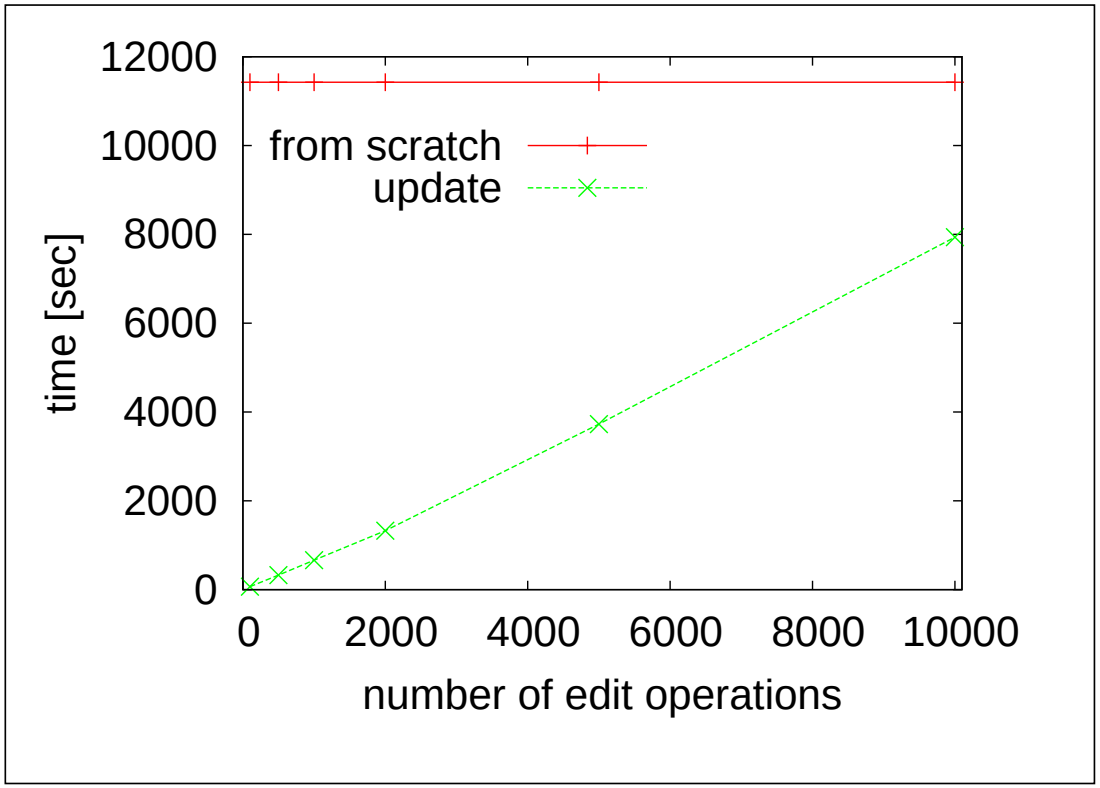
Update independent of tree size



- Experiment: **synthetic** data (XMark)
  - vary tree size (up to 27M nodes)
  - compute incremental update
  - compute index from scratch
  - measure wall clock time

Incremental Update Scalability

Update linear in **log size**



- Experiment: **DBLP** (211MB, 11M nodes)
  - vary number of edit operations
  - compute incremental update
  - compute index from scratch
  - measure wall clock time

Breakdown of Update Time

Δ^+, Δ^-: main share

Hashing very cheap

$\mathcal{I}_n = \mathcal{I}_0 \setminus \mathcal{I}^- \cup \mathcal{I}^+$: **sublinear** in **log size**

Action	Number of edit operations			
	1	10	100	1000
Δ_n^+	0.642s	3.903s	37.533s	391.513s
$\mathtt{I}^+ = \lambda(\Delta_n^+)$	0.184s	0.199s	0.287s	0.443s
Δ_n^-	0.196s	2.836s	27.967s	295.104s
$\mathtt{I}^- = \lambda(\Delta_n^-)$	0.177s	0.191s	0.185s	0.383s
$\mathtt{I}_0 \setminus \mathtt{I}^- \cup \mathtt{I}^+$	2.206s	2.770s	6.475s	19.780s
total	3.405s	9.900s	72.448s	707.224s

- Experiment: **DBLP** (211MB, 11M nodes)
  - vary number of edit operations
  - compute incremental update
  - wall clock time for each steps

Related Work

- pq**-Gram distance** (Augsten et al., VLDB 2005)
  - edit distance approximation
  - properties analyzed in (Augsten et al., VLDB 2005)
- Other **edit distance approximations**:
  - (Garofalakis & Kumar, TODS 2005): *approximation guarantees* with tree edit distance embedding
  - (Yang et al., SIGMOD 2005): *lower bound* for edit distance
  - (Weis and Naumann, SIGMOD 2005): XML *duplicate detection*
  - (Cobéna et al., ICDE 2002), (Lee et al., TKDE 2004): XML *change detection*
- **Approximate XML join** by (Guha et al., SIGMOD 2002), **index** by (Guha et al., ICDE 2003)
  - optimize join based on tree edit distance using reference sets
  - updates of reference sets not addressed

Conclusion and Future Work

- **Proof for incremental update** of pq-gram index with
  - old index
  - resulting document
  - edit-log
- **Update efficient**
  - constant in tree size
  - linear in edit-log size
- **Future work**
  - optimize edit-log (e.g. remove redundancy)
  - subtree edit operations (e.g. subtree move)
  - compute updates for edit ops in parallel