

Similarity Search

The pq -Gram Distance

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



WS 2021/22
Version October 26, 2021

Outline

- 1 The pq -Gram Distance
 - Definition
 - Algorithm
 - Fanout Weighting and Lower Bound
 - Experiments
- 2 Conclusion

Outline

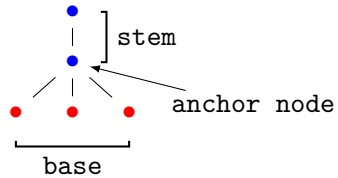
- 1 The pq -Gram Distance
 - Definition
 - Algorithm
 - Fanout Weighting and Lower Bound
 - Experiments
- 2 Conclusion

pq -Grams – Intuition

- q -Grams for strings:
 - split string into substrings (q -grams) of length q
 - strings with many common substrings are similar
- pq -Grams for trees:
 - split tree into small subtrees (pq -grams) of the same shape
 - trees with many common subtrees are similar

pq -Grams

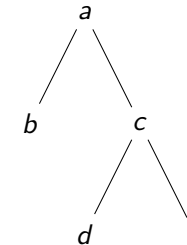
- The shape of a pq -gram ($p=2, q=3$):



- p nodes (anchor node and $p-1$ ancestors) form the **stem**
- q nodes (q consecutive children of the anchor node) form the **base**

pq -Extended Tree

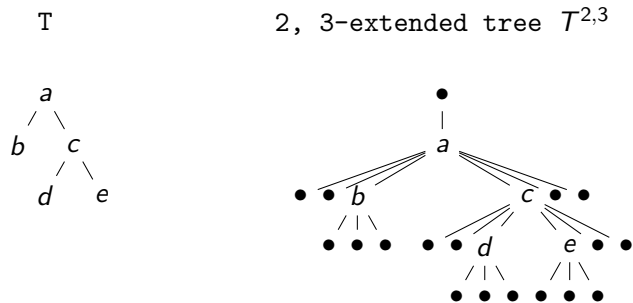
- Problem:** How can we split the following tree T into 2,3-grams?



- Solution:** Extend tree T with dummy nodes (\bullet):
 - $p-1$ ancestors to the root node
 - $q-1$ children before the first and after the last child of each non-leaf
 - q children for each leaf
- The result is the pq -extended tree T^{pq} .

Example: Extended Tree

- An example tree T and its extended tree T^{pq} ($p=2, q=3$):



Definition: pq -Gram [ABG05]

Definition (pq -Gram)

Let T be a tree, $T^{p,q}$ the respective extended tree, $p > 0, q > 0$. A subtree of $T^{p,q}$ is a **pq -gram** g of T iff

- g has q leaf nodes and p non-leaf nodes,
- all leaf nodes of g are children of a single node $a \in N(g)$ with fanout q , called the **anchor node**,
- the leaf nodes of g are consecutive siblings in $T^{p,q}$.

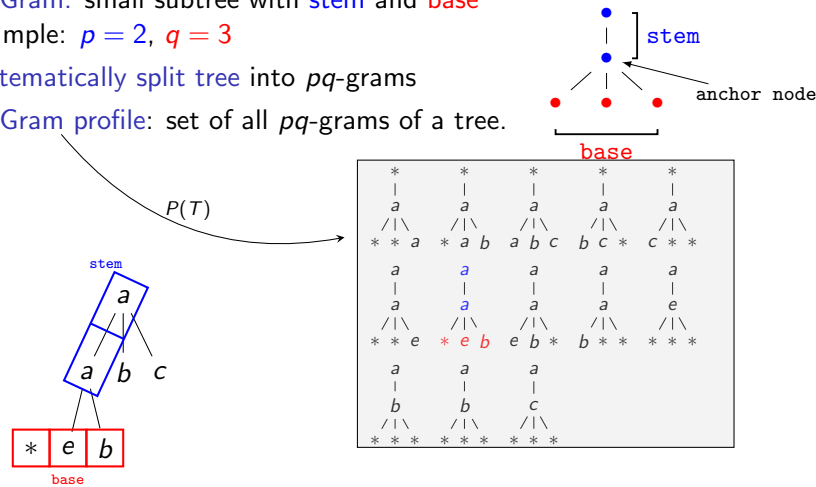
- Stem:** anchor node and its ancestors in the pq -gram.
- Base:** children of the anchor node in the pq -gram.

Definition (pq -Gram Profile)

The pq -gram profile, P_T , of a tree T is the set of all its pq -grams.

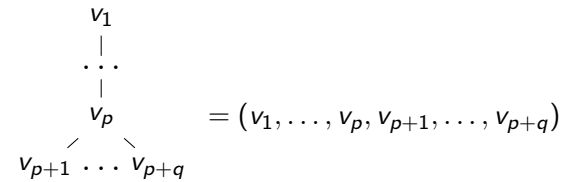
Example: Systematically Split Tree

- **pq-Gram**: small subtree with **stem** and **base**
Example: $p = 2, q = 3$
- Systematically split tree into pq-grams
- **pq-Gram profile**: set of all pq-grams of a tree.



Label Tuples

- **Linear encoding** of a pq-gram g with anchor node v_p :
(traverse pq-gram in preorder)



- **Label tuple**: tuple of the pq-gram's node labels

$$\lambda(g) = (\lambda(v_1), \dots, \lambda(v_{p+q}))$$

for the pq-gram $g = (v_1, \dots, v_{p+q})$.

pq-Gram Index

Definition (pq-Gram Index)

Let T be a tree with profile P_T , $p > 0, q > 0$. The pq-gram index, \mathcal{I} , of tree T is the **bag of all label tuples** of T ,

$$\mathcal{I}(T) = \bigoplus_{g \in P_T} \lambda(g)$$

- **Note:**
 - pq-grams are unique within a tree
 - but: different pq-grams may yield identical label tuples
 - thus the pq-gram index may contain duplicates

Storing the pq-Gram Index Efficiently

- **Problem**: How to store node labels efficiently?
 - **Long labels**: large storage overhead
 - **Varying label length**: in a relational database, the inefficient VARCHAR type must be used instead of the efficient CHAR type
- **Solution**: Hashing
 - compute fingerprint hash for labels
 - store concatenation of the hashed labels
- **Fingerprint hash function** (e.g., Karp-Rabin [KR87]):
 - maps a string s to a hash value $h(s)$
 - $h(s)$ is of fixed length
 - $h(s)$ is unique with high probability
(for two different strings $s_1 \neq s_2$, $h(s_1) \neq h(s_2)$ with high probability)

Overview: The pq-Gram Index

- pq-Gram profile:



- Hashing: map pq-gram to integer:



Note: labels may be strings of arbitrary length!

- pq-Gram index: bag of hashed pq-grams
 $\mathcal{I}(T) = \{03003, 03037, 03376, 03760, 03600, 33004, 33047, 33470, 33700, 37000, 36000, 34000, 37000\}$

Intuition: similar trees have similar pq-gram indexes.

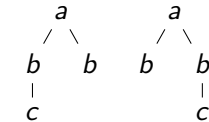
The pq-Gram Distance

Definition (pq-Gram Distance)

The pq-gram distance between two trees, T and T' , is defined as

$$\delta_g(T, T') = |\mathcal{I}(T) \uplus \mathcal{I}(T')| - 2|\mathcal{I}(T) \cap \mathcal{I}(T')|$$

- Metric normalization to $[0..1]$: $\delta'_g(T, T') = \frac{\delta_g(T, T')}{|\mathcal{I}(T) \uplus \mathcal{I}(T')| - |\mathcal{I}(T) \cap \mathcal{I}(T')|}$
- Pseudo-metric properties hold for normalization [ABG10]:
 - ✓ self-identity: $x = y \neq \Rightarrow \delta_g(x, y) = 0$
 - ✓ symmetry: $\delta_g(x, y) = \delta_g(y, x)$
 - ✓ triangle inequality: $\delta_g(x, z) \leq \delta_g(x, y) + \delta_g(y, z)$



- Different trees may have identical indexes:

Outline

- 1 The pq-Gram Distance
 - Definition
 - Algorithm
 - Fanout Weighting and Lower Bound
 - Experiments
- 2 Conclusion

Main Memory Algorithm (I)

CREATEINDEX($T, r, \mathcal{I}, stem, p, q$)

$stem := shift(stem, \lambda(r))$

$base$: shift register of size q (filled with $*$)

if r is a leaf then

$\mathcal{I} := \mathcal{I} \cup \{stem \circ base\}$

else

for each child c (from left to right) of r do

$base := shift(base, \lambda(c))$

$\mathcal{I} := \mathcal{I} \cup \{stem \circ base\}$

$\mathcal{I} := CREATEINDEX(T, c, \mathcal{I}, stem, p, q)$

for $k := 1$ to $q - 1$

$base := shift(base, *)$

$\mathcal{I} := \mathcal{I} \cup \{stem \circ base\}$

return \mathcal{I}

Main Memory Algorithm (II)

- **Input** of $\text{CREATEINDEX}(T, r, \mathcal{I}, \text{stem}, p, q)$:
 - a subtree of T rooted in r
 - the pq -gram index \mathcal{I} computed so far
 - the stem stem of r 's parent
 - the parameters p and q
- **Output** of $\text{CREATEINDEX}(T, r, \mathcal{I}, \text{stem}, p, q)$:
 pq -gram index including
 - the input index \mathcal{I}
 - the pq -gram index of r and all its descendants
 i.e., the pq -grams (label tuples) with anchor node r or a descendant of r

Main Memory Algorithm (III)

- pq -GRAM-INDEX(T, p, q) computes the pq -gram index for a complete tree T :

 pq -GRAM-INDEX(T, p, q)

stem : shift register of size p (filled with $*$)

\mathcal{I} : empty index

$\mathcal{I} = \text{CREATEINDEX}(T, \text{root}(T), \mathcal{I}, \text{stem}, p, q)$

return \mathcal{I}

Complexity of the pq -Gram Index AlgorithmTheorem (pq -Gram Index Complexity)

The pq -gram index of a tree T with size $|T|$ can be computed in $O(|T|)$ time.

Proof.

Each recursive call of $\text{createIndex}()$ processes one node in constant time, and each node is processed exactly once. \square

Size of the pq -Gram IndexTheorem (Size of the pq -Gram Index)

Let T be a tree with l leaves and i non-leaves. The size of the pq -gram index of T is

$$|\mathcal{I}^{pq}(T)| = 2l + qi - 1.$$

Proof.

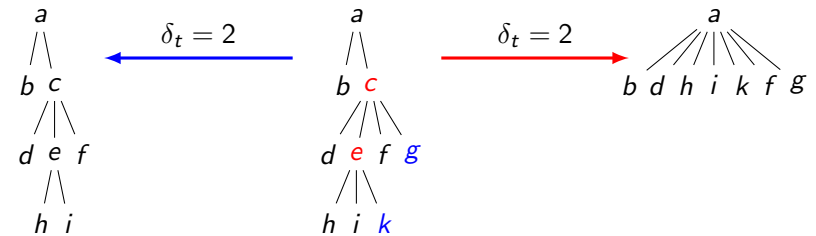
1. We **count** all pq -grams whose leftmost leaf is a **dummy node**: Each leaf is the anchor node of exactly one pq -gram whose leftmost leaf is a dummy node, giving l pq -grams. Each non-leaf is the anchor of $q - 1$ pq -grams whose leftmost leaf is a dummy, giving $i(q - 1)$ pq -grams.
2. We **count** all pq -grams whose leftmost leaf is **not a dummy node**: Each node of the tree except the root is the leftmost leaf of exactly one pq -gram, giving $l + i - 1$ pq -grams.

Overall number of pq -grams: $l + i(q - 1) + (l + i - 1) = 2l + qi - 1$. \square

Outline

- 1 The pq -Gram Distance
 - Definition
 - Algorithm
 - Fanout Weighting and Lower Bound
 - Experiments
- 2 Conclusion

Motivation: Unit Cost Model Not Always Intuitive



- Unit cost edit distance:
 - no difference between leaves and non-leaves
 - may lead to non-intuitive results
- **Conclusion:** Non-leaves should have more weight than leaves.

Fanout Weighted Tree Edit Distance

Definition (Fanout Weighted Tree Edit Distance)

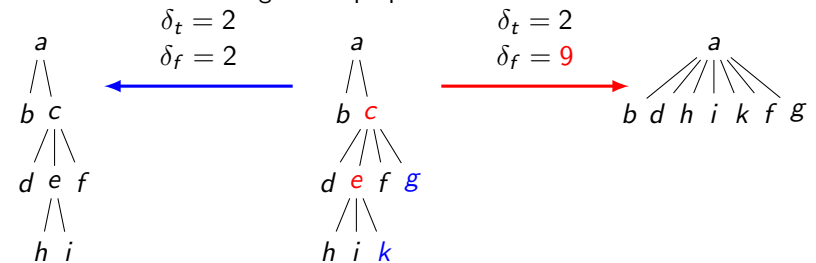
Let T and T' be two trees, $w \in N(T)$ a node with fanout f , $w' \in N(T')$ a node with fanout f' , $c > 0$ a constant. The **fanout weighted tree edit distance**, $\delta_f = (T, T')$, between T and T' is defined as the tree edit distance with the following costs for the edit operations:

- Delete: $\alpha(w \rightarrow \epsilon) = f + c$
- Insert: $\alpha(\epsilon \rightarrow w') = f' + c$
- Rename: $\alpha(w \rightarrow w') = (f + f')/2 + c$

- Cost of changing a **non-leaf** node: proportional to its fanout.
- Cost of changing a **leaf** node: constant c .

Example: Fanout-Weighted Tree Edit Distance

- Fanout-Weighted Tree Edit Distance:
 - leaf changes have small cost ($c = 1$ in the example)
 - non-leaf changes cost proportional to the node fanout



pq -Gram Distance Lower Bound

Theorem

Let $p = 1$ and $c \geq \max(2q - 1, 2)$ be the cost of changing a leaf node. The pq -gram distance provides a lower bound for the fanout weighted tree edit distance, i.e., for any two trees, T and T' ,

$$\frac{\delta_g(T, T')}{2} \leq \delta_f(T, T').$$

Proof.

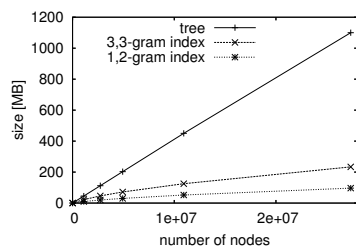
See [ABG10] (ACM Transactions on Database Systems). □

Outline

- 1 The pq -Gram Distance
 - Definition
 - Algorithm
 - Fanout Weighting and Lower Bound
 - Experiments
- 2 Conclusion

Size of the pq -Gram Index

- pq -Gram index size: linear in the tree size
- Experiment:
 - compute pq -gram index for trees with different number of nodes
 - compare tree and index size



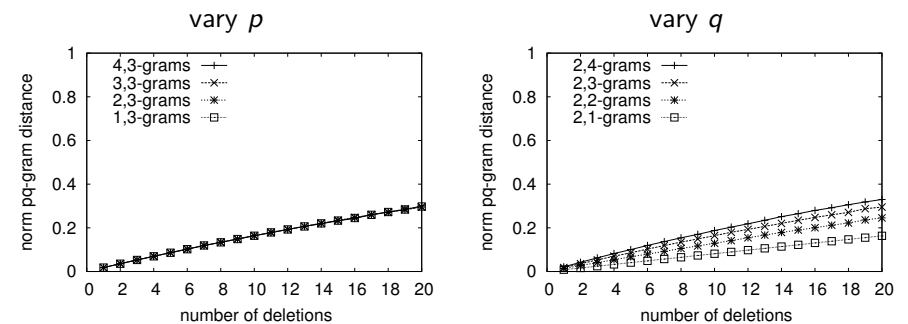
[Trees created with `xmlgen`.]

Why is the pq -gram index smaller than the tree?

- hash values are smaller than labels
- duplicate pq -grams of a tree are stored only once

Sensitivity to Structure Change — Leaf

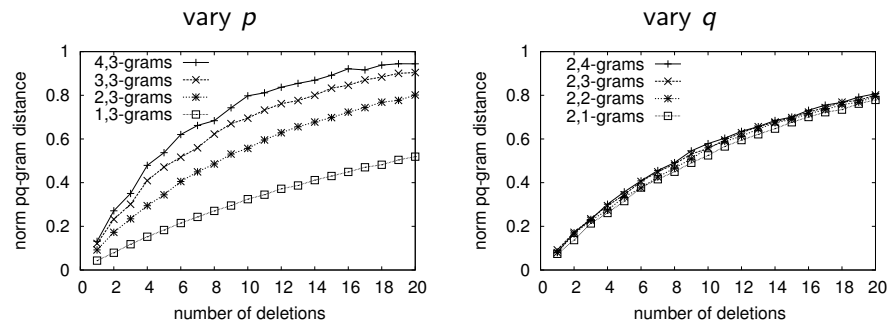
- Cost of leaf change \rightarrow depends only on q
- Experiment:
 - delete leaf nodes
 - measure normalized pq -gram distance



(Artificial tree with 144 nodes, 102 leaves, fanout 2–6 and depth 6. Average over 100 runs.)

Sensitivity to Structure Change — Non-Leaf

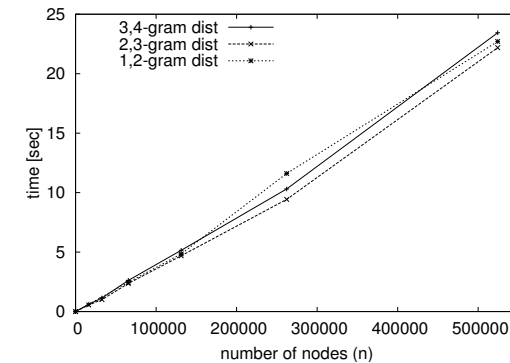
- Cost for non-leaf change \rightarrow controlled by p
- Experiment:
 - delete non-leaf nodes
 - measure normalized pq -gram distance



(Artificial tree with 144 nodes, 102 leaves, fanout 2–6 and depth 6. Average over 100 runs.)

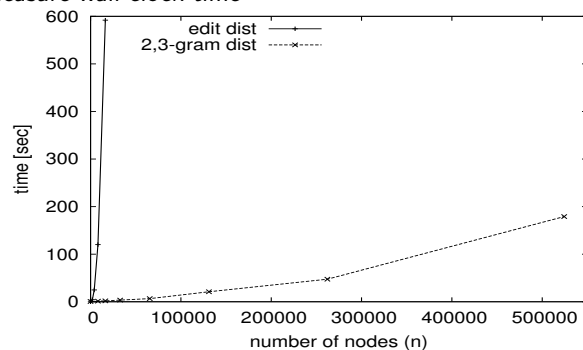
Influence of p and q on Scalability

- Scalability (almost) independent of p and q .
- Experiment: For pair of trees
 - compute pq -gram distance for varying p and q
 - vary tree size: up to 10^6 nodes
 - measure wall clock time



Scalability to Large Trees

- pq -gram distance \rightarrow scalable to large trees
- compare with edit distance
- Experiment: For pair of trees
 - compute tree edit distance and pq -gram distance
 - vary tree size: up to 5×10^5 nodes
 - measure wall clock time

 pq -Grams vs. other Edit Distance Approximations

Effectiveness: pq -grams outperform all other approximations




Experiment: two sets of address trees (299 and 302 trees)

- compute distances between all tree pairs
- find matches (symmetric nearest neighbor)

Distance	Correct	Recall	Precision	f-Measure	Runtime
fanout edit dist	259	86.6%	98.5%	0.922	19 min
unit edit dist	247	82.6%	96.5%	0.890	14 min
node intersection	197	65.9%	93.8%	0.774	4.3s
p, q -grams	236	78.9%	98.7%	0.877	8.1s
tree-embedding	206	68.9%	96.3%	0.803	7.1s
binary branch	193	64.5%	93.2%	0.763	7.4s
bottom-up	148	49.6%	92.5%	0.645	67.0s

Summary

- Binary Branch Distance
 - lower bound of the **unit cost** tree edit distance
 - trees are split into binary branches (small subgraphs)
 - similar trees have many common binary branches
 - complexity $O(n \log n)$ time
- *pq*-Gram Distance
 - lower bound for the **fanout weighted** tree edit distance
 - trees are split into *pq*-grams (small subtrees)
 - similar trees have many common *pq*-grams
 - complexity $O(n \log n)$ time

-  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.
-  Richard M. Karp and Michael O. Rabin.
Efficient randomized pattern-matching algorithms.
IBM Journal of Research and Development, 31(2):249–260, March 1987.