Outline

1 Binary Branch Distance
   - Binary Representation of a Tree
   - Binary Branches
   - Lower Bound for the Edit Distance
   - Complexity

2 The \(pq\)-Gram Distance
   - Definition
   - Computing the \(pq\)-Grams
   - Fanout Weighting and Lower Bound
   - Experiments

3 Conclusion

Example: Binary Tree

- Two different binary trees: \(T_B = (N, E_l, E_r)\)
  - \(T_{B1} = (\{a, b, c, d, e, f, g\}, \{(a, b), (b, c), (d, e), (e, f), (d, e), (e, g)\})\)
  - \(T_{B2} = (\{a, b, c, d, e, f, g\}, \{(a, b), (b, c), (e, f), (d, e), (e, g)\})\)

- A full binary tree:
Binary Branch Distance

Binary Representation of a Tree

- **Binary tree transformation:**
  1. link all neighboring siblings in a tree with edges
  2. delete all parent-child edges except the edge to the first child

  Transformation maintains
  - label information
  - structure information

- Original tree can be reconstructed from the binary tree:
  - a *left edge* represents a parent-child relationship in the original tree
  - a *right edge* represents a right-sibling relationship in the original tree

Example: Binary Tree Transformation

Represent tree $T$ as a binary tree:

$$T \rightarrow \text{binary representation of } T$$

Normalized Binary Tree Representation

- We extend the binary tree with null nodes $\epsilon$ as follows:
  - a null node for each missing left child of a non-null node
  - a null node for each missing right child of a non-null node

- Note: Leaf nodes get two null-children.
- The resulting normalized binary representation
  - is a full binary tree
  - all non-null nodes have two children
  - all leaves are null-nodes (and all null-nodes are leaves)

Example: Normalized Binary Tree

Transforming $T$ to the normalized binary tree $B(T)$:

$$T \rightarrow B(T)$$
A binary branch $BiB(v)$ is a subtree of the normalized binary tree $B(T)$ consisting of a non-null node $v$ and its two children.

Example:

$BiB(a) = \{(a, b, e), \{(a, b)\}, \{(a, e)\}\}$

$BiB(d) = \{(d, c_1, e_2), \{(d, c_1)\}, \{(d, e_2)\}\}$  

Although the two null nodes have identical labels ($\epsilon$), they are different nodes. We emphasize this by showing their IDs in subscript.

Example: Binary Branches of Trees and Datasets

Binary branch sets:

$BiB(T)$ is the set of all binary branches of $B(T)$

$BiB(S) = \bigcup_{T \in S} BiB(T)$ is the set of all binary branches of dataset $S$

Binary branches can be serialized as strings:

$BiB(v) = \{(v, a, b), \{(v, a)\}, \{(v, b)\}\} \rightarrow \lambda(v) \circ \lambda(a) \circ \lambda(b)$

we can sort these strings ($\epsilon > \lambda(v)$ for all non-null nodes $v$)

Note:

- nodes are unique in the tree, thus binary branches are unique
- labels are not unique, thus the serialized binary branches are not unique

The binary branch vector $BBV(T)$ is a representation of the binary branch set $BiB(T)$

Construction of the binary branch vector $BBV(T)$:

- serialize and sort all binary branches of $BiB(S)$
- $b_i$ is the $i$-th serialized binary branch in sort order
- $BBV(T)[i]$ is the number of binary branches in $B(T)$ that serialize to $b_i$

Note: $BBV(T)[i]$ is zero if $b_i$ does not appear in $BiB(T)$
Example: Binary Branches of Trees and Datasets

 Binary Branch Distance

\[ S = \{T_1, T_2\} \text{ is the data set} \]

\[ BiB_{sort}(S) \text{ is the sorted set of serialized strings of } BiB(S) \]

\[ BBV(T) \text{ is the binary branch vector of } T \]

The normalized binary tree representations are:

\[
\begin{align*}
BiB_{sort}(S) & = \{abc, bcb, bcc, bce, ced, dcb, dec, dce, dce\} \\
BBV(T_1) & = (1, 1, 0, 1, 0, 2, 0, 0, 2, 1) \\
BBV(T_2) & = (1, 0, 1, 0, 1, 2, 1, 1, 0, 2)
\end{align*}
\]

We compute the binary branch distance between \( T_1 \) and \( T_2 \):

\[
\delta_B(T_1, T_2) = \sum_{i=1}^{k} |b_i - b'_i|.
\]
Example: Binary Branch Distance

- The binary branch vectors of $T_1$ and $T_2$ are:
  
  $$
  \begin{align*}
  BiB_{sort}(S) & = \{abc, bcb, bcc, bce, ccc, dcb, dce, ecc\} \\
  BBV(T_1) & = \{1, 1, 0, 1, 0, 2, 0, 0, 2, 1\} \\
  BBV(T_2) & = \{1, 0, 1, 0, 1, 2, 1, 1, 0, 2\}
  \end{align*}
  $$

- The binary branch distance is
  
  $$
  \delta_B(T_1, T_2) = \sum_{i=1}^{10} |b_{1,i} - b_{2,i}|
  \leq |1 - 1| + |1 - 0| + |0 - 1| + |1 - 0| + |0 - 1| + |
  2 - 2| + |0 - 1| + |0 - 1| + |2 - 0| + |1 - 2|
  = 9,
  $$

where $b_{1,i}$ and $b_{2,i}$ are the $i$-th dimension of the vectors $BBV(T_1)$ and $BBV(T_2)$, respectively.

Proof Sketch: Illustration for Rename

- transform $T_1$ to $T_2$: $\text{ren}(c, x)$
  
  $$
  \begin{align*}
  & b \quad c \\
  & a \quad b \\
  & g \quad e \\
  & e \quad f \\
  & x \quad g
  \end{align*}
  $$

- binary trees $B(T_1)$ and $B(T_2)$
  
  $$
  \begin{align*}
  & b \quad a \\
  & a \quad b \\
  & e \quad e \\
  & e \quad e \\
  & x \quad e \\
  & e \quad e \\
  & e \quad e \\
  & e \quad e \\
  & e \quad e
  \end{align*}
  $$

- Two binary branches ($bce$, $cog$) exist only in $B(T_1)$
- Two binary branches ($bex$, $xeg$) exist only in $B(T_2)$
- $\delta_b(T_1, T_2) = 1$ (1 rename)
- $\delta_B(T_1, T_2) = 4$ (4 binary branches different)

Proof Sketch: Illustration for Insert

- transform $T_1$ to $T_2$: $\text{ins}(x, a, 2, 3)$
  
  $$
  \begin{align*}
  & b \quad e \\
  & a \quad f \\
  & g \quad b \\
  & a \quad x \\
  & e \quad f \\
  & x \quad e
  \end{align*}
  $$

- binary trees $B(T_1)$ and $B(T_2)$
  
  $$
  \begin{align*}
  & b \quad a \\
  & a \quad b \\
  & e \quad e \\
  & e \quad e \\
  & e \quad e \\
  & e \quad e \\
  & e \quad e \\
  & e \quad e \\
  & e \quad e
  \end{align*}
  $$

- Two binary branches ($bce$, $feg$) exist only in $B(T_1)$
- Tree binary branches ($bex$, $fe\epsilon$, $xeg$) exist only in $B(T_2)$
- $\delta_b(T_1, T_2) = 1$ (1 insertion)
- $\delta_B(T_1, T_2) = 5$ (5 binary branches different)
Proof Sketch

- In general it can be shown that
  - Rename changes at most 4 binary branches
  - Insert changes at most 5 binary branches
  - Delete changes at most 5 binary branches

Each edit operation changes at most 5 binary branches, thus

\[ \delta_B(T, T') \leq 5 \times \delta_t(T, T'). \]

Complexity

- Note: Improvement using a hash function:
  - we assume a hash function that maps the \(O(Nn)\) binary branches to \(O(Nn)\) buckets without collision
  - we do not sort \(BiB(S)\)
  - position \(i\) in the vector \(BBV(T)\) is computed using the hash function
  - \(O(Nn)\) time (instead of \(O(Nn\log(Nn))\)) and \(O(Nn)\) space

In the following we consider only the sort-algorithm with \(O(Nn\log(Nn))\) runtime.

Complexity: Binary Branch Vector

- Given a set \(S\) with \(N\) trees of average size \(n\).
- Construction of the binary branch vectors \(BBV(T)\) for all \(T \in S\):
  1. compute the binary branches of all trees in \(S\), \(BiB(S)\):
     - \(O(Nn)\) time and space (traverse all trees in \(S\))
  2. sort serialized binary branches of \(BiB(S)\) and store them in \(BiB_{sort}(S)\):
     - \(O(Nn\log(Nn))\) time and \(O(Nn)\) space
  3. construct \(BBV(T)\):
     - (a) traverse all trees again and compute their binary branches:
       - \(O(Nn)\) time and space
     - (b) for each binary branch find position \(i\) in \(BiB_{sort}(S)\):
       - \(O(Nn\log(Nn))\) time (binary search in \(V\) for \(Nn\) binary branches)
     - (c) \(BBV(T)[i]\) is incremented: \(O(1)\)

The overall complexity is \(O(Nn\log(Nn))\) time and \(O(Nn)\) space.

Complexity: Distance

- Given the binary branch vectors of a set of \(N\) trees (tree size \(n\)).
- Computing the distance between two of the \(N\) trees:
  - the binary branch vectors are of size \(O(Nn)\)
  - computing the distance has \(O(Nn)\) time complexity (subtracting two binary branch vectors)
- Complexity for a set that contains only two trees (\(N = 2\)):
  - constructing the binary branch vectors: \(O(n\log(n))\) time, \(O(n)\) space
  - computing the distance: \(O(n)\) time and space
  - overall complexity: \(O(n\log(n))\) time, \(O(n)\) space
**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-Extended Tree**

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

  ![Example Tree](attachment://example_tree.png)

- **Solution:** Extend tree \( T \) with dummy nodes (●):
  - \( p - 1 \) 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** \( T^{pq} \).
Definition: **pq-Gram**

**Definition (pq-Gram)**

Let $p > 0$, $q > 0$. A pq-gram, $g$, of a tree $T$ with anchor node $a \in N(T)$ is a subtree of the extended tree $T^{pq}$ that is composed of the following nodes:

- $p$ nodes $a_{p-1}, \ldots, a_1, a$ ($a_i$ is the ancestor of node $a$ at distance $i$)
- $q$ nodes $c_i, \ldots, c_{i+q-1}$ ($c_i$ is the $i$-th child of node $a$)

- **Stem**: the nodes $a_{p-1}, \ldots, a_1, a$ form the stem of the pq-gram $g$.
- **Base**: the nodes $c_i, \ldots, c_{i+q-1}$ form the base of the pq-gram $g$.

**Definition (pq-Gram Profile)**

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

**pq-Gram Index**

**Definition (pq-Gram Index)**

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

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

- **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

**Label-Tuples**

- **Linear encoding** of a pq-gram $g$ with anchor node $a$:
  (traverse pq-gram in preorder)
  $$\begin{align*}
  a_{p-1} & \ldots \\
  c_2 & \ldots \\
  c_i & \ldots \\
  c_{i+q-1}
  \end{align*}$$

  - Label-tuple: tuple of the pq-grams node labels
  $$\lambda(g) = (\lambda(v_1), \ldots, \lambda(v_{p+q}))$$
  for the pq-gram $g = (v_1, \ldots, v_{p+q})$.

**The pq-Gram Distance**

**Definition (pq-Gram Distance)**

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

$$\delta_g(T, T') = |I(T) \cup I(T')| - 2|I(T) \cap I(T')|$$

- **Metric normalization** to $[0..1]$: $\delta'_g(T, T') = \frac{\delta_g(T, T')}{2|I(T)\cup I(T')|} - |I(T)\cap I(T')|$
- **Pseudo-metric** properties hold for normalization [ABG]:
  - self-identity: $x = y \nRightarrow \delta_g(x, y) = 0$
  - 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:

  a
  b
  b
  c
  c

  $A$
  $B$
  $B$
  $C$
  $C$
### CREATEINDEX(T, r, I, stem, p, q)

- **stem**: shift(stem, λ(r))
- **base**: shift register of size q (filled with *)

**Algorithm**:  
1. If r is a leaf then  
   - I := I ∪ {stem ◦ base}
2. Else  
   - For each child c (from left to right) of r do  
     - base := shift(base, λ(c))
     - I := I ∪ {stem ◦ base}
   - I := CREATEINDEX(T, c, I, stem, p, q)
3. For k := 1 to q - 1 do  
   - base := shift(base, *)
   - I := I ∪ {stem ◦ base}

**Return**: I

---

### Main Memory Algorithm for a Single Tree (II)

- **Input** of CREATEINDEX(T, r, I, stem, p, q):
  - a subtree of T rooted in r
  - the pq-gram index I computed so far
  - the stem stem of r's parent
  - the parameters p and q

- **Output** of CREATEINDEX(T, r, I, stem, p, q):
  - pq-gram index including the input index 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

### pq-Gram-Index(T, p, q)

- **pq-Gram-Index(T, p, q)** computes the pq-gram index for a complete tree T:

**Algorithm**:  
1. **stem**: shift register of size p (filled with *)
2. **I**: empty index
3. **I = CREATEINDEX(T, root(T), I, stem, p, q)**
4. **Return**: I

---

### Complexity of the pq-Gram Index Algorithm

**Theorem (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 CREATEINDEX() processes one node, and each node is processed exactly once.
The pq-Gram Distance

Size of the pq-Gram Index

Theorem (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

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

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, 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: \( l + i(q - 1) + (l + i - 1) = 2l + qi - 1. \)

Motivation: Unit Cost Model Not Always Intuitive

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: \( \alpha(w \rightarrow w') = (f + f')/2 + c \)

- 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.

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
The pq-Gram Distance

**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 [ABG] (ACM Transactions on Database Systems).

---

**Experiments**

**Sensitivity to Structure Change — Leaf**

- **Cost of leaf change** → depends only on $q$
- **Experiment:**
  - delete leaf nodes
  - measure normalized pq-gram distance

**Sensitivity to Structure Change — Non-Leaf**

- **Cost for non-leaf change** → controlled by $p$
- **Experiment:**
  - delete non-leaf nodes
  - measure normalized pq-gram distance

**Influence of $p$ and $q$ on Scalability**

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

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

- pq-gram distance → 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 $5 \times 10^5$ nodes
  - measure wall clock time

<table>
<thead>
<tr>
<th>number of nodes (n)</th>
<th>time [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100000</td>
<td>100</td>
</tr>
<tr>
<td>200000</td>
<td>200</td>
</tr>
<tr>
<td>300000</td>
<td>300</td>
</tr>
<tr>
<td>400000</td>
<td>400</td>
</tr>
<tr>
<td>500000</td>
<td>500</td>
</tr>
</tbody>
</table>

- Binary Branch Distance
  - lower bound of the unit cost tree edit distance
  - complexity $O(n \log n)$ time

- pq-Gram Distance
  - lower bound for the fanout weighted tree edit distance
  - trees are represented by small subtrees
  - similar trees have many common subtrees
  - complexity $O(n \log n)$ time

What’s Next?

- Distances for Data Centric XML:
  - unordered tree edit distance
  - windowed pq-gram distance

Reference:

Nikolaus Augsten, Michael Böhlen, and Johann Gamper.
Approximate matching of hierarchical data using pq-grams.
ACM Transactions on Database Systems (TODS).
To appear.

Rui Yang, Panos Kalnis, and Anthony K. H. Tung.
Similarity evaluation on tree-structured data.