Similarity Search
The Binary Branch Distance

Nikolaus Augsten
Free University of Bozen-Bolzano
Faculty of Computer Science
DIS
Unit 10 – May 17, 2012

Outline

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

Binary Tree

- In a binary tree
  - each node has at most two children;
  - left child and right child are distinguished:
    a node can have a right child without having a left child;

Notation: \( T_B = (N, E_l, E_r) \)
- \( T_B \) denotes a binary tree
- \( N \) are the nodes of the binary tree
- \( E_l \) and \( E_r \) are the edges to the left and right children, respectively

Full binary tree:
- binary tree
- each node has exactly zero or two children.
**Example: Binary Tree**

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

- A full binary tree:

**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

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

\[
\begin{align*}
T & \rightarrow B(T) \\
\begin{array}{c}
\text{a} \\
\text{b}
\end{array} & \leftarrow \begin{array}{c}
\text{a} \\
\text{b} \\
\text{c} \\
\text{d} \\
\text{e}
\end{array} \\
\end{align*}
\]

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:
\[
\begin{align*}
BiB(a) &= \{(a, b, \epsilon), \{(a, b), \{(a, \epsilon)}\} \\
BiB(d) &= \{(d, \epsilon_1, \epsilon_2), \{(d, \epsilon), \{(d, \epsilon_2)}\} \quad 1
\end{align*}
\]

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

Binary branches can be serialized as strings:
- \(BiB(v) = \{(v, a, b), \{(v, a), \{(a, b)}\} \rightarrow \lambda(v) \circ \lambda(a) \circ \lambda(b)\)
- we can sort these strings \(\epsilon > \lambda(v)\) for all non-null nodes v

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
- \(BiB_{sort}(S)\) is the vector of sorted serialized strings of \(BiB(S)\)

Note:
- nodes are unique in the tree, thus binary branches are unique
- labels are not unique, thus the serialized binary branches are not unique
**Example: Binary Branches of Trees and Datasets**

- $T_1$: $a \rightarrow b \rightarrow c$, $b \rightarrow d \rightarrow e$
- $T_2$: $a \rightarrow b \rightarrow c$, $b \rightarrow d \rightarrow e$

$BiB(c_1) \neq BiB(c_4)$:
- $BiB(c_1) = \{(c_1, c_2), \{(c_1, d_2), \{(c_1, d_3), \{(c_1, d_4), \{(c_1, d_5), \{(c_4, d_6), \{(c_4, d_7)\})\})\})\}$
- $BiB(c_4) = \{(c_4, e_1), \{(c_4, e_2), \{(c_4, e_3), \{(c_4, e_4), \{(c_4, e_5)\})\})\}$

Serialization of both, $BiB(c_1)$ and $BiB(c_2)$, is identical: ‘ccd’

- Sorted vector of serialized strings of $BiB(S)$, where $S = \{T_1, T_2\}$:
  - $BiB_{sort}(S) = (abc, bc, bcc, bce, be, cde, d, db, dce, de, e, e, e)$

**Example: Binary Branch Vectors**

- $S = \{T_1, T_2\}$ is the data set
- $BiB_{sort}(S)$ is the vector of sorted serialized strings of $BiB(S)$
- $BBV(T_i)$ is the binary branch vector of $T_i$

The vector of serialized strings and the binary branch vectors are:

<table>
<thead>
<tr>
<th>$BiB_{sort}(S)$</th>
<th>$BBV(T_1)$</th>
<th>$BBV(T_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc bcb bcc bce ber ccd dcb dce dce ece</td>
<td>1 1 0 1 0 2 0 0 2 1</td>
<td>1 0 1 0 1 2 1 1 0 2</td>
</tr>
</tbody>
</table>

**Binary Branch Vector**

- The binary branch vector $BBV(T)$ is a representation of the binary branch set $BiB(T)$
- **Construction** of the binary branch vector $BBV(T)$:
  - compute $BiB_{sort}(S)$ (serialize and sort $BiB(S)$)
  - $b_i$ is the $i$-th serialized binary branch in sort order ($b_i = BiB_{sort}(S)[i]$)
  - $BBV(T)[i]$ 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 $BiB(T)$

**Outline**

1. **Binary Branch Distance**
   - Binary Representation of a Tree
   - Binary Branches
   - Lower Bound for the Edit Distance
   - Complexity
Definition (Binary Branch Distance)

Let $BBV(T) = (b_1, \ldots, b_k)$ and $BBV(T') = (b'_1, \ldots, b'_k)$ be binary branch vectors of trees $T$ and $T'$, respectively. The binary branch distance of $T$ and $T'$ is

$$\delta_B(T, T') = \sum_{i=1}^{k} |b_i - b'_i|.$$  

Intuition: We count the binary branches that do not match between the two trees.

Example: Binary Branch Distance

We compute the binary branch distance between $T_1$ and $T_2$:

1. The normalized binary tree representations are:

   - $B(T_1)$
   - $B(T_2)$

2. The binary branch vectors of $T_1$ and $T_2$ are:

   - $BBV(T_1) = [1, 1, 0, 1, 0, 2, 0, 0, 2, 1]$
   - $BBV(T_2) = [1, 0, 1, 0, 1, 2, 1, 1, 0, 2]$

3. The binary branch distance is

   $$\delta_B(T_1, T_2) = \sum_{i=1}^{10} |b_{1,i} - b_{2,i}| = |1 - 1| + |1 - 0| + |0 - 1| + |1 - 0| + |0 - 1| + |2 - 2| + |0 - 1| + |0 - 1| + |2 - 0| + |1 - 2| = 9,$$

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

Let $T$ and $T'$ be two trees. If the tree edit distance between $T$ and $T'$ is $\delta_t(T, T')$, then the binary branch distance between them satisfies

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

**Proof (Sketch — Full Proof in [YKT05])**

- Each node $v$ appears in at most two binary branches.
- Rename: Renaming a node causes at most two binary branches in each tree to mismatch. The sum is 4.
- Similar rational for insert and its complementary operation delete (at most 5 binary branches mismatch).

**Proof Sketch: Illustration for Rename**

- transform $T_1$ to $T_2$: $ren(c, x)$
  - $T_1$:
    - $a$ - $b$ - $c$ - $g$
    - $a$ - $b$ - $x$
  - $T_2$:
    - $a$ - $b$ - $g$
    - $a$ - $b$ - $x$
- Two binary branches ($bce, ceg$) exist only in $B(T_1)$
- Two binary branches ($bcx, xeg$) exist only in $B(T_2)$
- $\delta_t(T_1, T_2) = 1$ (1 rename)
- $\delta_B(T_1, T_2) = 4$ (4 binary branches different)

**Proof Sketch: Illustration for Insert**

- transform $T_1$ to $T_2$: $ins(x, a, 2, 3)$
  - $T_1$:
    - $a$ - $b$ - $f$ - $g$
    - $a$ - $x$
  - $T_2$:
    - $a$ - $b$ - $g$
    - $a$ - $x$
- Two binary branches ($bce, feg$) exist only in $B(T_1)$
- Tree binary branches ($bcx, fe, xeg$) exist only in $B(T_2)$
- $\delta_t(T_1, T_2) = 1$ (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

$$\delta_B(T, T') \leq 5 \times \delta_t(T, T').$$
1 Binary Branch Distance

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

**Complexity: Binary Branch Distance**

- Compute the distance between two trees of size $O(n)$:
  \( S = \{T_1, T_2\}, \ n = \max\{|T_1|, |T_2|\}\)
- Construction of the binary branch vectors $BBV(T_1)$ and $BBV(T_2)$:
  1. $BiB(S)$ – compute the binary branches of $T_1$ and $T_2$:
     - $O(n)$ time and space (traverse $T_1$ and $T_2$)
  2. $BiB_{sort}(S)$ – sort serialized binary branches of $BiB(S)$:
     - $O(n \log n)$ time and $O(n)$ space
  3. construct $BBV(T_1)$ and $BBV(T_2)$:
     - (a) recompute all binary branches: $O(n)$ time and space
     - (b) for each binary branch find position $i$ in $BiB_{sort}(S)$: $O(n \log n)$ time
       - (binary search in $BiB_{sort}(S)$ for $n$ binary branches)
     - (c) $BBV(T)[i]$ is incremented: $O(1)$
- Computing the distance:
  - the two binary branch vectors are of size $O(n)$
  - computing the distance has time complexity $O(n)$
    (subtracting two binary branch vectors)
- The overall complexity is $O(n \log n)$ time and $O(n)$ space.

**Improving the Time Complexity with a Hash Function**

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

**Complexity for Similarity Joins**

- Join two sets with $N$ trees each (tree size: $n$):
  - Compute Binary Branch Vectors (BBVs):
    $O(Nn \log (Nn))$ time, $O(N^2n)$ space
    - BBVs are of size $O(Nn)$
      - time: sort $O(Nn)$ binary branches / $O(Nn)$ binary searches in BBVs
      - space: $O(N)$ BBVs must be stored
  - Compute Distances: $O(N^3n)$ time
    - computing the distance between two trees has $O(Nn)$ time complexity
      (subtracting two binary branch vectors)
    - $O(N^2n)$ distance computations required
  - Overall Complexity: $O(N^3n + Nn \log n)^2$ time and $O(N^2n)$ space

$$O(N^3n + Nn \log Nn) = O(N^3n + Nn \log N + Nn \log n) = O(N^3n + Nn \log n)$$