**Outline**

1. **Tree Edit Distance**
   - Preliminaries and Definition
   - Forests Distance and Recursive Formula
   - Second Recursive Formula
   - The Tree Edit Distance Algorithm
   - Example: Tree Edit Distance Computation
   - Complexity of the Tree Edit Distance Algorithm

**Edit Operations**

- We assume **ordered, labeled trees**
- **Rename node**: $\text{ren}(v, l')$  
  - change label $l$ of $v$ to $l' \neq l$
- **Delete node**: $\text{del}(v)$ ($v$ is not the root node)
  - remove $v$
  - connect $v$'s children directly to $v$'s parent node (preserving order)
- **Insert node**: $\text{ins}(v, p, k, m)$
  - remove $m$ consecutive children of $p$, starting with the child at position $k$, i.e., the children $c_k, c_{k+1}, \ldots, c_{k+m-1}$
  - insert $c_k, c_{k+1}, \ldots, c_{k+m-1}$ as children of the new node $v$ (preserving order)
  - insert new node $v$ as $k$-th child of $p$
- Insert and delete are **inverse** edit operations (i.e., insert undoes delete and vice versa)
**Example: Edit Operations**

Represent edit operation as node pair \((a, b) \neq (\varepsilon, \varepsilon)\) (written also as \(a \rightarrow b\), \(\varepsilon\) is the null node)
- \(a \rightarrow \varepsilon\): delete \(a\)
- \(\varepsilon \rightarrow b\): insert \(b\)
- \(a \rightarrow b\): rename \(a\) to \(b\)

Cost function \(\alpha(a \rightarrow b)\):
- assign to each edit operation a non-negative real cost can be different for different nodes
- we use constant costs \(\omega_{ins}, \omega_{del}, \omega_{ren}\)

We constrain \(\alpha\) to be a distance metric:
1. triangle inequality: \(\alpha(a \rightarrow b) + \alpha(b \rightarrow c) \geq \alpha(a, c)\)
2. symmetry: \(\alpha(a \rightarrow b) = \alpha(b \rightarrow a)\)
3. identity: \(\alpha(a \rightarrow b) = 0 \iff \lambda(a) = \lambda(b)\)

**Postorder Traversal**

- Postorder traversal of an ordered tree:
  - traverse subtrees rooted in children of current node (from left to right) in postorder
  - visit current node
- Example: postorder = \((f, e, d, c, b, a)\)

Observations: The postorder number of a node is larger than
- the postorder numbers of all its descendants
- the postorder numbers of all its left siblings
**Subtrees and Subforests**

- A **subtree** $T'$ of $T$ is a tree that consists of:
  - a subset of the nodes of $T$: $N(T') \subseteq N(T)$
  - all edges in $T$ that connect these nodes: $E(T') \subseteq E(T)$

- **Ordered Forests**:
  - a forest is a set of trees
  - an *ordered* forest is a sequence of trees

- **Ordered Subforests** of a tree $T$:
  - formed by subtrees of $T$ with disjoined nodes
  - subtrees ordered by the postorder number in $T$ of their root

**Notation I/II**

- We use the following **notation**:
  - $T[i]$ is the $i$-th node of $T$ in **postorder** (we say: $T[i]$ is node $i$ of $T$)
  - $T[i..j]$ is the subtree formed by the nodes $T[i]$ to $T[j]$  
  - $l(i)$ is the left-most leaf descendant of node $T[i]$
  - $desc(T[i])$ is the set of all descendants of $T[i]$ including $T[i]$ itself (elements of $desc(T[i])$ are usually denoted with $d_i$)

- **Node identifiers**:
  - we assume that the node IDs correspond to their postorder number
  - we refer to a node simply by its ID, if the context is clear

**Notation II/II**

- $T[l(i)..i]$ is the **subtree rooted in** $T[i]$, i.e., the subtree consisting of node $i$ and all its descendants
- A **special subforest** of the form
  $$T[l(i)..d_i], \quad (d_i \in desc(T[i]))$$
  is a **prefix** of the subtree rooted in $T[i]$.

- **Observations**:
  - If a node $k$ is in $T[l(i)..d_i]$, also all its descendants are in $T[l(i)..d_i]$.
  - A (sub)tree with $n$ nodes has $n$ prefixes.
**Tree Edit Distance**

**Preliminaries and Definition**

**Example: Subtrees and Subforests**

- **Example tree:**
  ```
  f6
  d4 /\ e5
  a1 \  \
  c3   b2
  ```

- **Descendants:**
  \[ \text{desc}(T[4]) = \{T[1], T[2], T[3], T[4]\} \]

- **Left-most leaf descendants:**
  \[ l(1) = l(4) = l(6) = T[1] \]

- **Some ordered subforests of the form**
  \[ T[l(i)\ldots d_i], d_i \in \text{desc}(i) \]:
  - \[ T[l(4)\ldots3] \]
  - \[ T[l(4)\ldots4] \]
  - \[ T[l(6)\ldots5] \]
  - \[ T[l(5)\ldots5] \]

**Edit Mapping**

**Definition (Edit Mapping)**

An edit mapping \( M \) between \( T_1 \) and \( T_2 \) is a set of node pairs that satisfy the following conditions:

1. \( (a, b) \in M \Rightarrow a \in N(T_1), b \in N(T_2) \)
2. For any two pairs \( (a, b) \) and \( (x, y) \) of \( M \):
   - (i) \( a = x \Leftrightarrow b = y \) (one-to-one condition)
   - (ii) \( a \) is to the left of \( x \) \( \Leftrightarrow \) \( b \) is to the left of \( y \) (order condition)
   - (iii) \( a \) is an ancestor of \( x \) \( \Leftrightarrow \) \( b \) is an ancestor of \( y \) (ancestor condition)
3. Optional: \( a = \text{root}(T_1) \) and \( b = \text{root}(T_1) \) \( \Rightarrow \) \( (a, b) \in M \) (forbid deleting the root node)

\( i.e., a \) precedes \( x \) in both preorder and postorder

**Example: Mapping**

\[ M = \{(T_1[6], T_2[6]), (T_1[5], T_2[5]), (T_1[4], T_2[3]), (T_1[1], T_2[1]), (T_1[2], T_2[2])\} \]

- \( T_1[3] \) is deleted
- \( T_2[4] \) is inserted
- no proper rename (only rename to the same label with cost 0)

**Alternative definition of the tree edit distance** \( \text{ted}(T_1, T_2) \):

\[ \text{ted}(T_1, T_2) = \min\{\alpha(M) \mid M \text{ is a mapping from } T_1 \text{ to } T_2\} \]

**Cost of the mapping**

\[ \alpha(M) = \sum_{(a,b) \in M} \alpha(a \rightarrow b) + \sum_{a \in D} \alpha(a \rightarrow \varepsilon) + \sum_{b \in I} \alpha(\varepsilon \rightarrow b), \]

where \( D \) and \( I \) are the nodes of \( T_1 \) and \( T_2 \), respectively, not touched by a line in \( M \).
Forest Distance

Definition (Forest Distance)
The forest distance between two ordered forests is the minimum cost sequence of node edit operations (node deletion, node insertion, node rename) that transforms one forest into the other.

- **Edit mapping and edit operations** in a forest:
  - Each tree in the forest has a root node.
  - We imagine a dummy node that is the parent of all these root nodes.
  - The sibling order in the imaginary tree is the tree order in the forest.
  - The dummy node connects the forest to become a tree.
  - Then all edit operations and edit mappings valid between two imaginary trees are valid also between the respective forests.
  - The tree edit distance is a special case of the forest distance, where the forest has the form $T[l(i)\ldots i]$, i.e. it consists of a single tree.

Lemma (First Recursive Formula)

Given two trees $T_1$ and $T_2$, $i \in N(T_1)$ and $d_i \in \text{desc}(i)$, $j \in N(T_2)$ and $d_j \in \text{desc}(j)$, then:

(i) \quad \text{fdist}(\emptyset, \emptyset) = 0

(ii) \quad \text{fdist}(T_1[l(i)\ldots,d_i], \emptyset) = \text{fdist}(T_1[l(i)\ldots,d_i - 1], \emptyset) + \omega_{\text{del}}

(iii) \quad \text{fdist}(\emptyset, T_2[l(j)\ldots,d_j]) = \text{fdist}(\emptyset, T_2[l(j)\ldots,d_j - 1]) + \omega_{\text{ins}}

Proof.

Case (i) requires no edit operation. In cases (ii), the distance corresponds to the cost of deleting all nodes in $T_1[l(i)\ldots,d_i]$. In cases (iii), the distance corresponds to the cost of inserting all nodes in $T_2[l(j)\ldots,d_j]$. 

Lemma (Empty Forest [ZS89, AG97])

Given two trees $T_1$ and $T_2$, $i \in N(T_1)$ and $d_i \in \text{desc}(i)$, $j \in N(T_2)$ and $d_j \in \text{desc}(j)$, then:

\[
\begin{align*}
(i) & \quad \text{fdist}(\emptyset, \emptyset) = 0 \\
(ii) & \quad \text{fdist}(T_1[l(i)\ldots,d_i], \emptyset) = \text{fdist}(T_1[l(i)\ldots,d_i - 1], \emptyset) + \omega_{\text{del}} \\
(iii) & \quad \text{fdist}(\emptyset, T_2[l(j)\ldots,d_j]) = \text{fdist}(\emptyset, T_2[l(j)\ldots,d_j - 1]) + \omega_{\text{ins}}
\end{align*}
\]

Proof.

Case (i) requires no edit operation. In cases (ii), the distance corresponds to the cost of deleting all nodes in $T_1[l(i)\ldots,d_i]$. In cases (iii), the distance corresponds to the cost of inserting all nodes in $T_2[l(j)\ldots,d_j]$. 

First Recursive Formula: Forest Distance

Lemma (First Recursive Formula)

Given two trees $T_1$ and $T_2$, $i \in N(T_1)$ and $d_i \in \text{desc}(i)$, $j \in N(T_2)$ and $d_j \in \text{desc}(j)$, then:

\[
\text{fdist}(T_1[l(i)\ldots,d_i], T_2[l(j)\ldots,d_j])) = \min \left\{ \begin{array}{l}
\text{fdist}(T_1[l(i)\ldots,d_i - 1], T_2[l(j)\ldots,d_j]) + \omega_{\text{del}} \\
\text{fdist}(T_1[l(i)\ldots,d_i], T_2[l(j)\ldots,d_j - 1]) + \omega_{\text{ins}} \\
\text{fdist}(T_1[l(i)\ldots,d_i], T_2[l(j)\ldots,d_j]) + \omega_{\text{ren}}
\end{array} \right\}
\]

Proof.

Case (i) requires no edit operation. In cases (ii), the distance corresponds to the cost of deleting all nodes in $T_1[l(i)\ldots,d_i]$. In cases (iii), the distance corresponds to the cost of inserting all nodes in $T_2[l(j)\ldots,d_j]$. 

Second Recursive Formula

Recursive Formula: Distance to the Empty Forest

Lemma (Empty Forest)

Given a forest $F$, the forest distance between $F$ and the empty forest $\emptyset$ is the minimum cost sequence of node edit operations that transforms $F$ into $\emptyset$. 

Proof.

Case (i) requires no edit operation. In cases (ii), the distance corresponds to the cost of deleting all nodes in the forest $F$. In cases (iii), the distance corresponds to the cost of inserting all nodes into the empty forest $\emptyset$. 

Tree Edit Distance

Outline

1. Tree Edit Distance
   - Preliminaries and Definition
   - Forests Distance and Recursive Formula
   - Second Recursive Formula
   - The Tree Edit Distance Algorithm
   - Example: Tree Edit Distance Computation
   - Complexity of the Tree Edit Distance Algorithm

Recursive Formula: Distance to the Empty Forest

Lemma (Empty Forest [ZS89, AG97])

Given two trees $T_1$ and $T_2$, $i \in N(T_1)$ and $d_i \in \text{desc}(i)$, $j \in N(T_2)$ and $d_j \in \text{desc}(j)$, then:

(i) \quad \text{fdist}(\emptyset, \emptyset) = 0

(ii) \quad \text{fdist}(T_1[l(i)\ldots,d_i], \emptyset) = \text{fdist}(T_1[l(i)\ldots,d_i - 1], \emptyset) + \omega_{\text{del}}

(iii) \quad \text{fdist}(\emptyset, T_2[l(j)\ldots,d_j]) = \text{fdist}(\emptyset, T_2[l(j)\ldots,d_j - 1]) + \omega_{\text{ins}}

Proof.

Case (i) requires no edit operation. In cases (ii), the distance corresponds to the cost of deleting all nodes in $T_1[l(i)\ldots,d_i]$. In cases (iii), the distance corresponds to the cost of inserting all nodes in $T_2[l(j)\ldots,d_j]$. 

First Recursive Formula: Forest Distance

Lemma (First Recursive Formula)

Given two trees $T_1$ and $T_2$, $i \in N(T_1)$ and $d_i \in \text{desc}(i)$, $j \in N(T_2)$ and $d_j \in \text{desc}(j)$, then:

\[
\text{fdist}(T_1[l(i)\ldots,d_i], T_2[l(j)\ldots,d_j])) = \min \left\{ \begin{array}{l}
\text{fdist}(T_1[l(i)\ldots,d_i - 1], T_2[l(j)\ldots,d_j]) + \omega_{\text{del}} \\
\text{fdist}(T_1[l(i)\ldots,d_i], T_2[l(j)\ldots,d_j - 1]) + \omega_{\text{ins}} \\
\text{fdist}(T_1[l(i)\ldots,d_i], T_2[l(j)\ldots,d_j]) + \omega_{\text{ren}}
\end{array} \right\}
\]

Proof.

Case (i) requires no edit operation. In cases (ii), the distance corresponds to the cost of deleting all nodes in $T_1[l(i)\ldots,d_i]$. In cases (iii), the distance corresponds to the cost of inserting all nodes in $T_2[l(j)\ldots,d_j]$. 

Second Recursive Formula

Complexity of the Tree Edit Distance Algorithm

Example: Tree Edit Distance Computation

Nikolaus Augsten (DIS) Similarity Search Unit 5 – March 29, 2012 17 / 63
Proof.

Let \( M \) be the minimum-cost map between \( T_1[\{i\}..d_i] \) and \( T_2[\{j\}..d_j] \), i.e., the map we are looking for. Then for \( T_1[d_i] \) and \( T_2[d_j] \) there are three possibilities:

1. \( T_1[d_i] \) is not touched by a line in \( M \): \( T_1[d_i] \) is deleted and \( \text{fdist}(T_1[\{i\}..d_i], T_2[\{j\}..d_j]) = \text{fdist}(T_1[\{i\}..d_i - 1], T_2[\{j\}..d_j]) + \omega_{\text{del}} \)

2. \( T_2[d_j] \) is not touched by a line in \( M \): \( T_2[d_j] \) is inserted and \( \text{fdist}(T_1[\{i\}..d_i], T_2[\{j\}..d_j]) = \text{fdist}(T_1[\{i\}..d_i], T_2[\{j\}..d_j - 1]) + \omega_{\text{ins}} \)

3. Both, \( T_1[d_i] \) and \( T_2[d_j] \) are touched by a line in \( M \): We show (by contradiction) that in this case (\( T_1[d_i], T_2[d_j] \)) \( \in M \), i.e., \( T_1[d_i] \) is renamed to \( T_2[d_j] \): Assume \( (T_1[d_i], T_2[d_j]) \in M \) and \( (T_1[d_i], T_2[d_j]) \in M \).
   
   - Case \( T_1[d_i] \) is to the right of \( T_2[d_j] \): By sibling condition on \( M \) also \( T_2[d_j] \) must be to the right of \( T_2[d_j] \). Impossible in \( T_2[\{j\}..d_j] \).
   - Case \( T_1[d_i] \) is proper ancestor of \( T_2[d_j] \): By ancestor condition on \( M \) also \( T_2[d_j] \) must be ancestor of \( T_2[d_j] \). Impossible in \( T_2[\{j\}..d_j] \).

As these three cases express all possible mappings yielding \( \text{fdist}(T_1[\{i\}..d_i], T_2[\{j\}..d_j]) \), we take the minimum of these tree costs.

Example: First Recursive Formula (2/3)

\[
\begin{align*}
\text{T}_1 & \quad \text{T}_2 \\
\quad \quad \quad d_i & \quad \quad \quad c_3 \\
| & \quad \quad | \\
a_1 & a_1 \\
\downarrow & \downarrow \\
b_2 & b_2 \\
\end{align*}
\]

\( (i = 6, d_i = 3) \)

\[
\begin{align*}
\text{T}_1[\{i\}..d_i] & \quad \text{T}_2[\{j\}..d_j] \\
\quad \quad \quad \quad \quad \quad & \quad \quad \quad \quad \quad \quad \\
(3) & \quad \quad \quad \quad \quad \quad \\
\text{fdist}(T_1[\{i\}..d_i], T_2[\{j\}..d_j - 1]) + \omega_{\text{ins}} & \quad + \omega_{\text{ren}} \\
\quad \quad \quad \quad \quad \quad & \quad \quad \quad \quad \quad \quad \\
\text{cost: } 1 + 1 = 2 & \quad \quad \quad \quad \quad \quad \\
\end{align*}
\]

Example: First Recursive Formula (3/3)

\[
\begin{align*}
\text{T}_1 & \quad \text{T}_2 \\
\quad \quad \quad d_i & \quad \quad \quad c_3 \\
| & \quad \quad | \\
a_1 & a_1 \\
\downarrow & \downarrow \\
b_2 & b_2 \\
\end{align*}
\]

\( (j = 6, d_j = 3) \)

\[
\begin{align*}
\text{T}_1[\{i\}..d_i - 1] & \quad \text{T}_2[\{j\}..d_j] \\
\quad \quad \quad \quad \quad \quad & \quad \quad \quad \quad \quad \quad \\
(2) & \quad \quad \quad \quad \quad \quad \\
\text{fdist}(T_1[\{i\}..d_i], T_2[\{j\}..d_j - 1]) + \omega_{\text{ins}} & \quad + \omega_{\text{del}} \\
\quad \quad \quad \quad \quad \quad & \quad \quad \quad \quad \quad \quad \\
\text{cost: } 1 + 1 = 2 & \quad \quad \quad \quad \quad \quad \\
\end{align*}
\]

\[
\begin{align*}
\text{T}_1[\{i\}..d_i - 1] & \quad \text{T}_2[\{j\}..d_j - 1] \\
\quad \quad \quad \quad \quad \quad & \quad \quad \quad \quad \quad \quad \\
(1) & \quad \quad \quad \quad \quad \quad \\
\text{fdist}(T_1[\{i\}..d_i - 1], T_2[\{j\}..d_j - 1]) + \omega_{\text{del}} & \quad + \omega_{\text{del}} \\
\quad \quad \quad \quad \quad \quad & \quad \quad \quad \quad \quad \quad \\
\text{cost: } 1 + 1 = 2 & \quad \quad \quad \quad \quad \quad \\
\end{align*}
\]
**Analogies to the String Case**

- Why is the third formula not (in analogy to the string case):
  \[ \text{fdist}(T_1[1:(i)..di], T_2[1:(j)..dj-1]) + \omega_{\text{ren}} \]

- Consider the previous example:

- \( \text{ren}(c_3,d_3) \) does not transform \( T_1[1:(i)..di] \) to \( T_2[1:(j)..dj] \)

- In fact the mapping \( M = \{ (a_1,a_1), (b_2,b_2), (c_3,d_3) \} \) is not valid:
  - Connect all trees in the forest with a dummy node (\( \bullet \)):
  - As \( d_3 \) is an ancestor of \( a_1 \), \( c_3 \) must be an ancestor of \( a_1 \), which is false.

**Observation**

- \( \text{fdist}(T_1[1:(i)..<di], T_2[1:(j)..<dj]) = \min \begin{cases} \text{fdist}(T_1[1:(i)..di-1], T_2[1:(j)..dj]) + \omega_{\text{del}} \\ \text{fdist}(T_1[1:(i)..di], T_2[1:(j)..dj-1]) + \omega_{\text{ins}} \\ \text{fdist}(T_1[1:(i)..di], T_2[1:(j)..dj]) + \omega_{\text{ren}} \end{cases} \)

- Observation about the First Recursive Formula:
  - \( \text{fdist}(T_1[1:(i)..<di-1], T_2[1:(i)..<dj-1]) \) \( [D] \) is a prefix of the subtree rooted in \( d_j \)
  - all other subforests are prefixes of subtrees rooted in \( i \)
  - \( [D] \) does not fit the scheme (bad for dynamic programming algorithm)

- We derive the Second Recursive Formula:
  - we distinguish two cases (both forests are trees/one forest is not a tree)
  - in each case we replace term \( [D] \) by a new term that is easier to handle in a dynamic programming algorithm

**Lemma (Second Recursive Formula)**

Given two trees \( T_1 \) and \( T_2 \), \( i \in N(T_1) \) and \( d_i \in \text{desc}(i) \), \( j \in N(T_2) \) and \( d_j \in \text{desc}(j) \), then:

1. \( \text{If } l(i) = l(d_i) \) and \( l(j) = l(d_j) \), i.e., both forests are trees:

   \[ \text{fdist}(T_1[1:(i)..<di], T_2[1:(j)..<dj]) = \min \begin{cases} \text{fdist}(T_1[1:(i)..<di-1], T_2[1:(j)..<dj]) + \omega_{\text{del}} \\ \text{fdist}(T_1[1:(i)..<di], T_2[1:(j)..<dj-1]) + \omega_{\text{ins}} \\ \text{fdist}(T_1[1:(i)..<di], T_2[1:(j)..<dj]) + \omega_{\text{ren}} \end{cases} \]

2. \( \text{If } l(i) \neq l(d_i) \) and/or \( l(j) \neq l(d_j) \), i.e., one of the forests is not a tree:

   \[ \text{fdist}(T_1[1:(i)..<di], T_2[1:(j)..<dj]) = \min \begin{cases} \text{fdist}(T_1[1:(i)..<di-1], T_2[1:(j)..<dj]) + \omega_{\text{del}} \\ \text{fdist}(T_1[1:(i)..<di], T_2[1:(j)..<dj-1]) + \omega_{\text{ins}} \\ \text{fdist}(T_1[1:(i)..<di], T_2[1:(j)..<dj]) + \omega_{\text{ren}} \end{cases} \]
Proof of the Second Recursive Formula

<table>
<thead>
<tr>
<th>Proof.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) follows from the previous recursive formula for ( l(i) = l(d_i) ) and ( l(j) = l(d_j) ) as the following holds:</td>
</tr>
</tbody>
</table>
| \[
fdist(T_1[l(i)..l(d_i) - 1], T_2[l(j)..l(d_j) - 1]) = fdist(∅, ∅) = 0.
\]
| (2) The following inequation holds: |
| \[
\begin{align*}
&[A] fdist(T_1[l(i)..l(d_i)], T_2[l(j)..l(d_j)]) \\
&\leq fdist(T_1[l(i)..l(d_i) - 1], T_2[l(j)..l(d_j) - 1]) [B] \\
&\quad + fdist(T_1[l(d_i)..d_i], T_2[l(d_j)..d_j]) [C] \\
&\leq fdist(T_1[l(i)..l(d_i) - 1], T_2[l(j)..l(d_j) - 1]) [B] \\
&\quad + fdist(T_1[l(d_i)..d_i], T_2[l(d_j)..d_j - 1]) [D] \\
&\quad + \omega_{ren}
\end{align*}
\]
| \( A \leq B + C \) as the left-hand side is the minimal cost mapping, while the right-hand side is a particular case with a possibly sub-optimal mapping. |
| \( C \leq D + \omega_{ren} \) holds for the same reason. |
| As we are looking for the minimum distance, we can substitute \( D + \omega_{ren} \) by \( C \). |

Illustration: Proof of the Second Recursive Formula (1/2)

- Case (1): \( l(i) = l(d_i) \) and \( l(j) = l(d_j) \):

Illustration: Proof of the Second Recursive Formula (2/2)

- Case (2): \( l(i) \neq l(d_i) \) and/or \( l(j) \neq l(d_j) \):

**Implications by the Second Recursive Formula**

- Note: \( fdist(T_1[l(d_i)..d_i], T_2[l(d_j)..d_j]) \) is the tree edit distance between the subtrees rooted in \( T[d_i] \) and \( T[d_j] \). We use the following notation:

\[
treedist(d_i, d_j) = fdist(T_1[l(d_i)..d_i], T_2[l(d_j)..d_j])
\]

- Dynamic Programming: As the same sub-problem must be solved many times, we use a dynamic programming approach.

- Bottom-Up: As for the computation of the tree distance \( treedist(i,j) \) we need almost all values \( treedist(d_i, d_j) \) \( d_j \in desc(T_1[i]) \), \( d_j \in desc(T_1[j]) \), we use a bottom-up approach.

- Key Roots: If

- \( d_i \) is on the path from \( l(i) \) to \( T_1[i] \) and
- \( d_j \) is on the path from \( l(j) \) to \( T_2[j] \),

then \( treedist(d_i, d_j) \) is computed as a byproduct of \( treedist(i,j) \). We call the nodes that are not computed as byproducts the key roots.
Key Roots

Definition (Key Root)
The set of key roots of a tree $T$ is defined as
$$kr(T) = \{ k \in N(T) \mid \exists k' \in N(T) : k' > k \text{ and } l(k) = l(k')\}$$

- Alternative definition: A key root is a node of $T$ that either has a left sibling or is the root of $T$.
- Example: $kr(T) = \{3, 5, 6\}$

Only subtrees rooted in a key root need a separate computation.
The number of key roots is equal to the number of leaves in the tree.

The Tree Edit Distance Algorithm

The Tree Edit Distance Algorithm

Outline

1. Tree Edit Distance
   - Preliminaries and Definition
   - Forests Distance and Recursive Formula
   - Second Recursive Formula
   - The Tree Edit Distance Algorithm
   - Example: Tree Edit Distance Computation
   - Complexity of the Tree Edit Distance Algorithm

The Edit Distance Algorithm I/II

td[1..|T1|, 1..|T2|] : empty array for tree distances;
l1 = lmld(root(T1)); kr1 = kr(l1, |leaves(T1)|);
l2 = lmld(root(T2)); kr2 = kr(l2, |leaves(T2)|);
for $x = 1$ to $|kr1|$ do
  for $y = 1$ to $|kr2|$ do
    forest-dist(kr1[x], kr2[y], l1, l2, td);

- $l1$ is an array of size $|T1|$, $l1[i]$ is the leftmost leaf descendant of node $i$; $l2$ is the analog for $T2$ (detailed algorithm for lmld(.) follows)
- $kr1$ is an array that contains all the key roots of $T1$ sorted in ascending order; $kr2$ is the analog for $T2$ (detailed algorithm kr(.) follows)
- Algorithm and lemmas by [ZS89] (see also [AG97])

The Edit Distance Algorithm II/II

forest-dist(i, j, l1, l2, td)

$fd[l1[i] - 1..i, l2[j] - 1..j]$ : empty array;
$fd[l1[i] - 1, l2[j] - 1] = 0$;
for $d1 = l1[i]$ to $i$ do $fd[d1, l2[j] - 1] = fd[d1 - 1, l2[j] - 1] + \omega_{ins}$;
for $d1 = l1[i]$ to $j$ do $fd[l1[i] - 1, d1] = fd[l1[i] - 1, d1 - 1] + \omega_{ins}$;
for $d1 = l1[i]$ to $i$ do
  for $d2 = l2[j]$ to $j$ do
    if $l1[d1] = l1[i]$ and $l2[d2] = l2[j]$ then
      $fd[d1, d2] = \min(fd[d1 - 1, d2] + \omega_{del},$
      $fd[d1, d2] + \omega_{ins},$
      $fd[d1 - 1, d2 - 1] + \omega_{rem});$
    $td[d1, d2] = f(d1, d2);$
  else $fd[d1, d2] = \min(fd[d1 - 1, d2] + \omega_{del},$
  $fd[d1, d2 - 1] + \omega_{ins},$
  $fd[l1[d1] - 1, l2[d2] - 1] + td[d1, d2]);$
The Tree Edit Distance Algorithm

The Tree Edit Distance Algorithm

The Tree Distance Matrix

td[i][j] stores the tree edit distance between
- the tree rooted in T[i] (i.e., T[i]|l(i)..l(i)|) and
- the tree rooted in T[j] (i.e., T[j]|l(j)..d(j)).

Computing Key Roots and Left-Most Leaf Descendants

The tree edit distance algorithm uses the following functions:
- lmld(i): computes an array with the left-most leaf descendants of all descendants of a node i
- kr(l, lc): given the array l = lmld(i) of left-most leaf descendants, and the number lc of leaf descendants of i, compute all key roots of the subtree rooted in i

Computing the Left-Most Leaf Descendants

Imld(v, l)

foreach child c of v (left to right) do l ← lmld(c, l);
if v is a leaf then
  l[id(v)] ← id(v)
else
  c1 ← first child of v;
  l[id(v)] ← l[id(c1)];
return l;

Input: root node v of a tree T, empty array l[1..|T|]
Output: array l, l[i] is the left-most leaf descendant of node T[i]
Imld(root(T)) (see tree-edit-dist(…)) is implemented as lmld(root(T), l) with an empty array l[1..|T|].
Computing the Key Roots

\[
kr(l, lc) = \begin{cases} 
kr[1..lc]: \text{empty array;} \\
visited[l]: \text{boolean array of size } |l|, \text{init with false;} \\
k \leftarrow |kr|; \\
i \leftarrow |l|; \\
\text{while } k \geq 1 \text{ do} \\
\quad \text{if not visited}[l[i]] \text{ then} \\
\quad \quad kr[k-1] \leftarrow i; \\
\quad \quad visited[l[i]] \leftarrow \text{true}; \\
\quad i \leftarrow i - 1; \\
\text{return } kr;
\end{cases}
\]

\begin{itemize}
\item Input: 
\begin{itemize}
\item \(l[1..|T|] : l[i] \text{ is the left-most leaf descendent of node } T[i]\)
\item \(lc = |\text{leaves}(T)| \text{ is the number of leaves in } T\)
\end{itemize}
\item Output: array \(kr[1..|\text{leaves}(T)|] \) with key roots sorted by node ID
\item Note: Loop condition is correct, as \(k \geq 1 \Rightarrow i \geq 1\)
\end{itemize}

Example: Tree Edit Distance Computation

Example Trees and Edit Costs

\begin{itemize}
\item Example: Edit distance between \(T_1\) and \(T_2\).
\begin{itemize}
\item \(\omega_{\text{ins}} = \omega_{\text{del}} = 1\)
\item \(\omega_{\text{ren}} = 0 \text{ for identical rename, otherwise } \omega_{\text{ren}} = 1\)
\end{itemize}
\item Each of the following slide is the result of a call of forest-dist().
\end{itemize}
Tree Edit Distance Example: Tree Edit Distance Computation
Executing the Algorithm (2/9)

\[ l_1 = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 2 & 1 & 5 & 6
\end{array} \]

\[ kr_1 = \begin{array}{ccc}
3 & 5 & 6
\end{array} \]

- \( i = kr_1[x] = 3 \Rightarrow l_1[i] = 2 \)
- \( j = kr_2[y] = 5 \Rightarrow l_2[j] = 5 \)

\[ b[i] = b[d[i]] \text{ and } b[j] = b[d[j]] \]

permanent array \( td \):

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
0 & 1 & 2 & 1 & 2 & 2
\end{array}
\]

Executing the Algorithm (3/9)

\[ l_1 = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 1 & 1 & 5 & 1
\end{array} \]

\[ kr_1 = \begin{array}{ccc}
3 & 5 & 6
\end{array} \]

- \( i = kr_1[x] = 3 \Rightarrow l_1[i] = 2 \)
- \( j = kr_2[y] = 6 \Rightarrow l_2[j] = 1 \)

\[ b[i] = b[d[i]] \text{ and } b[j] = b[d[j]] \]

permanent array \( td \):

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
0 & 1 & 1 & 1 & 2 & 2
\end{array}
\]

Executing the Algorithm (4/9)

\[ l_1 = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 2 & 1 & 5 & 6
\end{array} \]

\[ kr_1 = \begin{array}{ccc}
3 & 5 & 6
\end{array} \]

- \( i = kr_1[x] = 5 \Rightarrow l_1[i] = 5 \)
- \( j = kr_2[y] = 2 \Rightarrow l_2[j] = 2 \)

\[ b[i] = b[d[i]] \text{ and } b[j] = b[d[j]] \]

permanent array \( td \):

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
4 & 1 & 3 & 2 & 4
\end{array}
\]

Executing the Algorithm (5/9)

\[ l_1 = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 2 & 1 & 5 & 1
\end{array} \]

\[ kr_1 = \begin{array}{ccc}
3 & 5 & 6
\end{array} \]

- \( i = kr_1[x] = 5 \Rightarrow l_1[i] = 5 \)
- \( j = kr_2[y] = 5 \Rightarrow l_2[j] = 5 \)

\[ b[i] = b[d[i]] \text{ and } b[j] = b[d[j]] \]

permanent array \( td \):

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
4 & 1 & 5 & 1
\end{array}
\]
Executing the Algorithm (6/9)

\[ l_1 \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 2 & 1 & 5 & 1 \end{array} \quad \text{kr}_1 \begin{array}{ccc} 3 & 5 & 6 \end{array} \]

- \( i = \text{kr}_1[x] = 5 \Rightarrow l_1[i] = 5 \)
- \( j = \text{kr}_2[y] = 6 \Rightarrow l_2[j] = 1 \)

- temporary array \( fd \):
  \[
  \begin{array}{cccccc}
  d_i \rightarrow & 1 & 2 & 3 & 4 & 5 & 6 \\
  d_j \downarrow & 1 & 0 & 2 & 2 & 3 & 1 \\
  5 & 1 & 0 & 2 & 2 & 3 & 1 \\
  \end{array}
  \]

permanent array \( td \):

\[
\begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 2 & 3 & 1 & 5 \\
3 & 1 & 2 & 2 & 2 & 4 \\
4 & 3 & 1 & 2 & 2 & 4 \\
5 & 1 & 0 & 2 & 2 & 3 \\
6 & 5 & 0 & 2 & 2 & 3 \\
\end{array}
\]

\( l_1[i] = l_1[d_i] \) and \( l_2[j] = l_2[d_j] \)

Executing the Algorithm (7/9)

\[ l_1 \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 2 & 1 & 5 & 1 \end{array} \quad \text{kr}_1 \begin{array}{ccc} 3 & 5 & 6 \end{array} \]

- \( i = \text{kr}_1[x] = 6 \Rightarrow l_1[i] = 1 \)
- \( j = \text{kr}_2[y] = 2 \Rightarrow l_2[j] = 2 \)

- temporary array \( fd \):
  \[
  \begin{array}{cccccc}
  d_i \rightarrow & 2 & 1 & 0 & 1 \\
  d_j \downarrow & 1 & 0 & 2 & 3 & 1 & 5 \\
  3 & 1 & 0 & 2 & 2 & 7 & 4 \\
  4 & 3 & 1 & 2 & 2 & 4 & 4 \\
  5 & 1 & 1 & 3 & 4 & 0 & 5 \\
  6 & 5 & 5 & 3 & 3 & 3 & 2 \\
  \end{array}
  \]

permanent array \( td \):

\[
\begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 1 & 2 & 3 & 5 & 6 \\
3 & 1 & 0 & 2 & 3 & 4 & 5 \\
4 & 2 & 1 & 0 & 1 & 2 & 3 & 4 \\
5 & 3 & 2 & 1 & 2 & 3 & 4 & 5 \\
6 & 4 & 3 & 2 & 1 & 2 & 3 & 4 \\
\end{array}
\]

\( l_1[i] = l_1[d_i] \) and \( l_2[j] = l_2[d_j] \)

Executing the Algorithm (8/9)

\[ l_1 \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 1 & 5 & 1 & 1 \end{array} \quad \text{kr}_1 \begin{array}{ccc} 3 & 5 & 6 \end{array} \]

- \( i = \text{kr}_1[x] = 6 \Rightarrow l_1[i] = 1 \)
- \( j = \text{kr}_2[y] = 5 \Rightarrow l_2[j] = 5 \)

- temporary array \( fd \):
  \[
  \begin{array}{cccccc}
  d_i \rightarrow & 5 \\
  d_j \downarrow & 1 & 1 & 1 & 1 \\
  1 & 1 & 1 & 1 & 1 \\
  2 & 1 & 0 & 2 & 3 \\
  3 & 2 & 1 & 0 & 1 \\
  4 & 3 & 2 & 1 & 2 \\
  5 & 4 & 3 & 2 & 1 \\
  6 & 5 & 4 & 3 & 2 \\
  \end{array}
  \]

permanent array \( td \):

\[
\begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 2 & 3 & 1 & 5 \\
3 & 1 & 2 & 2 & 2 & 4 \\
4 & 3 & 1 & 2 & 2 & 4 \\
5 & 1 & 0 & 2 & 2 & 3 \\
6 & 5 & 0 & 2 & 2 & 3 \\
\end{array}
\]

\( l_1[i] = l_1[d_i] \) and \( l_2[j] = l_2[d_j] \)

Executing the Algorithm (9/9)

\[ l_1 \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 2 & 1 & 5 & 1 \end{array} \quad \text{kr}_1 \begin{array}{ccc} 3 & 5 & 6 \end{array} \]

- \( i = \text{kr}_1[x] = 6 \Rightarrow l_1[i] = 1 \)
- \( j = \text{kr}_2[y] = 1 \Rightarrow l_2[j] = 1 \)

- temporary array \( fd \):
  \[
  \begin{array}{cccccc}
  d_i \rightarrow & 1 & 2 & 3 & 4 & 5 & 6 \\
  d_j \downarrow & 0 & 1 & 2 & 3 & 4 & 5 \\
  1 & 0 & 1 & 2 & 3 & 4 & 5 \\
  2 & 1 & 0 & 1 & 2 & 3 & 4 \\
  3 & 2 & 1 & 0 & 1 & 2 & 3 \\
  4 & 3 & 2 & 1 & 0 & 1 & 2 \\
  5 & 4 & 3 & 2 & 1 & 0 & 1 \\
  6 & 5 & 4 & 3 & 2 & 1 & 0 \\
  \end{array}
  \]

permanent array \( td \):

\[
\begin{array}{cccccc}
1 & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 1 & 2 & 3 & 5 & 6 \\
3 & 1 & 0 & 1 & 2 & 3 & 4 \\
4 & 2 & 1 & 0 & 1 & 2 & 3 \\
5 & 3 & 2 & 1 & 0 & 1 & 2 \\
6 & 4 & 3 & 2 & 1 & 0 & 1 \\
\end{array}
\]

\( l_1[i] = l_1[d_i] \) and \( l_2[j] = l_2[d_j] \)
Complexity of the Tree Edit Distance Algorithm

Similarity Search

Unit 5 – March 29, 2012

Tree Edit Distance

Preliminaries and Definition

Forests Distance and Recursive Formula

Second Recursive Formula

The Tree Edit Distance Algorithm

Example: Tree Edit Distance Computation

Complexity of the Tree Edit Distance Algorithm

Outline

1. Tree Edit Distance
   - Preliminaries and Definition
   - Forests Distance and Recursive Formula
   - Second Recursive Formula
   - The Tree Edit Distance Algorithm
   - Example: Tree Edit Distance Computation
   - Complexity of the Tree Edit Distance Algorithm

Tree Edit Distance Complexity of the Tree Edit Distance Algorithm

forest-dist: Time Complexity

forest-dist(i, j, l1, l2, td)

- `fd[l1[i] − 1..i, l2[j] − 1..j]`: empty array;
- `fd[l1[i] − 1, l2[j] − 1] = 0;
- for `dj = l2[j]` to `j` do `fd[l1[i] − 1, dj] = fd[l1[i] − 1, dj − 1] + ωins;`
- for `di = l1[i]` to `i` do
  - for `dj = l2[j]` to `j` do
    - if `l1[di] = l1[i]` and `l2[dj] = l2[j]` then
      - `fd[di, dj] = min( . . . );`
    - `td[di, dj] = f[di, dj];`
    - else `fd[di, dj] = min( . . . );`

- Input nodes are `i` and `j`.
- They are root nodes of subtrees `t1(i)` and `t2(j).`
- The nested loop is executed `|t1(i)| × |t2(j)|` times.
- ⇒ Time complexity `O(|t1(i)| × |t2(j)|)`

Notation:

- `|T|` is the number of nodes in `T`
- `depth(v)` is the number of ancestors of `v` (including `v` itself)
- `depth(T)` is the maximum depth of a node in `T`
- `leaves(T)` is the set of leaves of `T`
- `t(i)` is the subtree rooted in node `i`

Tree Edit Distance

Tree Edit Distance

tree-edit-dist: Time Complexity

Tree Edit Distance Complexity of the Tree Edit Distance Algorithm

Notation:

- `|T|` is the number of nodes in `T`
- `depth(v)` is the number of ancestors of `v` (including `v` itself)
- `depth(T)` is the maximum depth of a node in `T`
- `leaves(T)` is the set of leaves of `T`
- `t(i)` is the subtree rooted in node `i`

Computing `l1/2` and `kr1/2` is linear, `O(|T1| + |T2|)`

Main loop executes `forest-dist() |kr1| × |kr2|` times.

Complexity:

\[
\sum_{i \in kr1} \sum_{j \in kr2} |t1(i)| \times |t2(j)| = |t1(i)| \times \sum_{j \in kr2} |t2(j)|
\]

The following lemmas help us to reformulate this expression.
**Collapsed Depth**

**Definition:** The **collapsed depth** of a node \( v \) in \( T \) is

\[
cdepth(v) = |\text{anc}(v) \cap \text{kr}(T)|,
\]
i.e., the number of ancestors of \( v \) (including \( v \) itself) that are key roots.

Now we can **rewrite** the complexity formula:

\[
\sum_{i \in \text{kr}_1} |t_1(i)| \times \sum_{j \in \text{kr}_2} |t_2(j)| = \sum_{i=1}^{\text{|T}_1|} cdepth(i) \times \sum_{j=1}^{\text{|T}_2|} cdepth(j)
\]

\( cdepth(T) \geq cdepth(i) \) for a node \( i \) of \( T \), thus

\[
\sum_{i=1}^{\text{|T}_1|} cdepth(i) \times \sum_{j=1}^{\text{|T}_2|} cdepth(j) \leq |\text{T}_1||\text{T}_2|cdepth(\text{T}_1)cdepth(\text{T}_2)
\]

Two obvious **upper bounds** for the collapsed depth:
- the tree depth: \( cdepth(T) \leq \text{depth}(T) \)
- the number of key roots: \( cdepth(T) \leq |\text{kr}(T)| \)

We show that the number of key roots matches the number of leaves.

**Lemma (Number of Key Roots)**

The number of key roots of a tree is equal to the number of leaves:

\[
|\text{kr}(T)| = |\text{leaves}(T)|
\]

**Proof.**

We show that \( l() \) is a bijection from the key roots \( \text{kr}(T) \) to the \( \text{leaves}(T) \):

(a) **Injection** – for any \( i,j \in \text{kr}(T) \), \( i \neq j \Rightarrow l(i) \neq l(j) \):
If \( i > j \) and \( l(i) = l(j) \), \( j \) can not be a key root by definition. Analogous rational hold for \( j > i \).

(b) **Surjection** – Each leaf \( x \) has a key root \( i \in \text{kr}(T) \) such that \( l(i) = x \):
If there is no node \( i > x \) with \( l(i) = l(x) \), then by definition \( x \) itself is a key root \( l(x) = x \) is always true. Otherwise \( i \) is the key root of \( x \).
Theorem (Complexity of the Tree Edit Distance Algorithm)

Let $D_1$ and $D_2$ denote the depth, $L_1$ and $L_2$ the number of leave nodes, and $N_1$ and $N_2$ the total number of nodes of two trees $T_1$ and $T_2$, respectively.

1. The runtime of the tree edit distance algorithm is $O(N_1N_2 \min(D_1,L_1) \min(D_2,L_2))$.

2. Let $N = \max(N_1,N_2)$. For full, balanced, binary trees the runtime is $O(N^2 \log^2 N)$.

3. In the worst case $\min(D,L) = O(N)$ and the runtime is $O(N^4)$.

4. The algorithm needs $O(N_1N_2)$ space.

Proof of the Complexity Theorem

Proof.

1. Runtime (general formula): We have shown before, that the complexity is $O(|T_1||T_2|cdepth(T_1) \text{ cdepth}(T_2))$. As $cdepth(T) \leq |\text{leaves}(T)|$ (see definition of $cdepth(T)$ and previous lemma) and $cdepth(T) \leq depth(T)$ (follows from the definition of $cdepth(T)$), if follows that $cdepth(T) \leq \min(depth(T), |\text{leaves}(T)|)$.

2. Full, balanced, binary trees: In this case $depth(T) = O(\log(|T|))$.

3. Worst case: A full binary tree (i.e., each node has zero or two children) where each non-leaf nodes has at least one leaf child: $\min(depth(T), |\text{leaves}(T)|) = O(|T|)$.

4. Space: The size of the tree distance matrix $td$ is $|T_1| \times |T_2|$. In each call of forest-dist() we need a matrix of size $O(|T_1| \times |T_2|)$, which is freed when we exit the subroutine.

Recent Improvements of the Complexity

- Klein [Kle98] improves the worst case for the runtime to $O(|T_1|^2|T_2|) \log(|T_2|)$, thus from $O(N^6)$ to $O(N^3 \log(N))$.
- Dulucq and Touzet [DT03] also give an $O(N^3 \log(N))$ algorithm.
- Demaine et al. [DMRW07] give an $O(N^3)$ algorithm. They show that the algorithm is worst case optimal among all decomposition algorithms (i.e., algorithms like [ZS89, Kle98, DT03]), but it is not robust, i.e., it runs into the worst case when it could do better.
- Pawlik and Augsten [PA11] introduce the Robust Tree Edit Distance (RTED) algorithm which has optimal $O(N^3)$ worst case complexity and is robust.

Further reading:

