Approximation: Theory and Algorithms
Edit Distance Complexity, Upper and Lower Bounds

Nikolaus Augsten

Free University of Bozen-Bolzano
Faculty of Computer Science
DIS

Unit 9 – Mai 8, 2009
1 Complexity of the Tree Edit Distance Algorithm
   - Basic Complexity Formula
   - COLLapsed Depth and Key Roots
   - Complexity Results

2 Search Space Reduction for the Tree Edit Distance
   - Approximate Join
   - Lower Bound: Tree Traversal
   - Upper Bound: Constrained Edit Distance

3 Conclusion
Notation:

- $|T|$ is the number of nodes in $T$
- $\text{depth}(v)$ is the number of ancestors of $v$ (including $v$ itself)
- $\text{depth}(T)$ is the maximum depth of a node in $T$
- $\text{leaves}(T)$ is the set of leaves of $T$
- $t(i)$ is the subtree rooted in node $i$
forest-dist: Time Complexity

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

\[
\begin{align*}
fd[l1[i] - 1..i, l2[j] - 1..j] & : \text{empty array;} \\
fd[l1[i] - 1, l2[j] - 1] & = 0; \\
\text{for } d_i = l1[i] \text{ to } i \text{ do } fd[d_i, l2[j] - 1] & = fd[d_i - 1, l2[j] - 1] + \omega_{del}; \\
\text{for } d_j = l2[j] \text{ to } j \text{ do } fd[l1[i] - 1, d_j] & = fd[l1[i] - 1, d_j - 1] + \omega_{ins}; \\
\text{for } d_i = l1[i] \text{ to } i \text{ do } \\
\quad \text{for } d_j = l2[j] \text{ to } j \text{ do } \\
\qquad \text{if } l[d_i] = l[i] \text{ and } l[d_j] = l[j] \text{ then } \\
\qquad \quad fd[d_i, d_j] & = \min(\ldots); \\
\qquad \quad td[d_i, d_j] & = f[d_i, d_j]; \\
\quad \text{else } fd[d_i, d_j] & = \min(\ldots); 
\end{align*}
\]

- Input nodes are i and j.
- They are root nodes of subtrees \( t_1(i) \) and \( t_2(j) \).
- The nested loop is executed \( |t_1(i)| \times |t_2(j)| \) times.
- \( \Rightarrow \) Time complexity \( O(|t_1(i)| \times |t_2(j)|) \)
Complexity of the Tree Edit Distance Algorithm

Basic Complexity Formula

tree-edit-dist: Time Complexity

tree-edit-dist(T_1, T_2)

\[ td[1..|T_1|, 1..|T_2|] : \text{empty array for tree distances;} \]
\[ l_1 = \text{lmld}(\text{root}(T_1)); \quad kr_1 = \text{kr}(l_1, |\text{leaves}(T_1)|); \]
\[ l_2 = \text{lmld}(\text{root}(T_2)); \quad kr_2 = \text{kr}(l_2, |\text{leaves}(T_2)|); \]

\[ \text{for } x = 1 \text{ to } |kr_1| \text{ do} \]
\[ \quad \text{for } y = 1 \text{ to } |kr_2| \text{ do} \]
\[ \quad \text{forest-dist}(kr_1[x], kr_2[y], l_1, l_2, td); \]

- Computing \( l_{1/2} \) and \( kr_{1/2} \) is linear, \( O(|T_1| + |T_2|) \)
- Main loop executes forest-dist() \( |kr_1| \times |kr_2| \) times.
- Complexity:

\[
\sum_{i \in kr_1} \sum_{j \in kr_2} |t_1(i)| \times |t_2(j)| = \sum_{i \in kr_1} |t_1(i)| \times \sum_{j \in kr_2} |t_2(j)|
\]

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

**Definition:** The *collapsed depth* of a node $v$ in $T$ is

$$cdepth(v) = |anc(v) \cap kr(T)|,$$

i.e., the number of ancestors of $v$ (including $v$ itself) that are key roots.
Lemma (Collapsed Depth)

For a tree $T$ with key roots $kr(T)$

$$\sum_{k \in kr(T)} |t(k)| = |T| \sum_{k=1}^{cdepth(k)}$$

Proof.

- Consider the left-hand formula:
  - A node $i$ of $T$ is counted whenever it appears in a subtree $t(k)$.
  - Node $i$ is in the subtree $t(k)$ iff $k$ is the ancestor of $i$.
  - Only the subtrees of key roots are considered.

- Thus a node $i$ is counted once for each ancestor key root.

- $cdepth(i)$ is the number of ancestor key roots of $i$ (definition of collapsed depth).
Collapsed Depth

Now we can rewrite the complexity formula:

\[ \sum_{i \in kr_1} |t_1(i)| \times \sum_{j \in kr_2} |t_2(j)| = \sum_{i=1}^{cdepth(T)} cdepth(i) \times \sum_{j=1}^{cdepth(T)} cdepth(j) \]

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

\[ \sum_{i=1}^{cdepth(T)} cdepth(i) \times \sum_{j=1}^{cdepth(T)} cdepth(j) \leq |T_1||T_2|cdepth(T_1)cdepth(T_2) \]

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

- 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:

\[ |kr(T)| = |leaves(T)| \]

Proof.

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

(a) **Injection** – for any \( i, j \in kr(T) \), \( i \neq j \Rightarrow l(i) \neq l(j) \):
    
    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 \( i \in kr(T) \) 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 \( x \).
Complexity of the Tree Edit Distance Algorithm

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

   \[ O(N_1 N_2 \min(D_1, L_1) \min(D_2, L_2)) \].

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 $O(N_1 N_2)$ space.
Proof of the Complexity Theorem

Proof.

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

(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: $\min(depth(T), |leaves(T)|) = O(|T|)$.

(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 $O(N^4)$ 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 optimal among *decomposition algorithms* (algorithms as in [ZS89, Kle98, DT03]), i.e., the lower bound is also $\Omega(N^3)$. 
Definition: Approximate Join

Definition (Approximate Join)

Given two sets of trees, $S_1$ and $S_2$, and a distance threshold $\tau$, let $\delta_t(T_i, T_j)$ be a function that assesses the edit distance between two trees $T_i \in S_1$ and $T_j \in S_2$. The approximate join operation between two sets of trees reports in the output all pairs of trees $(T_i, T_j) \in S_1 \times S_2$ such that $\delta_t(T_i, T_j) \leq \tau$. 
Approximate Join Algorithm

\[
\text{approxJoin}(S_1, S_2)
\]

\[
\begin{align*}
\text{for each } T_i \in S_1 & \text{ do} \\
& \quad \text{for each } T_j \in S_2 \text{ do} \\
& \quad \quad \text{if } \text{upperBound}(T_i, T_j) \leq \tau \text{ then} \\
& \quad \quad \quad \text{output}(T_i, T_j) \\
& \quad \quad \text{if } \text{lowerBound}(T_i, T_j) \leq \tau \text{ then} \\
& \quad \quad \quad \text{if } \delta_t(T_i, T_j) \leq \tau \text{ then} \\
& \quad \quad \quad \quad \text{output}(T_i, T_j)
\end{align*}
\]
Preorder and Postorder Traversal Strings

- Each node label is a single character of an alphabet $\Sigma$.
- Traversal Strings:
  - $pre(T)$ is the string of $T$’s node labels in preorder
  - $post(T)$ is the string of $T$’s node labels in postorder

Lemma (Tree Inequality)

Let $pre(T_1)$ and $pre(T_2)$ be the preorder strings, and $post(T_1)$ and $post(T_2)$ be the postorder strings of two trees $T_1$ and $T_2$, respectively. Then

$$pre(T_1) \neq pre(T_2) \text{ or } post(T_1) \neq post(T_2) \Rightarrow T_1 \neq T_2$$

Proof.

The inversion of the argument is obviously true:

$$T_1 = T_2 \Rightarrow pre(T_1) = pre(T_2) \text{ and } post(T_1) = post(T_2)$$
If the traversal strings of two trees are equal, the trees can still be different:

$T_1$

```
/ \  
 a  a
 b  
```

$T_2$

```
/   
 a   
|   |
 b   
 a  
```

$pre(T_1) = aba = pre(T_2) = aba$
Lower Bound

Theorem (Lower Bound)

If the trees are at tree edit distance $k$, then the string edit distance between their preorder or postorder traversals is at most $k$.

Proof.

Tree operations map to string operations (illustration on next slide):

- **Insertion** ($\text{ins}(v, p, k, m)$): Let $t_1 \ldots t_f$ be the subtrees rooted in the children of $p$. Then the preorder traversal of the subtree rooted in $p$ is

  $$p \text{pre}(t_1) \ldots \text{pre}(t_{k-1}) \text{pre}(t_k) \ldots \text{pre}(t_m) \text{pre}(t_{m+1}) \ldots \text{pre}(t_f).$$

  Inserting $v$ moves the subtrees $k$ to $m$:

  $$p \text{pre}(t_1) \ldots \text{pre}(t_{k-1})v \text{pre}(t_k) \ldots \text{pre}(t_m) \text{pre}(t_{m+1}) \ldots \text{pre}(t_f).$$

  The string distance is 1. Analog rationale for postorder.

- **Deletion**: Inverse of insertion.

- **Rename**: With node rename a single string character is renamed.
Illustration for the Lower Bound Proof (Preorder)

\[ \text{ins}(v, p, k, m) \quad \overset{\kappa}{\rightarrow} \quad \text{del}(v) \]

\[ p \; \text{pre}(t_1) \ldots \text{pre}(t_{k-1}) \]
\[ \text{pre}(t_k) \ldots \text{pre}(t_m) \]
\[ \text{pre}(t_{m+1}) \ldots \text{pre}(t_f) \]
From the lower bound theorem it follows that

$$\max(\delta_s(pre(T_1), pre(T_2)), \delta_s(post(T_1), post(T_2))) \leq \delta_t(T_1, T_2)$$

where $\delta_s$ and $\delta_t$ are the string and the tree edit distance, respectively.

The string edit distance can be computed faster:
- string edit distance runtime: $O(n^2)$
- tree edit distance runtime: $O(n^3)$

Approximate join: match all trees with $\delta_t(T_1, T_2) \leq \tau$
- if $\max(\delta_s(pre(T_1), pre(T_2)), \delta_s(post(T_1), post(T_2))) > \tau$
  then $\delta_t(T_1, T_2) > \tau$
- thus we do not have to compute the expensive tree edit distance
Example: Traversal String Lower Bound

$\delta_s(\text{pre}(T_1), \text{pre}(T_2)) = 2$

$\delta_s(\text{post}(T_1), \text{post}(T_2)) = 2$

$\delta_t(T_1, T_2) = 2$
The string distances of preorder and postorder may be different.
The string distances and the tree distance may be different.

$$T_1$$
```
    a
   / 
  b   a
   
  c
```

$$T_2$$
```
    a
   / 
  b   
   
  c
```

pre($T_1$) = abac  pre($T_2$) = abac
post($T_1$) = bcaa  post($T_2$) = acba

$$\delta_s(\text{pre}(T_1), \text{pre}(T_2)) = 0$$
$$\delta_s(\text{post}(T_1), \text{post}(T_2)) = 2$$
$$\delta_t(T_1, T_2) = 3$$
Edit Mapping

- Recall the definition of the 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) $a = x \iff b = y$ (one-to-one condition)
   - (ii) $a$ is to the left of $x \iff b$ is to the left of $y$ (order condition)
   - (iii) $a$ is an ancestor of $x \iff b$ 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)

---

\(^1i.e., \ a \text{ precedes } x \text{ in both preorder and postorder}\)
Constrained Edit Distance

- We compute a special case of the edit distance to get a faster algorithm.
- \( lca(a, b) \) is the lowest common ancestor of \( a \) and \( b \).
- **Additional requirement** on the mapping \( M \):
  \[ (4) \text{ for any pairs } (a_1, b_1), (a_2, b_2), (x, y) \text{ of } M: \]
  \[
  lca(a_1, a_2) \text{ is a proper ancestor of } x
  \iff
  lca(b_1, b_2) \text{ is a proper ancestor of } y.
  \]
- **Intuition**: Two distinct subtrees of \( T_1 \) are mapped to two distinct subtrees of \( T_2 \).
Example: Constrained Edit Distance

- **Constrained** edit distance (dashed lines): $\delta_c(T_1, T_2) = 5$
  - constrained mapping $M_c = \{(a, a), (d, d), (c, i), (f, f)(g, g)\}$
  - edit sequence: $\text{ren}(c, i), \text{del}(b), \text{del}(e), \text{ins}(h), \text{ins}(e)$

- **Unconstrained** edit distance (dotted lines): $\delta_t(T_1, T_2) = 3$
  - mapping $M_t = \{(a, a), (d, d), (e, e), (c, i), (f, f)(g, g)\}$
  - edit sequence: $\text{ren}(c, i), \text{del}(b), \text{ins}(h)$
Example: Constrained Edit Distance

• \((e, e)\) violates the 4th condition of the constrained mapping:
  • \(\text{lca}(e, f)\) in \(T_1\) is \(a\)
  • \(a\) is a proper ancestor of \(d\) in \(T_1\)
  • assume \((e, e), (f, f), (d, d) \in M_c\)
  • \(\text{lca}(e, f)\) in \(T_2\) is \(h\)
  • \(h\) is not a proper ancestor of \(d\) in \(T_2\)
**Theorem (Complexity of the Constrained Edit Distance)**

Let $T_1$ and $T_2$ be two trees with $|T_1|$ and $|T_2|$ nodes, respectively. There is an algorithm that computes the constrained edit distance between $T_1$ and $T_2$ with runtime

$$O(|T_1||T_2|).$$

**Proof.**

See [Zha95, GJK+02].
Constrained Edit Distance: Upper Bound

Theorem (Upper Bound)

Let $T_1$ and $T_2$ be two trees, let $\delta_t(T_1, T_2)$ be the unconstrained and $\delta_c(T_1, T_2)$ be the constrained tree edit distance, respectively. Then

$$\delta_t(T_1, T_2) \leq \delta_c(T_1, T_2)$$

Proof.

See [GJK+02].
Use of the Upper Bound

- The constrained edit distance can be computed faster:
  - constrained edit distance runtime: $O(n^2)$
  - unconstrained edit distance runtime: $O(n^3)$

- Approximate join: match all trees with $\delta_t(T_1, T_2) \leq \tau$
  - if $\delta_c(T_1, T_2) \leq \tau$ then also $\delta_t(T_1, T_2) \leq \tau$.
  - thus we do not have to compute the expensive tree edit distance
Summary

- Tree Edit Distance Complexity
- Search Space Reduction
  - Lower Bound: Traversal Strings
  - Upper Bound: Constrained Edit Distance
What’s Next?

- Reference Sets (Upper and Lower Bound)
- Binary Branch Distance (Lower Bound)
Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann.
An optimal decomposition algorithm for tree edit distance.

Serge Dulucq and Hélène Touzet.
Analysis of tree edit distance algorithms.

Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu.
Approximate XML joins.
Philip N. Klein.
Computing the edit-distance between unrooted ordered trees.
In *Proceedings of the 6th European Symposium on Algorithms*,
volume 1461 of *Lecture Notes in Computer Science*, pages 91–102,

Kaizhong Zhang.
Algorithms for the constrained editing distance between ordered
labeled trees and related problems.

Kaizhong Zhang and Dennis Shasha.
Simple fast algorithms for the editing distance between trees and
related problems.