

# Similarity Search

## Traversal Strings and Constrained Edit Distance

Nikolaus Augsten

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



WS 2021/22

Version October 26, 2021

## Outline

- 1 Search Space Reduction for the Tree Edit Distance
  - Similarity Join and Search Space Reduction
  - Lower Bound: Traversal Strings
  - Upper Bound: Constrained Edit Distance
- 2 Conclusion

## Outline

- 1 Search Space Reduction for the Tree Edit Distance
  - Similarity Join and Search Space Reduction
    - Lower Bound: Traversal Strings
    - Upper Bound: Constrained Edit Distance
- 2 Conclusion

## Definition: Similarity Join

### Definition (Similarity 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 similarity 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  $\delta_t(T_i, T_j) \leq \tau$ .

## Similarity Join Algorithm with Upper and Lower Bounds

simJoin( $S_1, S_2$ )

```

for each  $T_i \in S_1$  do
  for each  $T_j \in S_2$  do
    if upperBound( $T_i, T_j$ )  $\leq \tau$  then
      output( $T_i, T_j$ )
    else if lowerBound( $T_i, T_j$ )  $> \tau$  then
      /* do nothing */
    else if  $\delta_t(T_i, T_j) \leq \tau$  then
      output( $T_i, T_j$ )
    
```

## Outline

- 1 Search Space Reduction for the Tree Edit Distance
  - Similarity Join and Search Space Reduction
  - Lower Bound: Traversal Strings
  - Upper Bound: Constrained Edit Distance
  
- 2 Conclusion

## 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

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

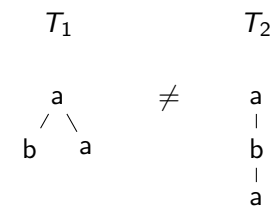
### Proof.

The inversion of the argument is obviously true:

$$T_1 = T_2 \Rightarrow pre(T_1) = pre(T_2) \text{ and } post(T_1) = post(T_2) \quad \square$$

## Notes: Traversal Strings and Tree Inequality

- If the traversal strings of two trees are equal, the trees can still be different:



$$pre(T_1) = \mathbf{aba} \quad pre(T_2) = \mathbf{aba}$$

## Lower Bound

### Theorem (Lower Bound)

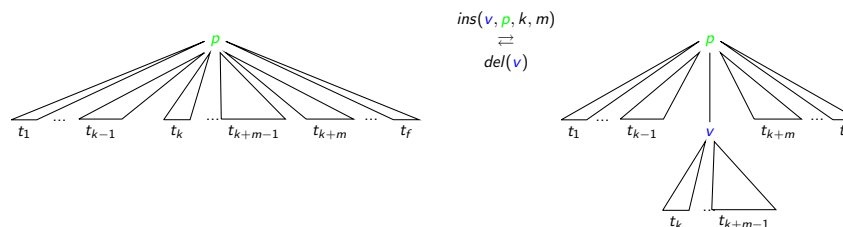
If the trees are at tree edit distance  $k$ , then the string edit distance between their preorder and postorder traversals is at most  $k$ .

### Proof.

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

- **Insertion** ( $ins(v, p, k, m)$ ): Let  $t_1 \dots t_f$  be the subtrees rooted in the children of  $p$ . Then the preorder traversal of the subtree rooted in  $p$  is  $p \text{ pre}(t_1) \dots \text{pre}(t_{k-1}) \text{pre}(t_k) \dots \text{pre}(t_{k+m-1}) \text{pre}(t_{k+m}) \dots \text{pre}(t_f)$ . Inserting  $v$  moves the subtrees  $k$  to  $m$ :  
 $p \text{ pre}(t_1) \dots \text{pre}(t_{k-1}) v \text{ pre}(t_k) \dots \text{pre}(t_{k+m-1}) \text{pre}(t_{k+m}) \dots \text{pre}(t_f)$ .  
 The string distance is 1. Analog rationale for postorder.
- **Deletion**: Inverse of insertion.
- **Rename**: With node rename a single string character is renamed.  $\square$

## Illustration for the Lower Bound Proof (Preorder)



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

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

## Lower Bound

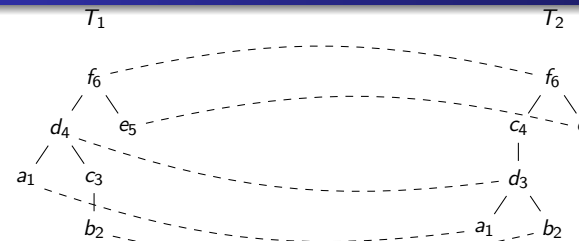
- From the lower bound theorem it follows that

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

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:  $O(n^3)$
- **Similarity join**: match all trees with  $\delta_t(T_1, T_2) \leq \tau$ 
  - if  $\max(\delta_s(\text{pre}(T_1), \text{pre}(T_2)), \delta_s(\text{post}(T_1), \text{post}(T_2))) > \tau$  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



$\text{pre}(T_1) = \text{fdacbe}$

$\text{pre}(T_2) = \text{fcdabe}$

$\text{post}(T_1) = \text{abcdef}$

$\text{post}(T_2) = \text{abdcef}$

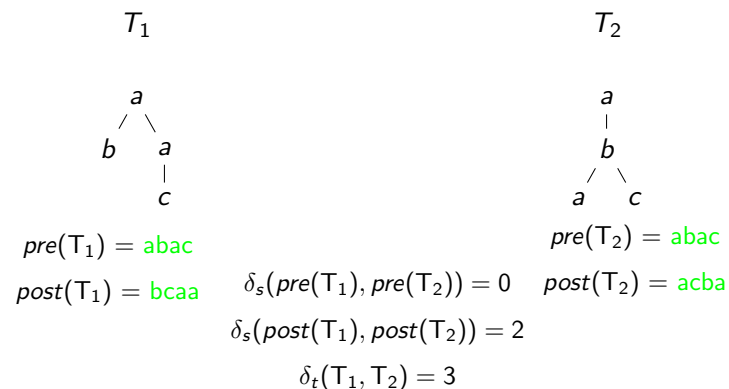
$$\delta_s(\text{pre}(T_1), \text{pre}(T_2)) = 2$$

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

$$\delta_t(T_1, T_2) = 2$$

## Example: Traversal String Lower Bound

- The string distances of preorder and postorder may be different.
- The string distances and the tree distance may be different.



## Outline

- 1 Search Space Reduction for the Tree Edit Distance
  - Similarity Join and Search Space Reduction
  - Lower Bound: Traversal Strings
  - Upper Bound: Constrained Edit Distance
- 2 Conclusion

## 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)  $a = x \Leftrightarrow b = y$  (one-to-one condition)
  - (ii)  $a$  is to the left of  $x^1 \Leftrightarrow b$  is to the left of  $y$  (order condition)
  - (iii)  $a$  is an ancestor of  $x \Leftrightarrow b$  is an ancestor of  $y$  (ancestor condition)

<sup>1</sup>i.e.,  $a$  precedes  $x$  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) for any pairs  $(a_1, b_1), (a_2, b_2), (x, y)$  of  $M$ :

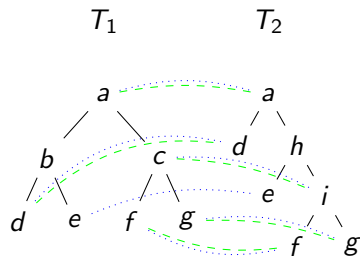
$$lca(a_1, a_2) \text{ is a proper ancestor of } x$$

$$\Leftrightarrow$$

$$lca(b_1, b_2) \text{ is a proper ancestor of } y.$$

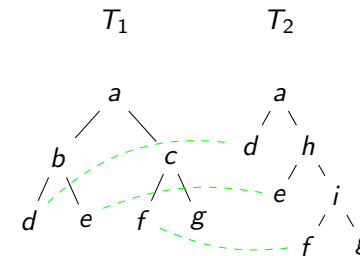
- **Intuition:** Distinct subtrees of  $T_1$  are mapped to distinct subtrees of  $T_2$ .

## 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:  $ren(c, i), del(b), del(e), ins(h), ins(e)$
- **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:  $ren(c, i), del(b), ins(h)$

## Example: Constrained Edit Distance



- $(e, e)$  violates the 4th condition of the constrained mapping:
  - $lca(e, f)$  in  $T_1$  is  $a$
  - $a$  is a proper ancestor of  $d$  in  $T_1$
  - assume  $(e, e), (f, f), (d, d) \in M_c$
  - $lca(e, f)$  in  $T_2$  is  $h$
  - $h$  is **not a proper ancestor** of  $d$  in  $T_2$

## Complexity of the Constrained Edit Distance

### 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<sup>+</sup>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<sup>+</sup>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:  $O(n^3)$
- **Similarity 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

 Sudepto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu.

Approximate XML joins.

In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 287–298, Madison, Wisconsin, 2002. ACM Press.

 Kaizhong Zhang.

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

*Pattern Recognition*, 28(3):463–474, 1995.