

# Approximation: Theory and Algorithms

## The Tree Edit Distance (II)

Nikolaus Augsten

Free University of Bozen-Bolzano  
Faculty of Computer Science  
DIS

Unit 7 – April 17, 2009

# Outline

- 1 Tree Edit Distance (II)
  - Recursive Formula for Non-Empty Forests
  - The Tree Edit Distance Algorithm
- 2 Conclusion

# 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:

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

## Proof

## Proof.

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

- (1)  $T_1[d_i]$  is not touched by a line in  $M$ :  $T_1[d_i]$  is **deleted** and  

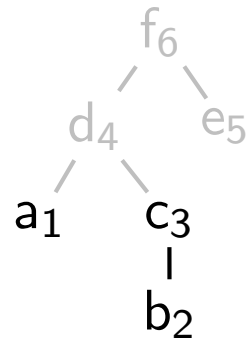
$$fdist(T_1[l(i)..d_i], T_2[l(j)..d_j]) = fdist(T_1[l(i)..d_i - 1], T_2[l(j)..d_j]) + \omega_{del}$$
- (2)  $T_2[d_j]$  is not touched by a line in  $M$ :  $T_2[d_j]$  is **inserted** and  

$$fdist(T_1[l(i)..d_i], T_2[l(j)..d_j]) = fdist(T_1[l(i)..d_i], T_2[l(j)..d_j - 1]) + \omega_{ins}$$
- (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  $(T_1[d_i], T_2[x]) \in M$  and  $(T_1[y], T_2[d_j]) \in M$ .
  - Case  $T_1[d_i]$  is to the right of  $T_1[y]$ : By sibling condition on  $M$  also  $T_2[d_j]$  must be to the right of  $T_2[x]$ . Impossible in  $T_2[l(j)..d_j]$ .
  - Case  $T_1[d_i]$  is proper ancestor of  $T_1[y]$ : By ancestor condition on  $M$  also  $T_2[d_j]$  must be ancestor of  $T_2[x]$ . Impossible in  $T_2[l(j)..d_j]$ .

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

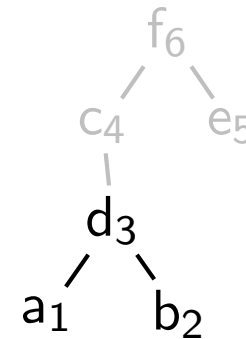
# Example: First Recursive Formula (1/3)

$T_1$



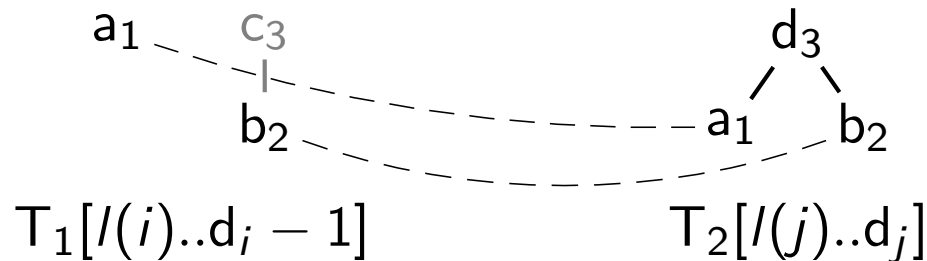
$T_1[l(i)..d_i]$   
 $(i = 6, d_i = 3)$

$T_2$



$T_2[l(j)..d_j]$   
 $(j = 6, d_j = 3)$

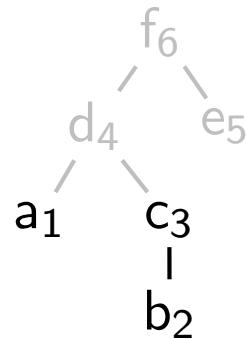
(1)  $fdist(T_1[l(i)..d_i - 1], T_2[l(j)..d_j]) + \omega_{del}$



- edit script:  $ins(d_3), del(c_3)$
- cost:  $1 + 1 = 2$

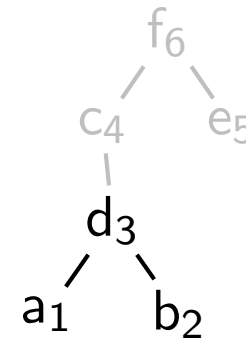
# Example: First Recursive Formula (2/3)

$T_1$



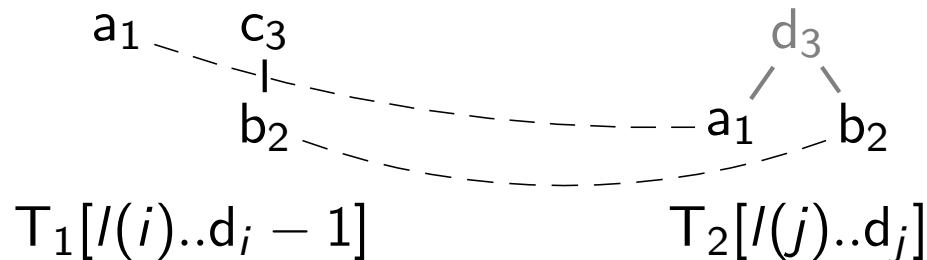
$T_1[l(i)..d_i]$   
 $(i = 6, d_i = 3)$

$T_2$



$T_2[l(j)..d_j]$   
 $(j = 6, d_j = 3)$

(2)  $fdist(T_1[l(i)..d_i], T_2[l(j)..d_j - 1]) + \omega_{ins}$

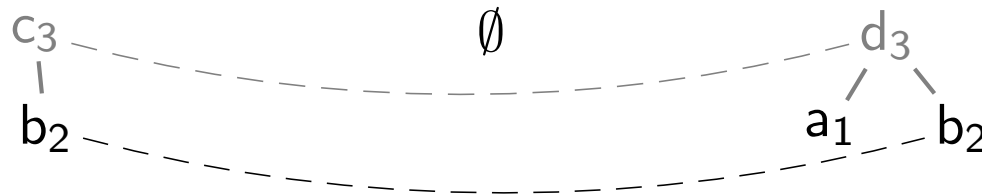


- edit script:  $del(c_3), ins(d_3)$
- cost:  $1 + 1 = 2$

# Example: First Recursive Formula (3/3)

$$\begin{aligned}
 (3) \quad & fdist(T_1[l(i)..l(d_i) - 1], T_2[l(j)..l(d_j) - 1]) \\
 & + fdist(T_1[l(d_i)..d_i - 1], T_2[l(d_j)..d_j - 1]) \\
 & + \omega_{ren}
 \end{aligned}$$

$a_1$



$$T_1[l(i)..l(d_i) - 1] \quad T_1[l(d_i)..d_i - 1] \quad T_2[l(j)..l(d_j) - 1] \quad T_2[l(d_j)..d_j - 1]$$

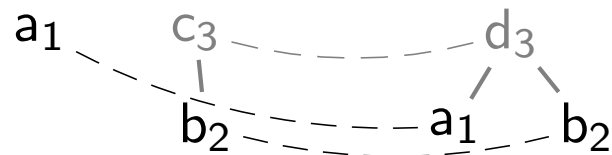
- $T_1[l(i)..d_i - 1] \rightarrow T_2[l(j)..d_j - 1]$ :  $del(a_1)$
- $T_1[l(d_i)..d_i - 1] \rightarrow T_2[l(d_j)..d_j - 1]$ :  $ins(a_1)$
- $c_3 \rightarrow d_3$ :  $ren(c_3, d_3)$
- cost:  $1 + 1 + 1 = 3$

# Analogy to the String Case

- Why is the third formula not (in analogy to the string case):

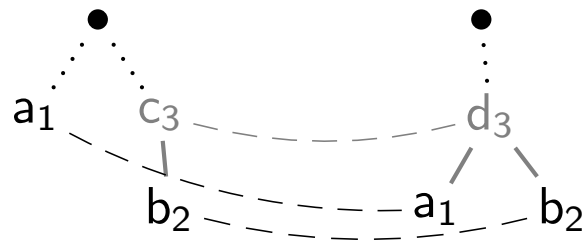
$$fdist(T_1[l(i)..d_i - 1], T_2[l(j)..d_j - 1]) + \omega_{ren}$$

- Consider the previous example:



$$T_1[l(i)..d_i - 1] \quad T_2[l(j)..d_j - 1]$$

- $ren(c_3, d_3)$  does **not** transform  $T_1[l(i)..d_i]$  to  $T_2[l(j)..d_j]$
- 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

- Observation about the First Recursive Formula:
  - suggests bottom-up dynamic programming algorithm
  - $fdist(T_1[l(d_i)..d_i-1], T_2[l(d_j)..d_j-1])$   $[D]$  does not fit this scheme
- 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 bottom up algorithm

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

# Second Recursive Formula: Forest Distance

## 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) If  $l(i) = l(d_i)$  and  $l(j) = l(d_j)$ , i.e., both forests are trees:

$$f\text{dist}(T_1[l(i)..d_i], T_2[l(j)..d_j]) = \min \begin{cases} f\text{dist}(T_1[l(i)..d_i - \mathbf{1}], T_2[l(j)..d_j]) + \omega_{del} \\ f\text{dist}(T_1[l(i)..d_i], T_2[l(j)..d_j - \mathbf{1}]) + \omega_{ins} \\ f\text{dist}(T_1[l(i)..d_i - \mathbf{1}], T_2[l(j)..d_j - \mathbf{1}]) + \omega_{ren} \end{cases}$$

(2) 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:

$$f\text{dist}(T_1[l(i)..d_i], T_2[l(j)..d_j]) = \min \begin{cases} f\text{dist}(T_1[l(i)..d_i - \mathbf{1}], T_2[l(j)..d_j]) + \omega_{del} \\ f\text{dist}(T_1[l(i)..d_i], T_2[l(j)..d_j - \mathbf{1}]) + \omega_{ins} \\ f\text{dist}(T_1[l(i)..l(d_i) - \mathbf{1}], T_2[l(j)..l(d_j) - \mathbf{1}]) \\ \quad + f\text{dist}(T_1[l(d_i)..d_i], T_2[l(d_j)..d_j]) \end{cases}$$

# Proof of the Second Recursive Formula

Proof.

- (1) follows from the previous recursive formula for  $l(i) = l(d_i)$  and  $l(j) = l(d_j)$  as the following holds:

$$fdist(T_1[l(i)..l(d_i) - 1], T_2[l(j)..l(d_j) - 1]) = fdist(\emptyset, \emptyset) = 0.$$

- (2) The following inequation holds:

$$\begin{aligned}
 [A] \quad fdist(T_1[l(i)..d_i], T_2[l(j)..d_j]) &\leq fdist(T_1[l(i)..l(d_i) - 1], T_2[l(j)..l(d_j) - 1]) && [B] \\
 &\quad + fdist(T_1[l(d_i)..d_i], T_2[l(d_j)..d_j]) && [C] \\
 &\leq fdist(T_1[l(i)..l(d_i) - 1], T_2[l(j)..l(d_j) - 1]) && [B] \\
 &\quad + fdist(T_1[l(d_i)..d_i - 1], T_2[l(d_j)..d_j - 1]) && [D] \\
 &\quad + \omega_{ren}
 \end{aligned}$$

$A \leq B + C$  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.

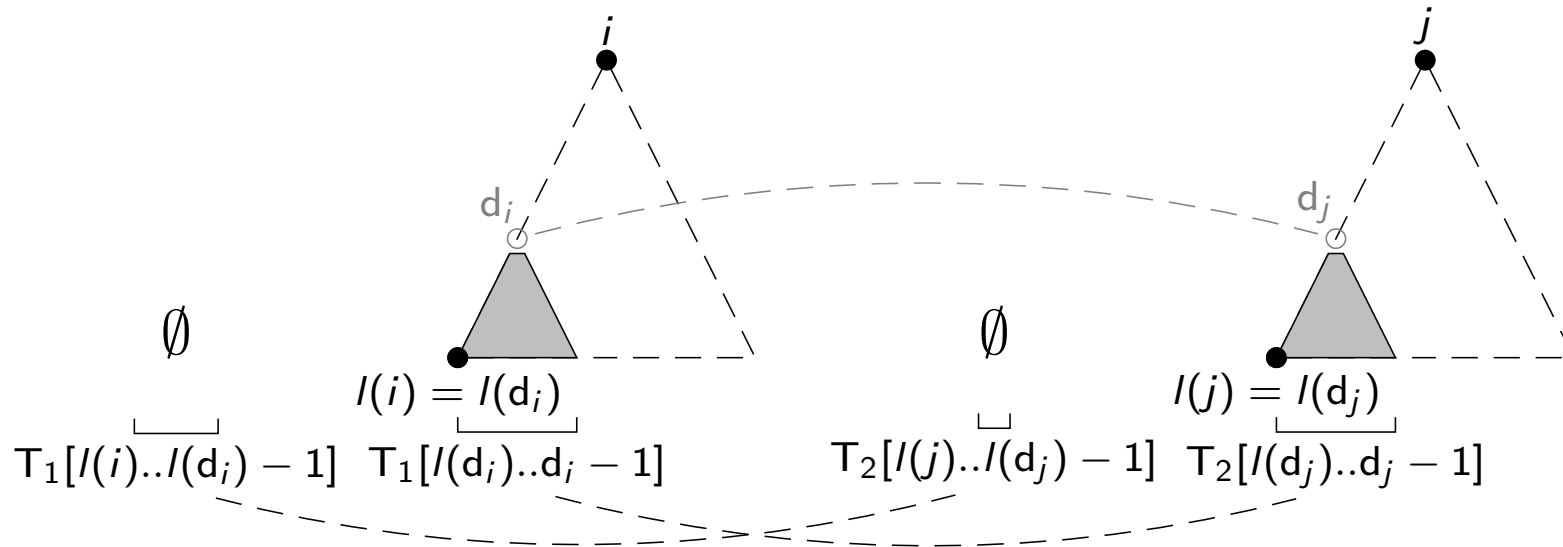
$C \leq D + \omega_{ren}$  holds for the same reason.

As we are looking for the *minimum* distance, we can substitute  $D + \omega_{ren}$  by  $C$ .

□

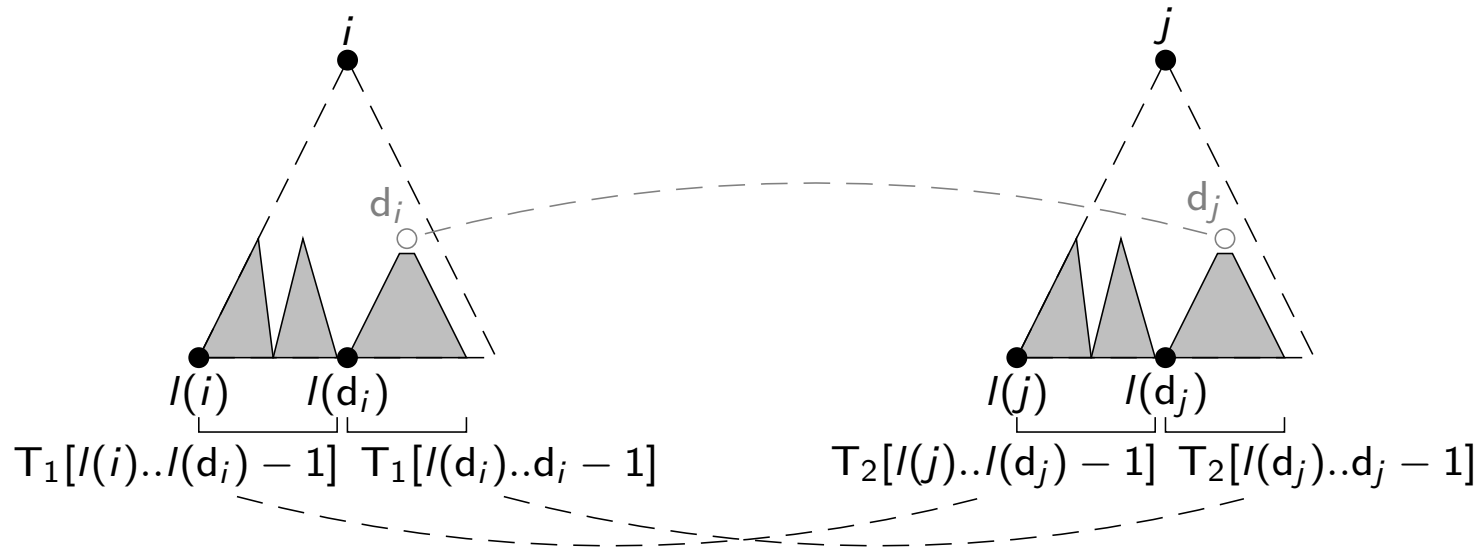
# Illustration: Proof of the Second Recursive Formula (1/2)

- 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  $treedist(d_i, d_j)$  ( $d_i \in desc(T_1[i]), d_j \in desc(T_1[j])$ ), 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 a byproducts the key roots.

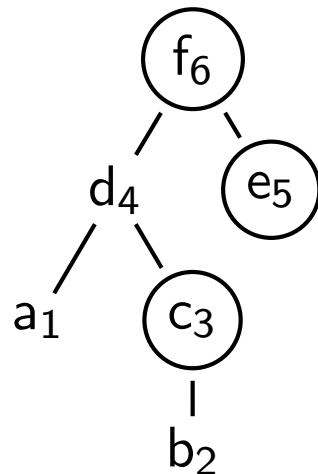
# Key Roots

## Definition (Key Root)

The set of *key roots* of a tree  $T$  is defined as

$$kr(T) = \{k \in N(T) \mid \nexists k' \in N(T) : k' > k \text{ and } l(k) = l(k')\}$$

- **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 Edit Distance Algorithm I/II

**tree-edit-dist( $T_1, T_2$ )**

```

td[1.. $|T_1|$ , 1.. $|T_2|$ ] : empty array for tree distances;
l1 = lml(root( $T_1$ )); kr1 = kr(l1,  $|leaves(T_1)|$ );
l2 = lml(root( $T_2$ )); kr2 = kr(l2,  $|leaves(T_2)|$ );
for  $x = 1$  to  $|kr_1|$  do
    for  $y = 1$  to  $|kr_2|$  do
        forest-dist(kr1[ $x$ ], kr2[ $y$ ], l1, l2, td);
  
```

- $l_1$  is an array of size  $|T_1|$ ,  $l_1[i]$  is the leftmost leaf descendant of node  $i$ ;  $l_2$  is the analog for  $T_2$  (detailed algorithm *l*ml(.) follows)
- $kr_1$  is an array that contains all the key roots of  $T_1$  sorted in ascending order;  $kr_2$  is the analog for  $T_2$  (detailed algorithm *kr*(.,.) follows)
- Algorithm and lemmas by [ZS89] (see also [AG97])

# The Edit Distance Algorithm II/II

**forest-dist**( $i, j, l_1, l_2, td$ )

$fd[l_1[i] - 1..i, l_2[j] - 1..j]$  : empty array;

$fd[l_1[i] - 1, l_2[j] - 1] = 0$ ;

**for**  $d_i = l_1[i]$  **to**  $i$  **do**  $fd[d_i, l_2[j] - 1] = fd[d_i - 1, l_2[j] - 1] + \omega_{del}$ ;

**for**  $d_j = l_2[j]$  **to**  $j$  **do**  $fd[l_1[i] - 1, d_j] = fd[l_1[i] - 1, d_j - 1] + \omega_{ins}$ ;

**for**  $d_i = l_1[i]$  **to**  $i$  **do**

**for**  $d_j = l_2[j]$  **to**  $j$  **do**

**if**  $l[d_i] = l[i]$  **and**  $l[d_j] = l[j]$  **then**

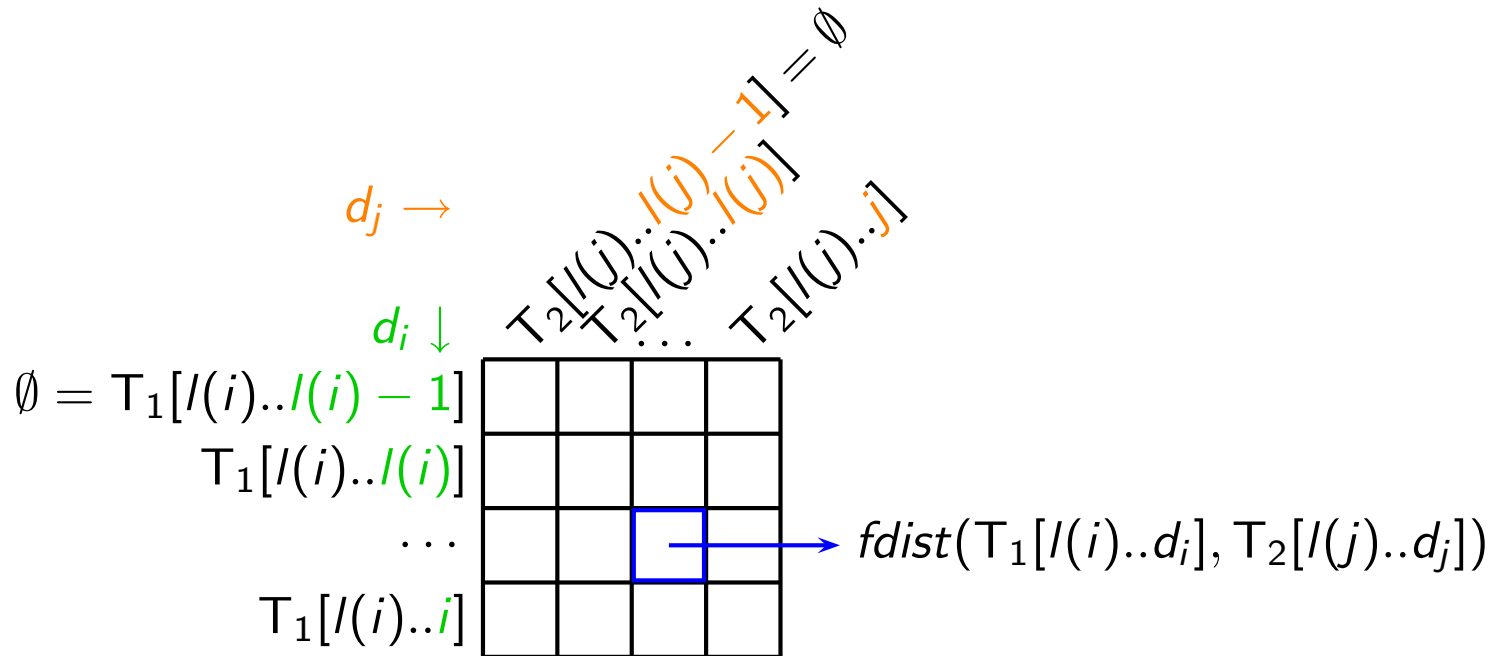
$fd[d_i, d_j] = \min(fd[d_i - 1, d_j] + \omega_{del},$   
                                      $fd[d_i, d_j - 1] + \omega_{ins},$   
                                      $fd[d_i - 1, d_j - 1] + \omega_{ren});$

$td[d_i, d_j] = f[d_i, d_j];$

**else**  $fd[d_i, d_j] = \min(fd[d_i - 1, d_j] + \omega_{del},$   
                                      $fd[d_i, d_j - 1] + \omega_{ins},$   
                                      $fd[l[d_i] - 1, l[d_j] - 1] + td[d_i, d_j]);$

# The Temporary Forest Distance Matrix

- $fd[d_i, d_j]$  contains the forest distance between
  - $T_1[l(i)..d_i]$ , where  $d_i \in desc(T_1[i])$  and
  - $T_2[l(j)..d_j]$ , where  $d_j \in desc(T_2[j])$ .



- $fd$  is temporary and exists only in `forest-dist()`

# The Tree Distance Matrix

- $td[i][j]$  stores the tree edit distance between
  - the tree rooted in  $T_1[i]$  (i.e.,  $T_1[l(i)..i]$ ) and
  - the tree rooted in  $T_2[j]$  (i.e.,  $T_2[l(j)..j]$ ).
- each call of `forest-dist()` fills new values into  $td$
- $td[|T_1|, |T_2|]$  stores the tree edit distance between  $T_1$  and  $T_2$

# Summary

- Tree Edit Distance
  - recursive formula for the edit distance
  - dynamic programming algorithm

# What's Next?

- Tree Edit Distance (III):
  - correctness and complexity of the algorithm
  - edit distance variants
- Lower Bounds for the Edit Distance



Alberto Apostolico and Zvi Galil, editors.

*Pattern Matching Algorithms*, chapter Tree Pattern Matching, pages 341–371.

Oxford University Press, 1997.



Kaizhong Zhang and Dennis Shasha.

Simple fast algorithms for the editing distance between trees and related problems.

*SIAM Journal on Computing*, 18(6):1245–1262, 1989.