The Editing Distance Between Trees

Ferienakademie Sarntal 1999

Carsten Isert

Institut für Informatik, Technische Universität München

e-mail: isert@in.tum.de

Overview

- Introduction
- Operations: Modify, Insert, Delete
- Dynamic Programming and Outline of Algorithm
- Definitions and Introduction to Mappings
- Simple Dynamic Programming Algorithm
- Complexity
- Extensions
- Tree Alignment
- Summary

Introduction

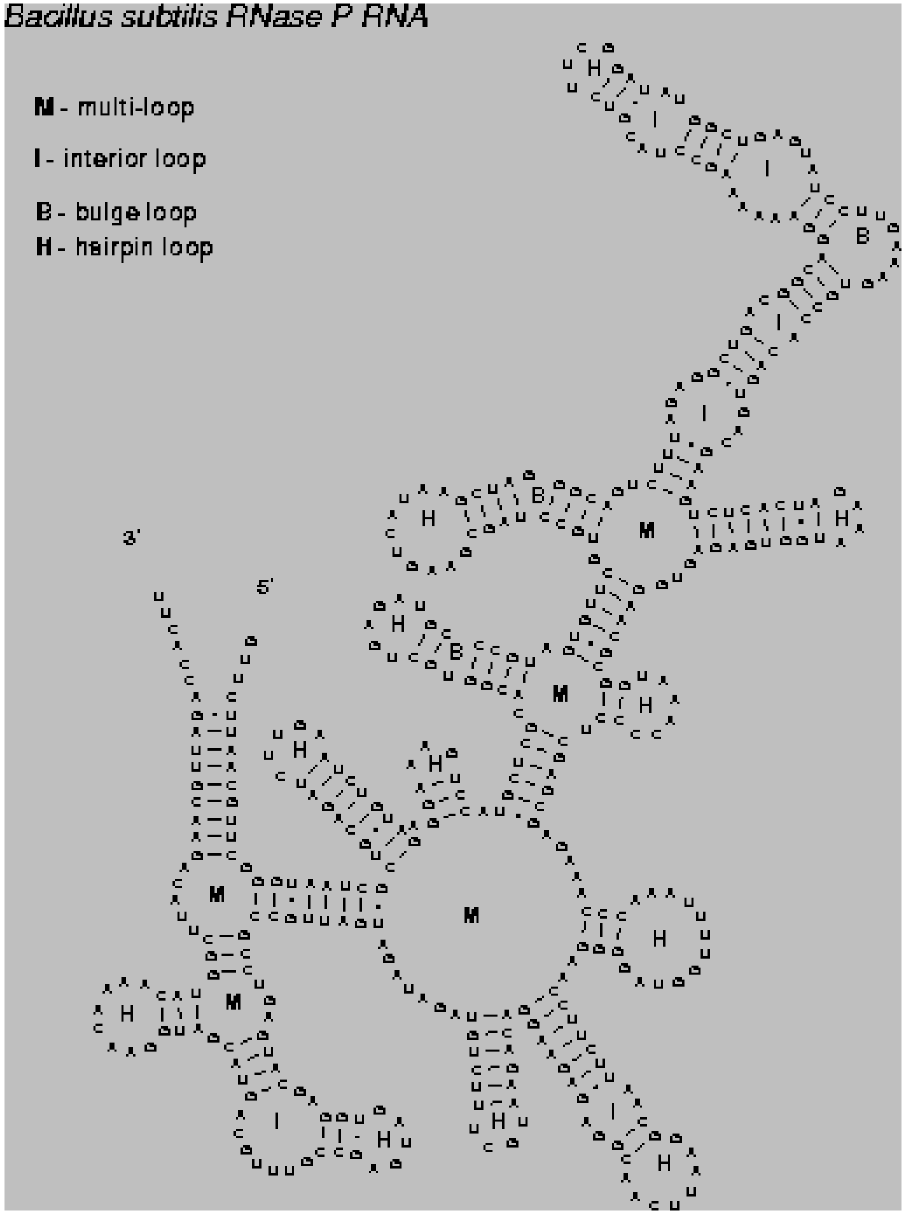
Motivation

- Representations of grammar parses (error recovery), RNA secondary structures, image clusters, chemical structures
- How different/similar are two trees?
- Similarity to string problem
- Metrics: largest common subtree, smallest common supertree, tree edit distance, alignment

Properties

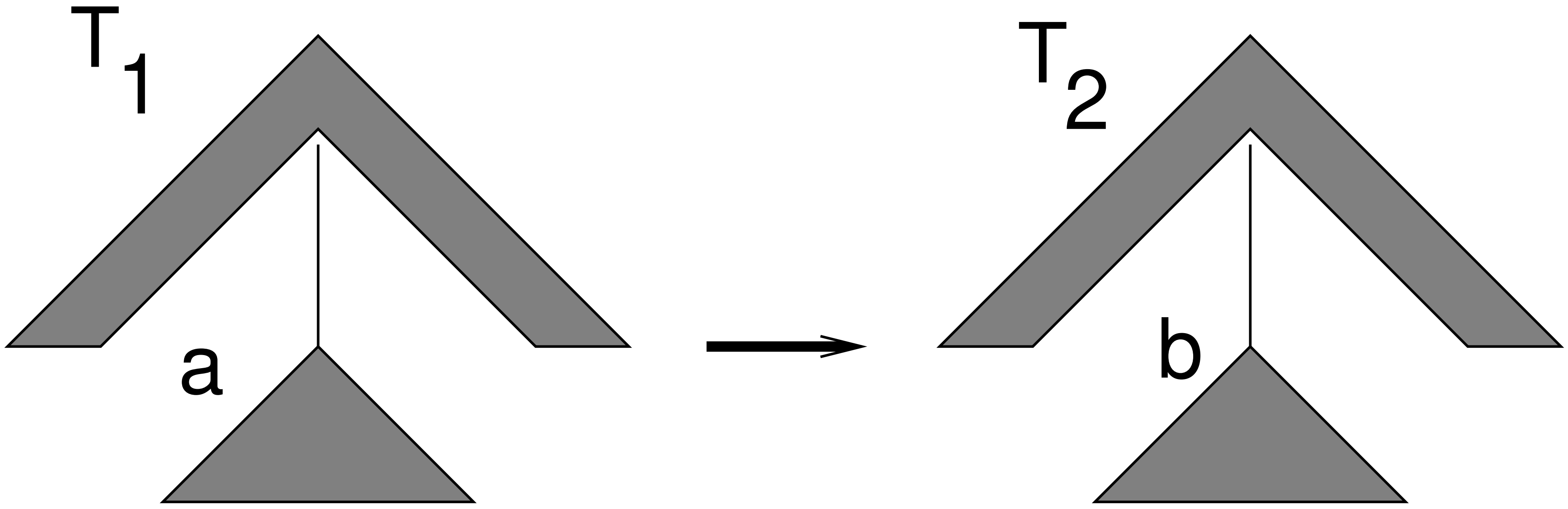
- Ordered labeled trees
- Left-to-right order among siblings is significant

Example RNA



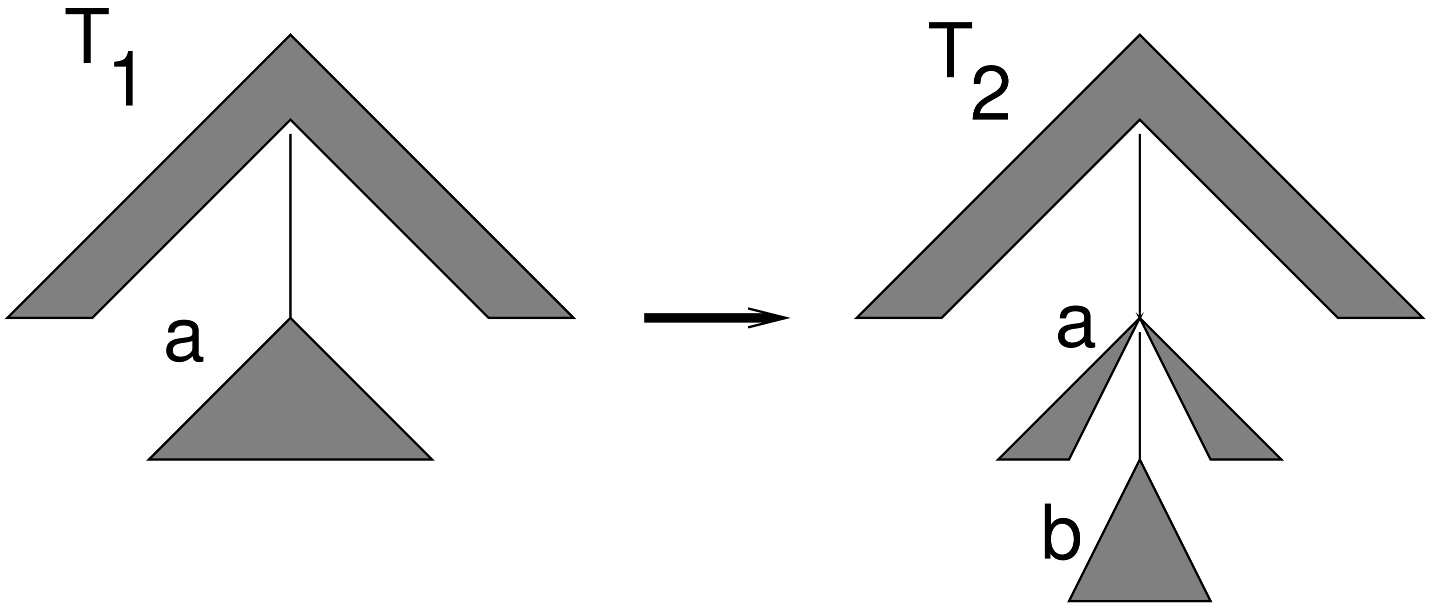
Bacillus subtilis RNase P RNA
M - multi-loop
I - interior loop
B - bulge loop
H - hairpin loop
3'
5'

Operations: Modify



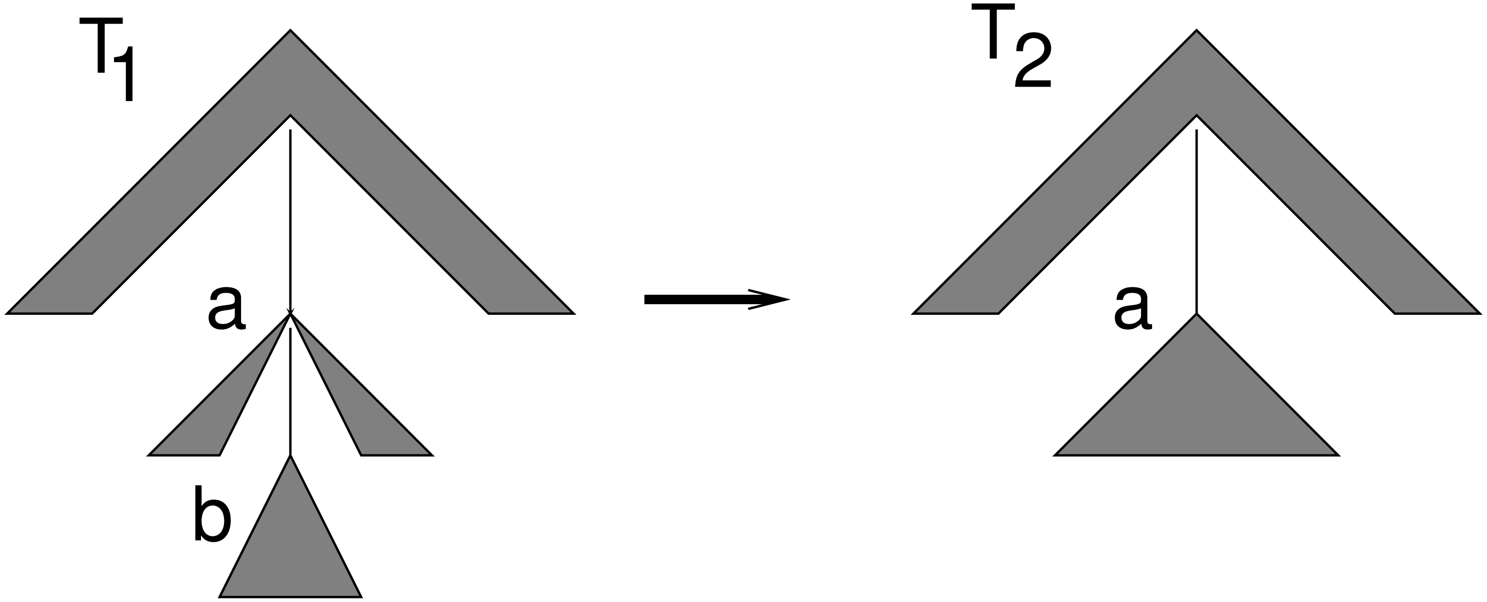
- Change the label from a to b
- Will be represented by the operation $(a \mapsto b)$ or (a, b)
- Ambiguity of node labels (resolved by context)

Operations: Insert



- Insert node b as an additional son of a and make a consecutive number of children of a become children of b
- Will be represented by the operation $(\Lambda \mapsto b)$ or (Λ, b)
- Nodes can be inserted above the root

Operations: Delete



- Delete son b from node a and make the children of b become the children of a
- Will be represented by the operation $(b \mapsto \Lambda)$ or (b, Λ)
- The root can be deleted if it has only one child

Dynamic Programming

Strategy for Dynamic Programming

1. Characterize structure of optimal solution
2. Recursively define value of optimal solution
3. Compute value of optimal solution bottom-up
4. Construct optimal solution

Elements of Dynamic Programming

- Optimal substructure
- Overlapping subproblems

Outline of Algorithm

Main Loop

- Select tree nodes that need separate computation
- Compute distance between subtrees rooted at these nodes
- Dynamic programming approach reuses previously computed results (bottom-up)
- Final result is computed at the two roots

Subtree Computations

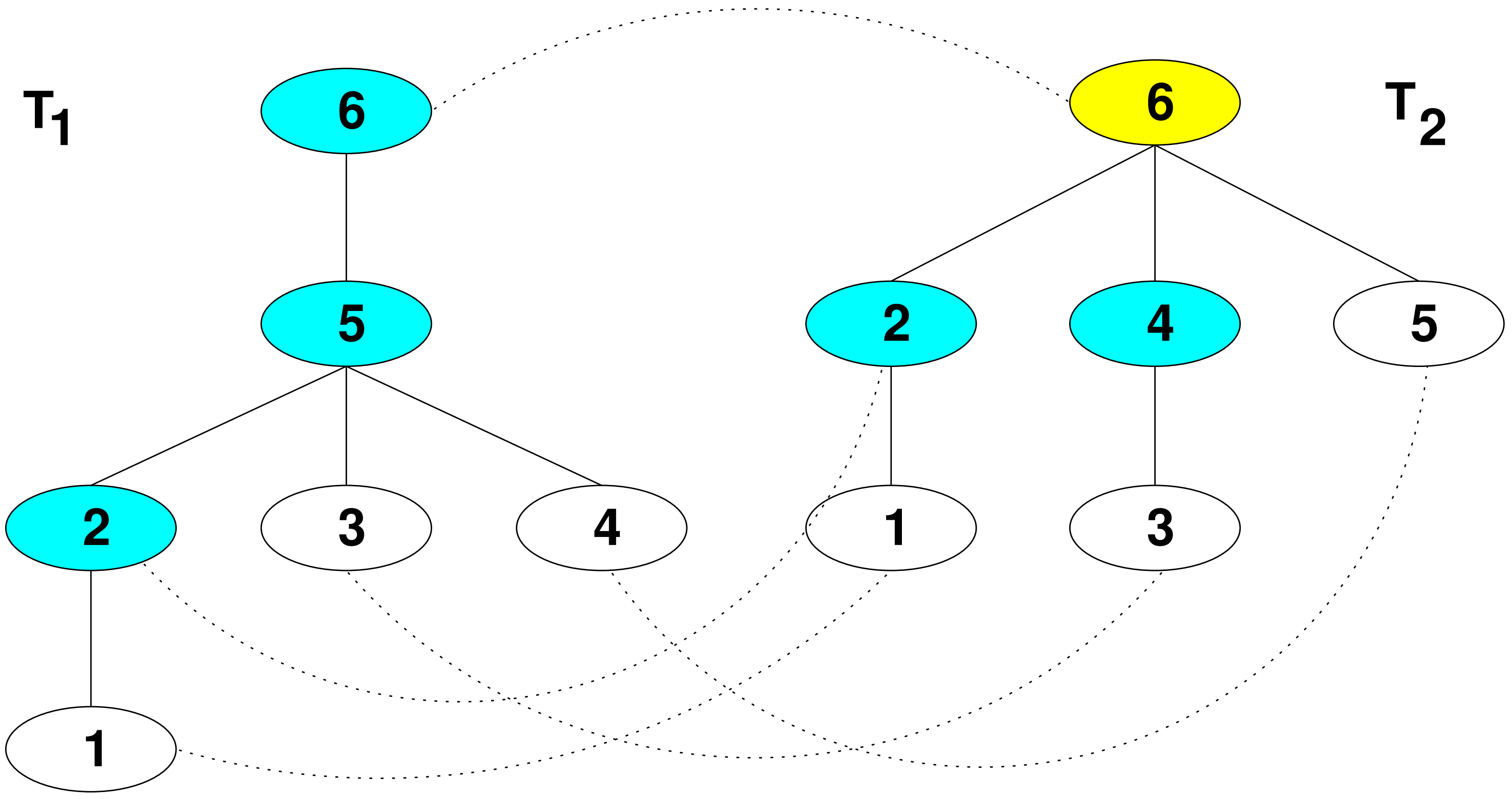
- Split tree into (reasonable) forests
- Compute the distance between these forests
- Look up old tree and forest values
- Fill in new tree values (for forests that are really trees)

Definitions

- Associated cost $\gamma(a \mapsto b)$ is distance metric:
  - $\gamma(a \mapsto b) \geq 0; \quad \gamma(a \mapsto a) = 0$
  - $\gamma(a \mapsto b) = \gamma(b \mapsto a)$
  - $\gamma(a \mapsto c) \leq \gamma(a \mapsto b) + \gamma(b \mapsto c)$
- Let $S = s_1, \ldots, s_k$ be a sequence of operations
- The S-derivation from tree T_1 to tree T_2 be is given by $U_0 = T_1$, $U_k = T_2$, and $U_{i-1} \mapsto U_i$ via s_i for $1 \leq i \leq k$
- Cost of S is $\gamma(S) = \sum_{i=1}^{|S|} \gamma(s_i)$
- Distance between T_1 and T_2 is

$$\delta(T_1, T_2) = \min\{\gamma(S) | \text{S is a sequence taking } T_1 \text{ to } T_2\}$$

Mappings



Postorder numbering of nodes in trees

Intuitive Meaning of Mappings

- Graphical specification of edit operations
- $T[i]$ denotes ith node in the postorder numbering
- A dotted line indicates that the modify operation must be applied if labels of $T_1[i]$ and $T_2[j]$ mismatch
- Untouched nodes of T_1 have to be deleted
- Untouched nodes of T_2 have to be inserted
- Example: delete node 5, insert node 4, relabel 6

Mappings (3)

Definition: M is a mapping if the following conditions hold for all $(i_1, j_1), (i_2, j_2) \in M$, $(1 \leq i_k \leq |T_1|, 1 \leq j_k \leq |T_2|)$:

- One-to-one: $i_1 = i_2$ iff $j_1 = j_2$
- Sibling order preserved: $T_1[i_1]$ is to the left of $T_1[i_2]$ iff $T_2[j_1]$ is to the left of $T_2[j_2]$
- Ancestor order preserved: $T_1[i_1]$ is an ancestor of $T_1[i_2]$ iff $T_2[j_1]$ is an ancestor of $T_2[j_2]$

Cost: Let I, J be the sets of nodes not touched by mapping M

$$\gamma(M) = \sum_{(i,j)\in M} \gamma(T_1[i] \mapsto T_2[j]) + \sum_{i \in I} \gamma(T_1[i] \mapsto \Lambda) + \sum_{j \in J} \gamma(\Lambda \mapsto T_2[j])$$

Mappings (4)

Properties

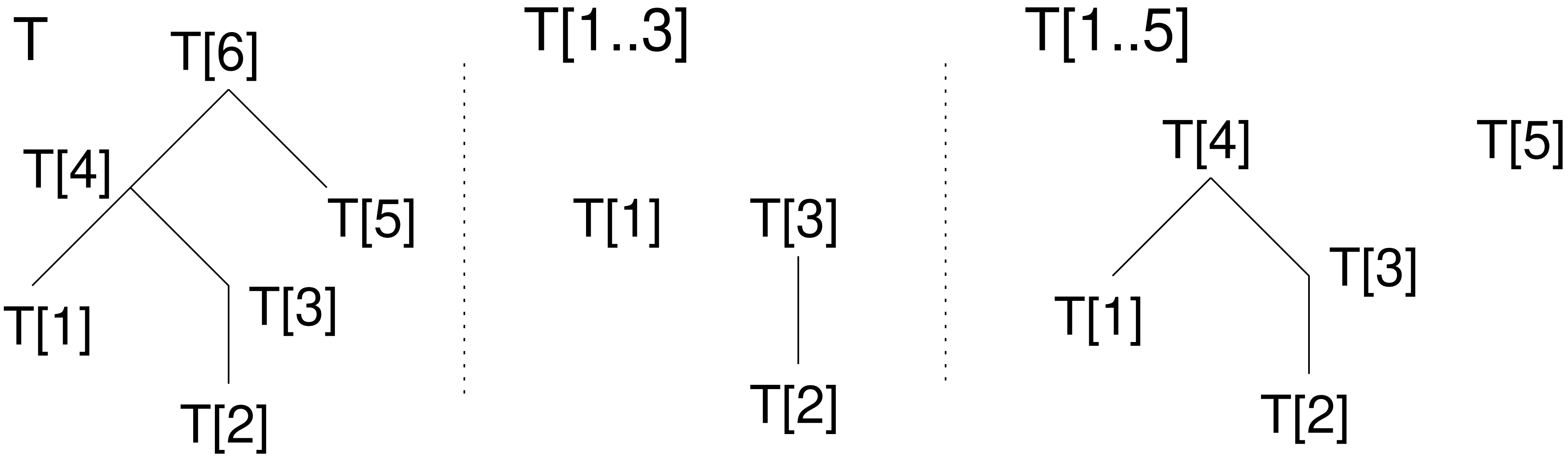
- $M_1 \circ M_2$ is a mapping
- $\gamma(M_1 \circ M_2) \leq \gamma(M_1) + \gamma(M_2)$
- For a sequence $S = s_1, \ldots, s_k$ from T_1 to T_2, there exists a mapping M such that $\gamma(M) \leq \gamma(S)$
- For any mapping M there exists a sequence S such that $\gamma(M) = \gamma(S)$
- This equivalence makes definition of distances constructive
- $\delta(T_1, T_2) = \min\{\gamma(M) | M$ is a mapping from T_1 to $T_2\}$

Simple Dynamic Programming Algorithm

Definitions

- $T[i..j]$ is the forest of nodes from i to j
- If $i > j$ then $T[i..j] = \emptyset$
- $T[1..i]$ is also called $forest(i)$
- $l(i)$ is leftmost leaf descendant of the subtree rooted at $T[i]$
- The parent of $T[i]$ is denoted $p(i)$
- $p^0(i) = i, p^k(i) = p(p^{k-1}(i))$
- $anc(i) = \{p^k(i) | 0 \leq k \leq depth(i)\}$

Algorithm (2)



Examples of Definitions

Algorithm (3)

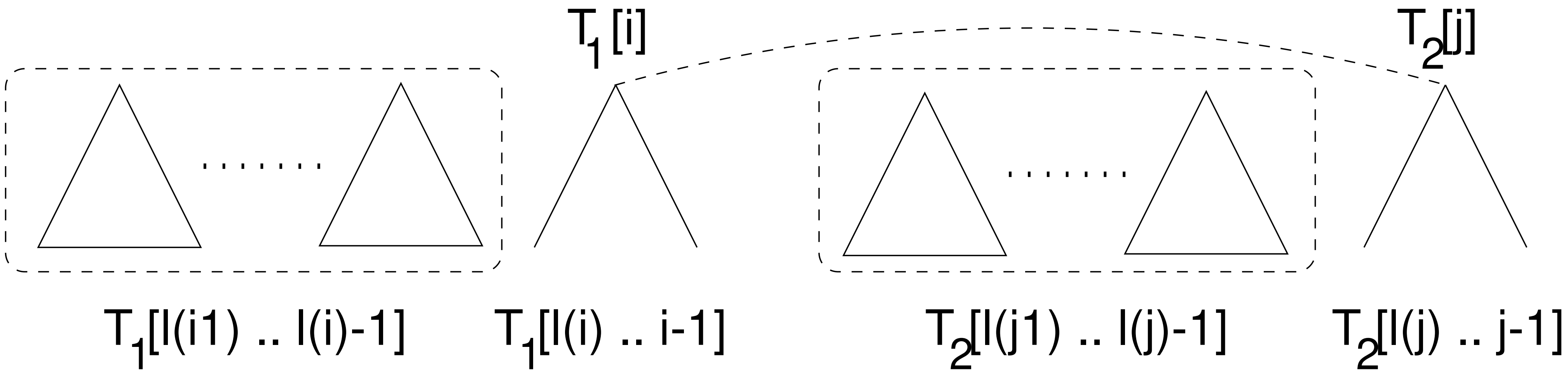
Further Definitions

- $fdist(i'..i, j'..j)$ is distance between forests $T_1[i'..i]$ and $T_2[j'..j]$
- $tdist(i, j)$ is distance between subtrees rooted at i and j

Lemma 1: For $i_1 \in anc(i)$ and $j_1 \in anc(j)$ the distance $fdist$ between two forests $T_1[l(i_1)..i]$ and $T_2[l(j_1)..j]$ satisfies:

- $fdist(\emptyset, \emptyset) = 0$
- $fdist(T_1[l(i_1)..i], \emptyset) = fdist(T_1[l(i_1)..i-1, \emptyset]) + \gamma(T_1[i], \Lambda)$
- $fdist(\emptyset, T_2[l(j_1)..j]) = fdist(\emptyset, T_2[l(j_1)..j-1]) + \gamma(\Lambda, T_2[j])$

Algorithm (4)



Visualization of $fdist(l(i_1)..i, l(j_1)..j)$

Lemma 2:

$$fdist(l(i_1)..i, l(j_1)..j) = \min \begin{cases} fdist(l(i_1)..i-1, l(j_1)..j)+ \\ \qquad \gamma(T_1[i] \mapsto \Lambda) \\ fdist(l(i_1)..i, l(j_1)..j-1)+ \\ \qquad \gamma(\Lambda \mapsto T_2[j]) \\ fdist(l(i_1)..l(i)-1, l(j_1)..l(j)-1)+ \\ \qquad fdist(l(i)..i-1, l(j)..j-1)+ \\ \qquad \gamma(T_1[i] \mapsto T_2[j]) \end{cases}$$

Algorithm (6)

Lemma 3:

- If $l(i) = l(i_1)$ and $l(j) = l(j_1)$

$$fdist(l(i_1)..i, l(j_1)..j) = \min \begin{cases} fdist(l(i_1)..i-1, l(j_1)..j) + \gamma(T_1[i], \Lambda) \\ fdist(l(i_1)..i, l(j_1)..j-1) + \gamma(\Lambda, T_2[j]) \\ fdist(l(i_1)..i-1, l(j_1)..j-1) + \\ \quad \gamma(T_1[i], T_2[j]) \end{cases}$$

- If $l(i) \neq l(i_1)$ or $l(j) \neq l(j_1)$

$$fdist(l(i_1)..i, l(j_1)..j) = \min \begin{cases} fdist(l(i_1)..i-1, l(j_1)..j) + \gamma(T_1[i], \Lambda) \\ fdist(l(i_1)..i, l(j_1)..j-1) + \gamma(\Lambda, T_2[j]) \\ fdist(l(i_1)..l(i)-1, l(j_1)..l(j)-1) + \\ \quad tdist(i, j) \end{cases}$$

Algorithm (7)

- $fdist(l(i_1)..i, l(j_1)..j) \leq$
  $fdist(l(i_1)..l(i) - 1, l(j_1)..l(j) - 1) + tdist(i, j)$
- $tdist(i, j) \leq fdist(l(i)..i - 1, l(j)..j - 1) + \gamma(T_1[i], T_2[j])$

Further Definitions

- $LR\_keyroots(T) = \{k|$ there exists no $k' > k$ such that $l(k) = l(k')\}$
- Set contains root and all nodes with left siblings
- Each element needs separate computation

Algorithm (8)

Algorithm Pseudo C Code:

```
preprocessing();    // left1/2, LR_keyroots1/2
                    // init tdist array, costs

for (i=1 ; i <= |T1| ; i++)
  if (LR_keyroots1[i])
    for (j=1 ; j <= |T2| ; j++)
      if (LR_keyroots2[j])
        treedist(i,j);

return (tdist[i][j]);
```

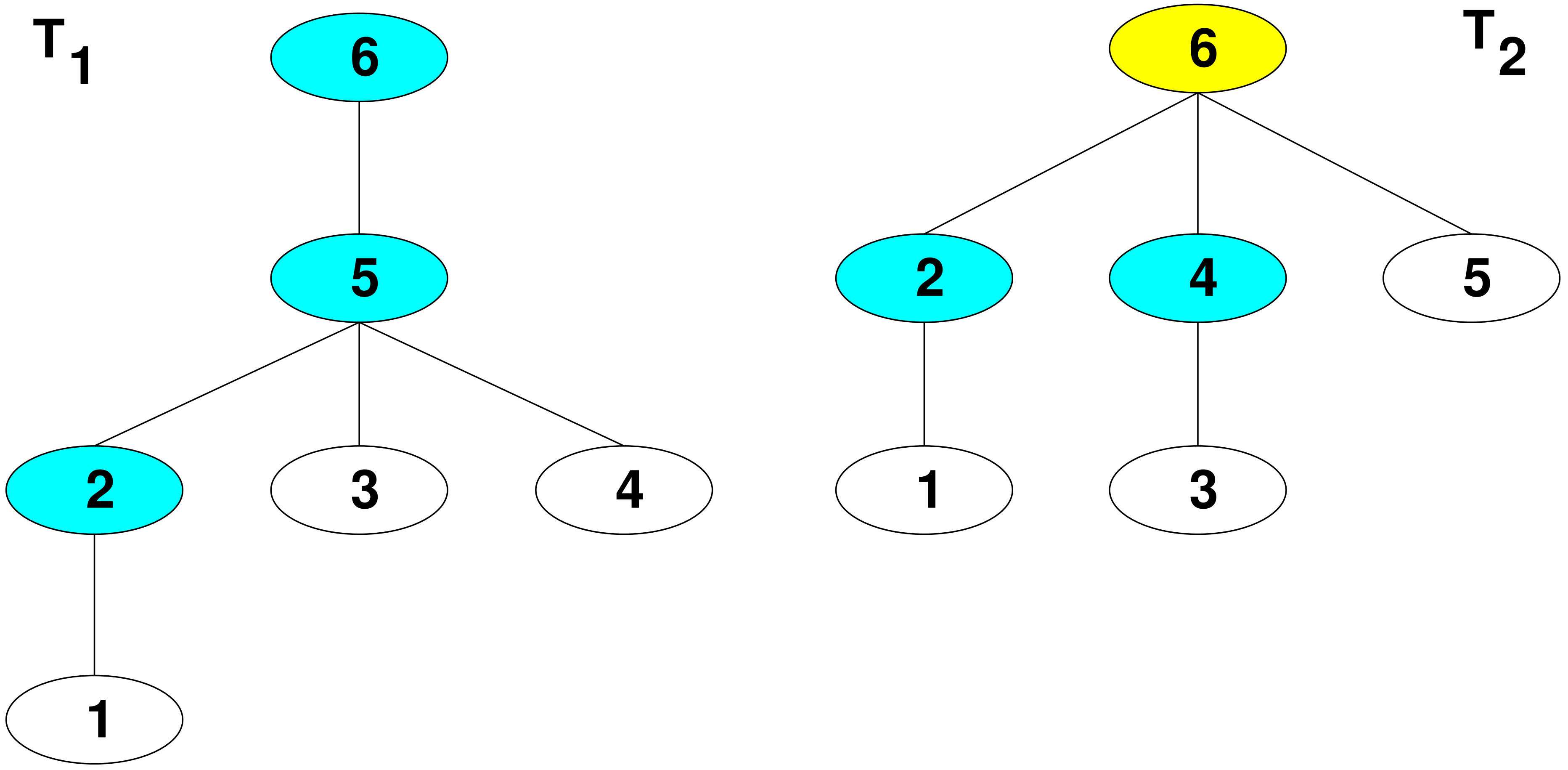
Algorithm (9)

```
treedist(pos1, pos2)

{
  bound1 = pos1 - left1[pos1] + 2;
  bound2 = pos2 - left2[pos2] + 2;

  // init fdist array to size [bound1][bound2]

  fdist[0][0] = 0;
  for (i=1, k=left1[pos1] ; i<bound1 ; i++, k++)
    fdist[i][0] = fdist[i-1][0] + c[k][0];
  for (j=1, l=left2[pos2] ; j<bound2 ; j++, l++)
    fdist[0][j] = fdist[0][j-1] + c[0][l];
```

Algorithm (10)

```
for (k=left1[pos1], i=1 ; k<=pos1 ; k++, i++)
  for (l=left2[pos2], j=1  ; l<=pos2 ; l++, j++)
    if ((left1[k]==left1[pos1]) && (left2[l]==left2[pos2])) {
      fdist[i][j] = MIN(fdist[i-1][j] + c[k][0],
                        fdist[i][j-1] + c[0][l],
                        fdist[i-1][j-1] + c[k][l]);
      tdist[k][l] = fdist[i][j];
    }
    else
      m = left1[k] - left1[pos1]; n = left2[l] -left2[pos2];
      if (m<0) m=0; if (n<0) n=0;
      fdist[i][j] = MIN(fdist[i-1][j] + c[k][0],
                        fdist[i][j-1] + c[0][l],
                        fdist[m][n] + tdist[k][l]);
}
```

Algorithm (11)



Example Problem

Algorithm (12)

```
Left array: 1 1 3 4 1 1
Left array: 1 1 3 3 5 1
LR_keyroots: 0 0 1 1 0 1   (i.e. {3,4,6})
LR_keyroots: 0 0 0 1 1 1   (i.e. {4,5,6})

Forestdist for 3 4
 0  1  2
 1  0  1

Treedist after 3 4
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  1  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
```

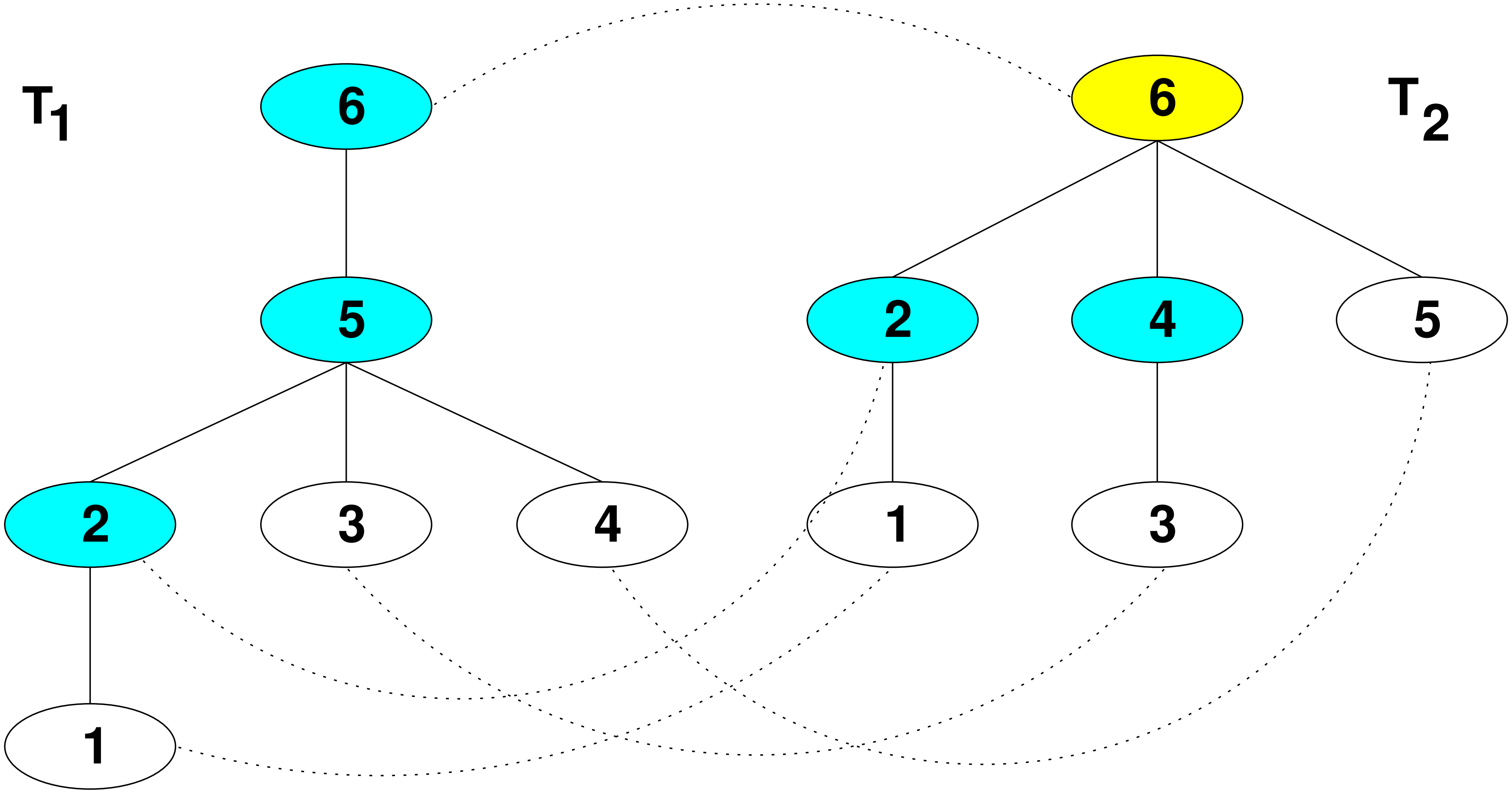
Algorithm (13)

```
Forestdist for 6 4
 0  1  2
 1  0  1
 2  1  0
 3  2  1
 4  3  2
 5  4  3
 6  5  4
Treedist after 6 4
 0  0  0  1  0  0
 0  0  1  0  0  0
 0  1  0  1  0  5
 0  1  0  1  0  5
 0  0  4  3  0  0
 0  0  5  4  0  0
```

Algorithm (14)

```
Forestdist for 6 6
 0  1  2  3  4  5  6
 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  2  1  2
 5  4  3  2  3  2  2
 6  5  4  3  4  3  3
Treedist after 6 6
 0  1  0  1  0  5
 1  0  1  0  1  4
 0  1  0  1  0  5
 0  1  0  1  0  5
 4  3  4  3  4  2
 5  4  5  4  5  3
Edit distance between trees is: 3
```

Algorithm (15)



Example Mapping

Complexity

Lemmata and definitions

- $|LR\_keyroots(T)| \leq |leaves(T)|$
- Collapsed depth:
  $LR\_colldepth(i) = |anc(i) \cap LR\_keyroots(T)|$
- Of tree T: $LR\_colldepth(T) = \max\{LR\_colldepth(i)\}$
- $LR\_colldepth(T) \leq \min(depth(T), leaves(T))$
- $Size(i)$ is number of nodes in $T[l(i)..i]$
- The following equality holds:

$$\sum_{i \in LR\_keyroots(T)} Size(i) = \sum_{j=1}^{|T|} |LR\_colldepth(j)|$$

Complexity (2)

Theorem

The space complexity is $O(|T_1| \times |T_2|)$

The time complexity is:

$$O(|T_1|\times|T_2|\times\min(depth(T_1), leaves(T_1))\times\min(depth(T_2), leaves(T_2)))$$

- Preprocessing takes linear time in $|T_1|$ or $|T_2|$
- Subtree distance takes $Size(i) \times Size(j)$
- Main loop calls this subroutine for all members of $LR\_keyroots$

Complexity (3)

$$
\begin{aligned}
\#\text{ops} &= \sum_{i \in LR\_keyroots(T_1)} \; \sum_{j \in LR\_keyroots(T_2)} Size(i) \times Size(j) \\
&= \sum_{i \in LR\_keyroots(T_1)} Size(i) \times \sum_{j \in LR\_keyroots(T_2)} Size(j) \\
&= \sum_{i=1}^{|T_1|} |LR\_colldepth(i)| \times \sum_{j=1}^{|T_2|} |LR\_colldepth(j)| \\
&\leq |T_1| \times |T_2| \times LR\_colldepth(T_1) \times LR\_colldepth(T_2) \\
&\quad \square
\end{aligned}
$$

Extensions

Generate mappings as solution

- Save optimal configuration for all $tdist$ values
- Save where the lookup occurred

Strings

- String has only one leaf
- Time complexity $O(|T_1| \times |T_2|)$
- $LR\_keyroots(S) = \{root\}$
- $l(i) = 1$, therefore only first case
- Tree algorithm is reduced to string algorithm

Extensions (2)

Approximate String Matching:

- Compute minimum number of edit operations between pattern and text where any prefix can be removed from the text

Approximate Tree Matching:

- Two alternatives for deletions
- Removing at node $T[i]$: remove subtree rooted at $T[i]$
- Pruning at node $T[i]$: remove all descendants of $T[i]$
- Algorithm follows the same template as the normal case
- Include a fourth alternative for removing/pruning

Extensions (3)

Unit cost optimizations

- Compute if trees are within distance k
- Use the following algorithm for the general case:

```
for (j=0 ; j<=log2(|T1|+|T2|) ; j++)
  if ((sol = treedist(T1, T2, 2^j)) <= 2^j)
    return (sol);
```

- To compute `treedist(T1, T2, k)`:

```
for (i=1 ; i<=|T1| ; i++)
  for (j=MAX(i-k, 1) ; j<=MIN(|T2|, i+k) ; j++)
    if (|left1[i]-left2[j]| <= k)
      _treedist(i, j, k-|left1[i] - left2[j]|);
```

Extensions (4)

- The computation in `_treedist(i, j, k)` computes only *relevant* values
- Running time is $O(k^2 \times \min(|T_1|, |T_2|) \times \min(L_1, L_2))$
- L is number of levels in the tree

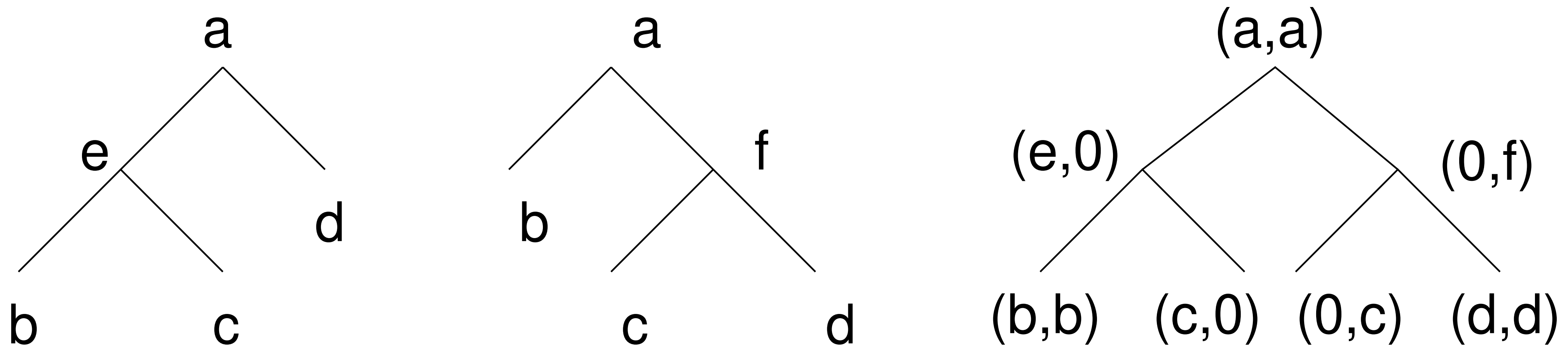
Further Optimizations

- Check if subtrees are equal
- Use suffix trees
- Algorithm constructed similar to Landau-Vishkin string algorithm
- Parallel implementation: $k^2 \times N$ processors
- Running time: $O((k^2 \times \log(k)) + \log(N))$

Tree Alignment

- Insert *spaces* into T_1 and T_2 such that the resulting trees T_1' and T_2' have the same structure
- Labels can be different
- *Overlay* T_1' and T_2'
- Assign scores for each pair of labels
- Value of alignment $\mathcal{A}$ is the sum of the scores
- An *optimal* alignment minimizes this sum
- Runs in time $O(|T_1| \times |T_2| \times (deg(T_1) + deg(T_2))^2)$

Tree Alignment (2)



Example of tree alignment

Tree Alignment (3)

- Alignment and edit are the same for strings
- edit distance $\leq$ alignment (for same costs)
- Alignment is restricted version of tree edit where insertions precede deletions
- Order not important for sequences
- Alignment charges more for operations at the top
- Algorithm structure is similar to tree edit

Summary

- Similarity of trees can be solved in different ways
- Standard approach is tree edit distance
- Operations: Modify, Insert, Delete
- Employs a dynamic programming approach
- Runs in polynomial time
- Parallelizations and further optimizations are possible
- Another interesting approach is tree alignment

Bibliography

- Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. SIAM J. Comput., 18:1245-1262, 1989
- Dennis Shasha and Kaizhong Zhang. Fast algorithms for the unit cost editing distance between trees. J. Algorithms, 11:581-621, 1990
- Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees — an alternative to tree edit. Theor. Comput. Sci., 143(1):137-148, 1995