Optimal Decomposition Strategy For Tree Edit Distance

Shaofeng Jiang

Supervised By Dr. Kaizhong Zhang

sjian7@uwo.ca

December 13, 2017

Overview

1. Introduction
2. Notation
3. Recursive Solution
4. Algorithms for the Tree Editing Distance
    - A Simple Algorithm
    - Zhang and Shasha's Algorithm
    - Klein's Algorithm
    - Demaine's Algorithm
5. Algorithmic Improvement
6. Applications in the RNA Secondary Structure Comparison
7. Evaluation
8. Conclusion

Ordered Labelled Tree

An ordered labelled tree is a tree in which the nodes are labeled and the left-to-right order among siblings is significant.

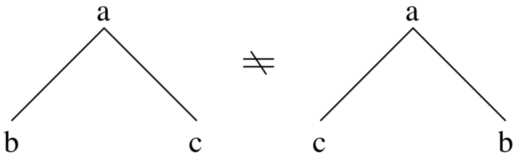


Figure: Ordered Labelled Tree

The ordered labelled tree can represent:

- an XML document
- a natural language parse
- RNA secondary structure
- and so on...

Tree Edit Distance

- The string edit distance is the minimum cost to transform one string to another.
  - Align two sequences of nucleotides

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

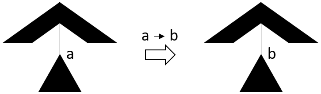
  - Resulting alignment

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

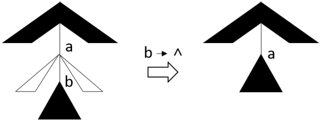
- The tree edit distance is introduced by Tai as a generalization of the string edit distance.
- The edit distance between two trees P and D is the minimum cost to change P to D via a sequence of basic edit operations.

Edit Operations

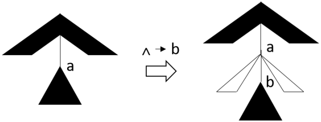
- Three valid edit operations supported:
  - **Substitution.** replace a label of a node by another label.



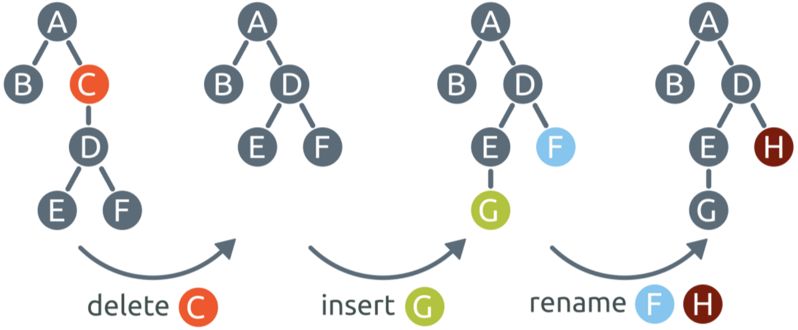
  - **Deletion.** deletion of a node



  - **Insertion.** insertion of a node.

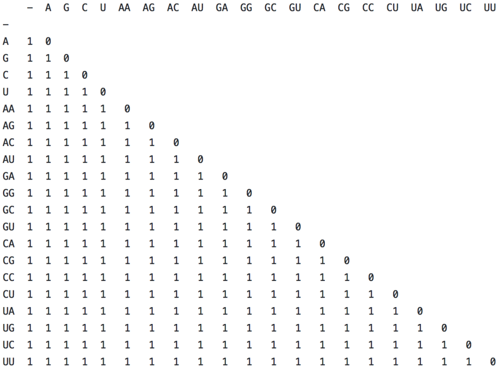


Edit Operations



Cost of Edit Operations

- Each edit operations has a cost.
- Let γ be a cost function of each operations. The cost function γ is a distance metric.
    - $\gamma(a \rightarrow b)$ can be 1 or other user defined value.

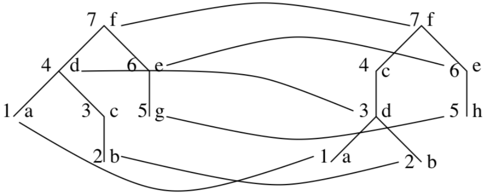


	–	A	G	C	U	AA	AG	AC	AU	GA	GG	GC	GU	CA	CG	CC	CU	UA	UG	UC	UU
–																					
A	1	0																			
G	1	1	0																		
C	1	1	1	0																	
U	1	1	1	1	0																
AA	1	1	1	1	1	0															
AG	1	1	1	1	1	1	0														
AC	1	1	1	1	1	1	1	0													
AU	1	1	1	1	1	1	1	1	0												
GA	1	1	1	1	1	1	1	1	1	0											
GG	1	1	1	1	1	1	1	1	1	1	0										
GC	1	1	1	1	1	1	1	1	1	1	1	0									
GU	1	1	1	1	1	1	1	1	1	1	1	1	0								
CA	1	1	1	1	1	1	1	1	1	1	1	1	1	0							
CG	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0						
CC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0					
CU	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0				
UA	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0			
UG	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0		
UC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	
UU	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

- The edit distance is the sum of the cost of the minimum cost sequence.

Mapping in Trees

- Mapping between trees.
- The cost of a mapping is the sum of its insertion, deletion and substitutions.

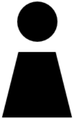


Edit sequence: $(c \rightarrow \lambda), (g \rightarrow h), (\lambda \rightarrow c)$

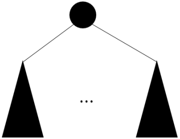
Figure: The Edit Sequence and its Cost

Notation

- **Tree**
  - $l(f)$



  - $l(A_1 \circ A_2 \circ \cdots \circ A_n)$

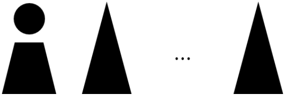


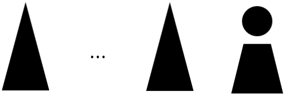
Let T be a tree,

$r(T)$	the root of T
T°	$T - r(T)$
$lr(T)$	the root of the leftmost tree in the forest T°
$rr(T)$	the root of the rightmost tree in the forest T°

Notation

- **Forest**
  - $l(f) \circ t$
  - $t \circ l(f)$





Let F be a forest,

$\|F\|$	the number of nodes in the forest F
$\#leaves(F)$	the number of leaves in the forest F
$F(i)$	the sub-tree of F rooted at node i
$F - i$	the forest after deleting node i
$lr(F)$	the root of the leftmost tree in the forest F
$rr(F)$	the root of the rightmost tree in the forest F

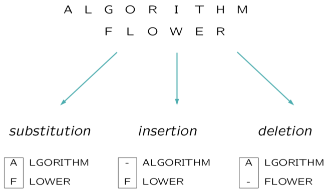
String Edit Distance and its Recursive Solution

- String edit distance is the minimum cost to change one string to another via a sequence of basic edit operations.
- The recursive solution of the string edit problem $d(S_1, S_2)$ is

$$d(S_1, S_2) = min \begin{cases} d(S_1 - u, S_2) + \delta(u, \varnothing) \\ d(S_1, S_2 - v) + \delta(\varnothing, v) \\ d(S_1 - u, S_2 - v) + \delta(u, v) \end{cases}$$

String Edit Distance and its Recursive Solution

- If u and v is both the first element of string S_1 and S_2, then it is a leftmost decomposition.



A L G O R I T H M
F L O W E R

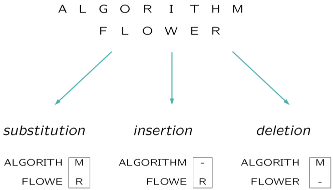
substitution *insertion* *deletion*

A LGORITHM
F LOWER

- ALGORITHM
F LOWER

A LGORITHM
- FLOWER

- If u and v is both the last element of string S_1 and S_2, then it is a rightmost decomposition.



A L G O R I T H M
F L O W E R

substitution *insertion* *deletion*

ALGORITH M
FLOWE R

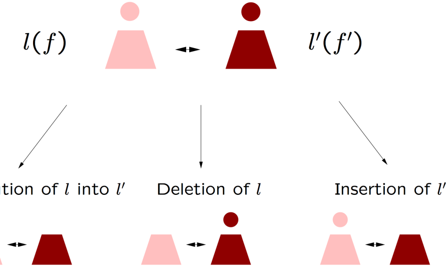
ALGORITHM -
FLOWE R

ALGORITH M
FLOWER -

Tree Edit Distance and its Recursive Solution

For tree-to-tree distance, Let T_1 and T_2 are both trees of the form $l(f)$ and $l'(f')$,

$$d(l(f), l'(f')) = min \begin{cases} d(f, l'(f')) + \delta(l, \varnothing) \\ d(l(f), l'(f')) + \delta(\varnothing, l') \\ d(f, f') + \delta(l, l') \end{cases}$$



$l(f)$

$l'(f')$

Substitution of l into l'

Deletion of l

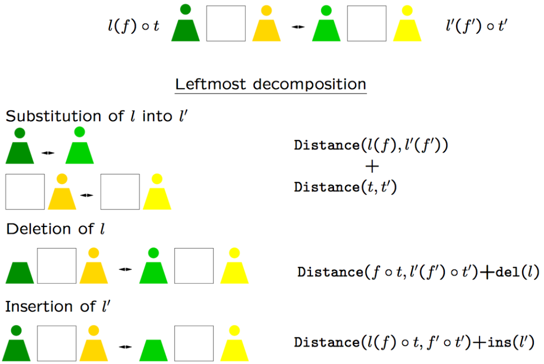
Insertion of l'

Figure: Tree-to-Tree Edit Distance

Tree Edit Distance and its Recursive Solution

For forest-to-forest distance, Let F_1 and F_2 are both trees of the form $l(f) \circ t$ and $l'(f') \circ t'$,

$$d(l(f) \circ t, l'(f') \circ t') = min \begin{cases} d(f \circ t, l'(f') \circ t') + \delta(l, \varnothing) \\ d(l(f) \circ t, f' \circ t') + \delta(\varnothing, l') \\ d(t, t') + d(l(f), l'(f')) \end{cases}$$



$l(f) \circ t$ $l'(f') \circ t'$

Leftmost decomposition

Substitution of l into l'

$\mathtt{Distance}(l(f), l'(f'))$
$+$
$\mathtt{Distance}(t, t')$

Deletion of l

$\mathtt{Distance}(f \circ t, l'(f') \circ t') + \mathtt{del}(l)$

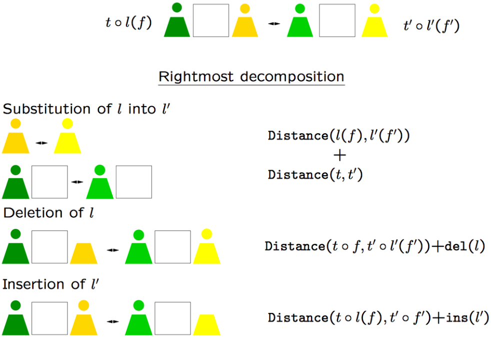
Insertion of l'

$\mathtt{Distance}(l(f) \circ t, f' \circ t') + \mathtt{ins}(l')$

Tree Edit Distance and its Recursive Solution

For forest-to-forest distance, Let F_1 and F_2 are both trees of the form $l(f) \circ t$ and $l'(f') \circ t'$,

$$d(t \circ l(f), t' \circ l'(f')) = min \begin{cases} d(t \circ f, t' \circ l'(f')) + \delta(l, \varnothing) \\ d(t \circ l(f), t' \circ f') + \delta(\varnothing, l') \\ d(t, t') + d(l(f), l'(f')) \end{cases}$$



$t \circ l(f)$ $t' \circ l'(f')$

Rightmost decomposition

Substitution of l into l'

$\texttt{Distance}(l(f), l'(f'))$ + $\texttt{Distance}(t, t')$

Deletion of l

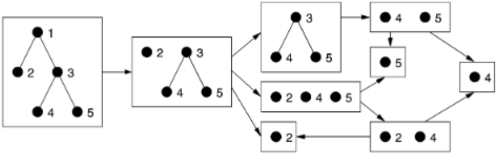
$\texttt{Distance}(t \circ f, t' \circ l'(f')) + \texttt{del}(l)$

Insertion of l'

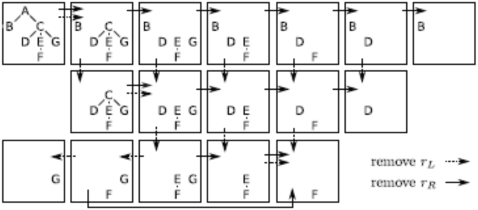
$\texttt{Distance}(t \circ l(f), t' \circ f') + \texttt{ins}(l')$

Relevant Sub-forests and Full Decomposition

- Relevant sub-forests

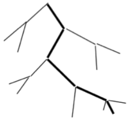


- Full decomposition

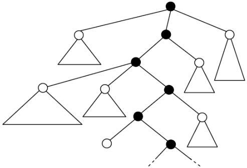


Root-leaf Path and Relevant Sub-trees

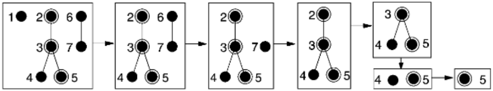
- Root-leaf path



- Relevant sub-trees



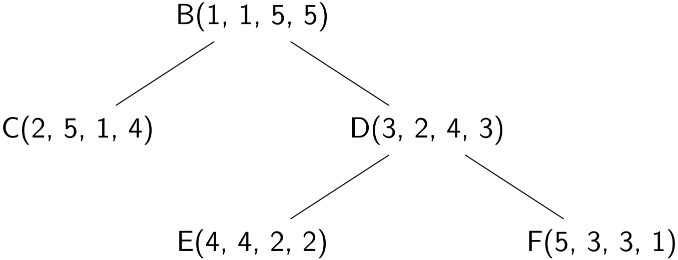
- Path decomposition



Tree Indexing

- $preOrder\_left\_to\_right \rightarrow preL$
- $preOrder\_right\_to\_left \rightarrow preR$
- $postOrder\_left\_to\_right \rightarrow postL$
- $postOrder\_right\_to\_left \rightarrow postR$

Consider a tree T(preL, preR, postL, postR)



B(1, 1, 5, 5)

C(2, 5, 1, 4)

D(3, 2, 4, 3)

E(4, 4, 2, 2)

F(5, 3, 3, 1)

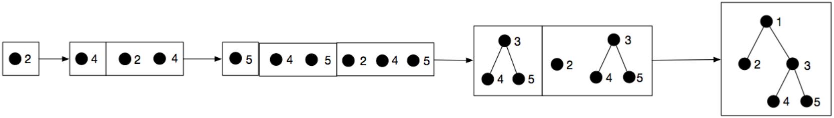
Note:

- $preL + postR = treeSize + 1$
- $preR + postL = treeSize + 1$

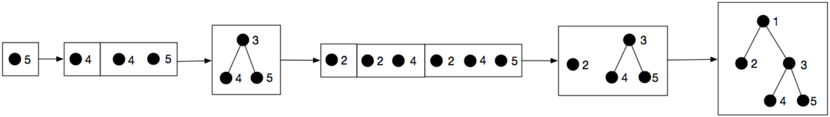
Bottom-up Enumeration

Two orders of the enumeration of the full decomposition:

- Prefix-suffix order



- Suffix-prefix order

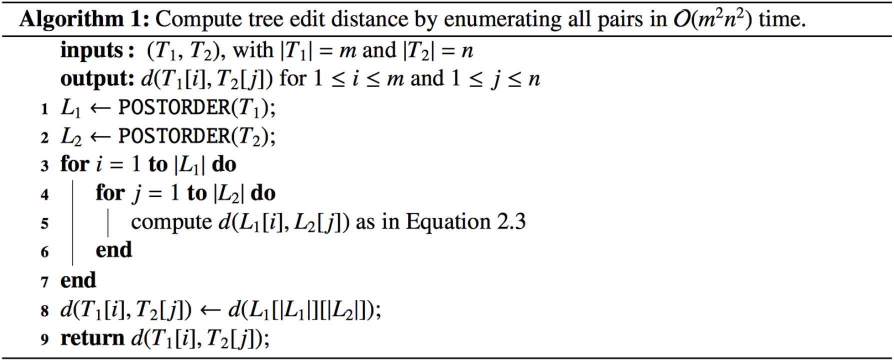


A Simple Algorithm

Main Idea:

Enumerate each pairs of sub-forests in tree A and B in prefix-suffix post order or suffix-prefix post order.

Pseudocode:



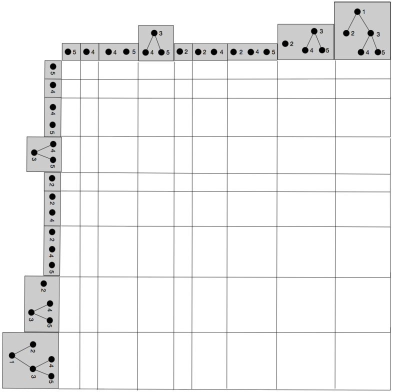
Algorithm 1: Compute tree edit distance by enumerating all pairs in $O(m^2n^2)$ time.

```
inputs : $(T_1, T_2)$, with $|T_1| = m$ and $|T_2| = n$
output: $d(T_1[i], T_2[j])$ for $1 \leq i \leq m$ and $1 \leq j \leq n$
$L_1 \leftarrow \text{POSTORDER}(T_1)$;
$L_2 \leftarrow \text{POSTORDER}(T_2)$;
for $i = 1$ to $|L_1|$ do
    for $j = 1$ to $|L_2|$ do
        compute $d(L_1[i], L_2[j])$ as in Equation 2.3
    end
end
$d(T_1[i], T_2[j]) \leftarrow d(L_1[|L_1|][|L_2|])$;
return $d(T_1[i], T_2[j])$;
```

A Simple Algorithm

Implementation:

- Implemented in dynamic programming.
- Runs in $\mathcal{O}(m^2 n^2)$ time using $\mathcal{O}(m^2 n^2)$ space.

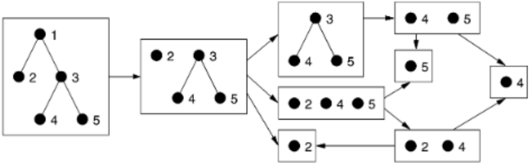


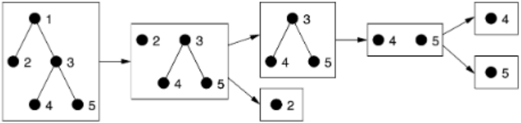
- The upper bound of the post-order enumeration method for the tree edit distance problem is $\mathcal{O}(n^4)$

Improved Algorithmic Path Strategies

Key Ideas:

- Different path decomposition creates different number of relevant sub-forests.
- Take advantage of the overlap among sub-forests that are contained in the same sub-tree, and the overlap of sub-trees as well.

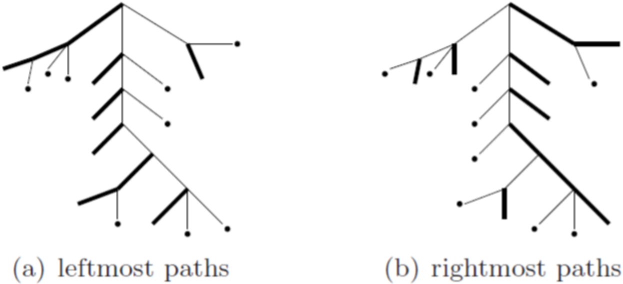




Zhang and Shasha's Algorithm

Key Ideas:

- Fixed direction in each recursive calls.
  - Recursive right decomposition - leftmost paths.
  - Recursive left decomposition - rightmost paths.



(a) leftmost paths

(b) rightmost paths

Zhang and Shasha's Algorithm

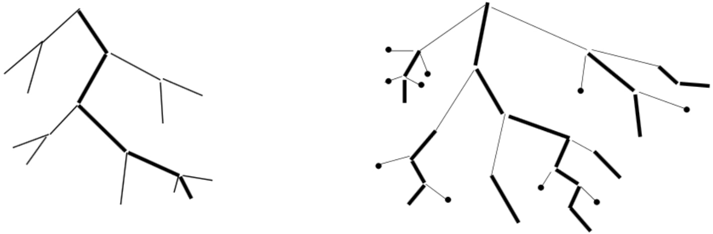
Implementation:

- Implemented via dynamic programming.
- Enumerates sub-trees pairs rooted at key roots.
  - Leftmost paths: either tree root or has a left sibling
  - Rightmost paths: either tree root or has a right sibling
- Uses two memorized tables to store intermediate results:
  - a temporary table to store forest-forest distance.
  - a permanent table to store tree-tree distance.
- Runs in $\mathcal{O}(|A| * |B| * min\{depth(A), leaves(A)\} * min\{depth(B), leaves(B)\})$ time and $\mathcal{O}(|A| * |B|)$ space.

Klein's Algorithm

Key Ideas:

- Zhang and Shasha's algorithm depends on shapes of trees.
- Heavy node.
- Heavy paths generate the least number of sub-forests on one tree.



Klein's Algorithm

Implementation:

- Implemented via dynamic programming.
- applies heavy paths to the larger tree and fully decomposes the smaller tree.
- Uses temporary tables to store forest-forest distance and a permanent tables to store tree-tree distance.
- Runs in $\mathcal{O}(|A| \log(|A|) * |B|^2)$ time using $\mathcal{O}(|A| * |B|)$ space.

Demaine's Algorithm

Key Ideas:

- Klein's algorithm consider the shape of one tree with no consideration on the other tree.
- Heavy paths can be applied on both trees.
- Always applies heavy paths on the larger tree.

Implementation:

- Implemented via dynamic programming.
- Runs in $\mathcal{O}(|A|\,|B|^2\,(1+\log(\frac{|A|}{|B|}))$ time using $\mathcal{O}(|A| * |B|)$ space.

Conclusion

	Time	Space	Comments
Simple Algorithm	$\mathcal{O}(n^4)$	$\mathcal{O}(n^4)$	first algorithm
Zhang	$\mathcal{O}(n^2 \log^2(n))$	$\mathcal{O}(n^2)$	efficient for balanced trees
Klein	$\mathcal{O}(n^3 \log(n))$	$\mathcal{O}(n^2)$	no consider smaller trees
Demaine	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	worse case is frequent

Table: State-of-the-Art Algorithms in the Tree Edit Distance Problem

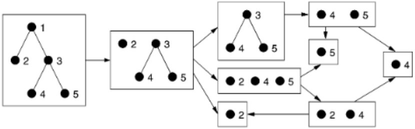
Optimal Root-leaf Path

Key Idea:

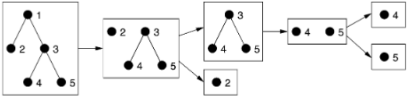
- heavy paths is a kind of greedy algorithm which usually lead to the local optimum rather than global optimum.
- quantifies each possible paths and selects the path with the least number of sub-problems is selected
- implemented via dynamic programming.
- Preprocessing time is bounded by $\mathcal{O}(|A|\,|B|)$

Relevant Leftmost/Rightmost/Special Sub-forests

- Path decomposition → relevant sub-forests
- Recursive leftmost path decomposition → relevant leftmost sub-forests



- Recursive rightmost path decomposition → relevant rightmost sub-forests



- Full decomposition → relevant special sub-forests

Relevant Leftmost/Rightmost/Special Sub-forests

Let T be a tree of the form $l(T_1 \circ \cdots \circ T_n)$ and the size of T is n,

- The Number of Relevant Leftmost Sub-forests($\#left(T)$)

$$\#left(T) = |T| - |T_1| + \sum_{i=1}^{n} \#left(T_i)$$

- The Number of Relevant Rightmost Sub-forests($\#right(T)$)

$$\#right(T) = |T| - |T_n| + \sum_{i=1}^{n} \#right(T_i)$$

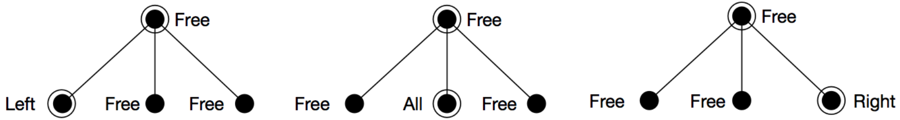
- Relevant Special Sub-forests($\#special(T)$)

$$\#spec(T) = \frac{n(n+3)}{2} - \sum_{i \in T} |T(i)|$$

Status of a Node in a Tree

Let T be a tree with a root-leaf path,

- Free node: the node is the root of T, or is not on the root-leaf path.
- Left node:the node is on the root-leaf path and is the leftmost child of its parent.
- Right node:the node is on the root-leaf path and is the rightmost child of its parent.
- All node:the node is on the root-leaf path and is neither the leftmost child nor the rightmost child of its parent.

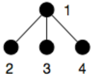


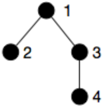
Quantify the Number of Sub-Problem

- Seven memorized tables to store intermediate results.
    - Free
    - LeftA
    - RightA
    - AllA
    - LeftB
    - RightB
    - AllB
- Dynamic programming.

Quantify the Number of Sub-Problem

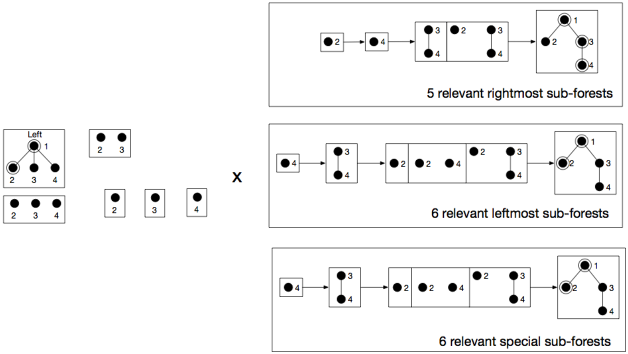
Let A be a tree with a root-leaf path and B be another tree,





Quantify the Number of Sub-Problem

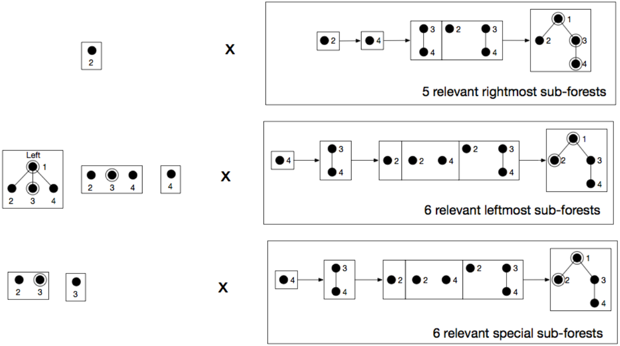
Let A be a tree with a root-leaf path and B be another tree.


Left
x
5 relevant rightmost sub-forests
6 relevant leftmost sub-forests
6 relevant special sub-forests

$$\begin{aligned}
\#leftA(1,1) &= Free(3,1) + Free(4,1) + LeftA(2,1) \\
&+ |A - A(2)| * \#left(B) \\
&= \#left(B) + \#left(B) + \#left(B) + 3 * \#left(B)
\end{aligned}$$

Quantify the Number of Sub-Problem

Let A be a tree with a root-leaf path and B be another tree.



5 relevant rightmost sub-forests

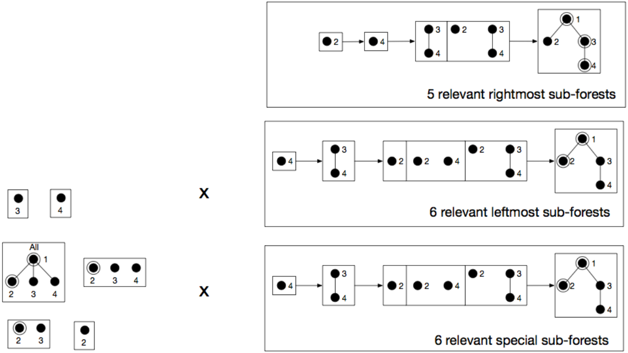
6 relevant leftmost sub-forests

6 relevant special sub-forests

$$
\begin{aligned}
\#leftA(1,1) &= Free(2,1) + Free(4,1) + AllA(3,1) \\
&+ |1 + A(4)| * \#left(B) + |A(2)| * \#special(B) \\
&= \#right(B) + \#left(B) + \#special(B) \\
&+ \#special(B) + 2 * \#left(B)
\end{aligned}
$$

Quantify the Number of Sub-Problem

Let A be a tree with a root-leaf path and B be another tree.



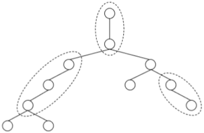
5 relevant rightmost sub-forests
x
6 relevant leftmost sub-forests
All
x
6 relevant special sub-forests

$$
\begin{aligned}
\#AllA(1,1) &= Free(3,1) + Free(4,1) + AllA(2,1) \\
&+ |A - A(2)| * \#special(B) \\
&= \#left(B) + \#left(B) + \#special(B) + 3 * \#special(B)
\end{aligned}
$$

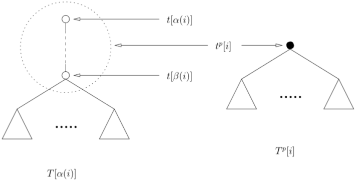
Vertical Reduction on Trees

Key Ideas:

- A large number of non-branching nodes can be compressed into one node.



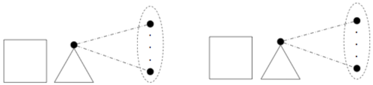
- The compact representation of the original tree can aid in improving the running time for computing the tree edit distance.

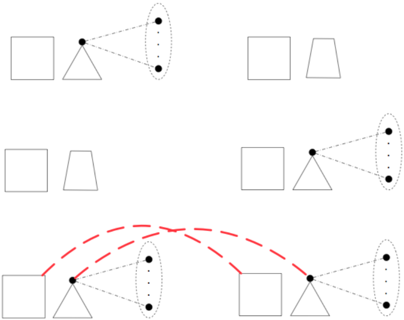


- $T \rightarrow \widetilde{T}$

Algorithm

Forest-to-Forest Distance:

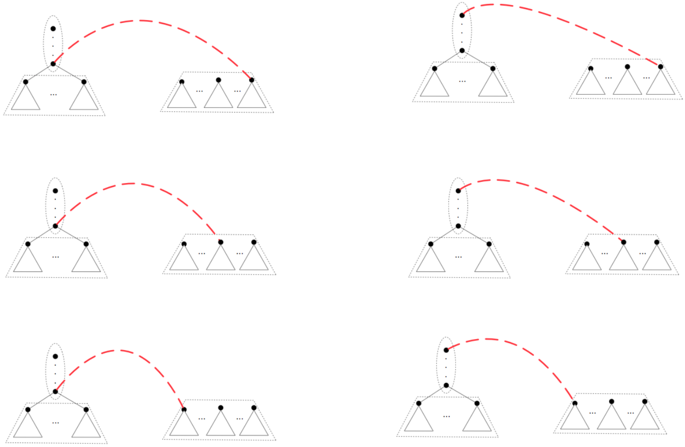




Algorithm

Tree-to-Tree Distance:

- Compute $d(\widetilde{T_1}, \widetilde{T_2}^{\circ})$

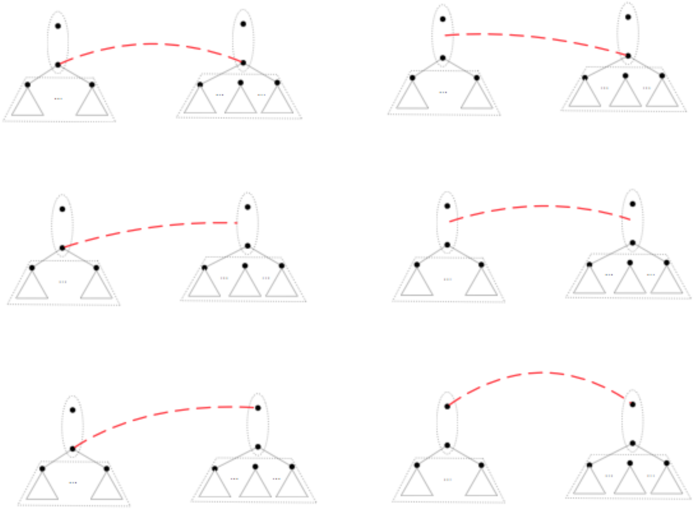


- The computation of $d(\widetilde{T_1}^{\circ}, \widetilde{T_2})$ is symmetric.

Algorithm

Tree-to-Tree Distance:

- Compute $d(\widetilde{T_1}, \widetilde{T_2})$

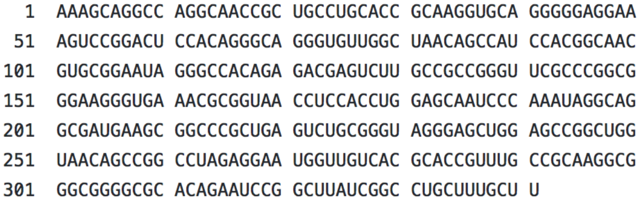


Implementation

- three memorized tables to store intermediate results:
  - D_t: a $(|T_1|+1)*(|T_2|+1)$ two dimensional permanent array.
  - $\widetilde{D}_t$: a $(\left|\widetilde{T}_1\right|+1)*(\left|\widetilde{T}_2\right|+1)$ two dimensional permanent array.
  - $\widetilde{D}_f$: a $(\left|\widetilde{T}_1\right|+1)*(\left|\widetilde{T}_2\right|+1)$ two dimensional temporary array.
- Dynamic programming.

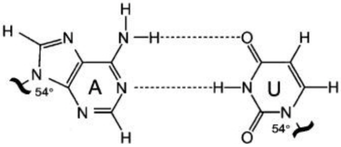
RNA

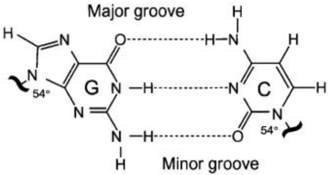
- RNA is an essential molecule in organisms.
  - Cellular organisms
  - Viruses
- RNA is assembled as a chain of nucleotides.



```
  1   AAAGCAGGCC AGGCAACCGC UGCCUGCACC GCAAGGUGCA GGGGGAGGAA
 51   AGUCCGGACU CCACAGGGCA GGGUGUUGGC UAACAGCCAU CCACGGCAAC
101   GUGCGGAAUA GGGCCACAGA GACGAGUCUU GCCGCCGGGU UCGCCCGGCG
151   GGAAGGGUGA AACGCGGUAA CCUCCACCUG GAGCAAUCCC AAAUAGGCAG
201   GCGAUGAAGC GGCCCGCUGA GUCUGCGGGU AGGGAGCUGG AGCCGGCUGG
251   UAACAGCCGG CCUAGAGGAA UGGUUGUCAC GCACCGUUUG CCGCAAGGCG
301   GGCGGGGCGC ACAGAAUCCG GCUUAUCGGC CUGCUUUGCU U
```

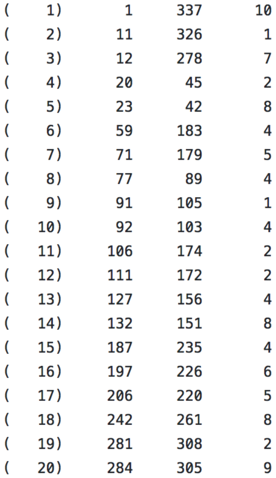
- RNA is unstable.





RNA Secondary Structure

- a list of base pairs satisfies the following constraints:
  - A base cannot participate in more than one base pair,
  - No two base pairs cross.



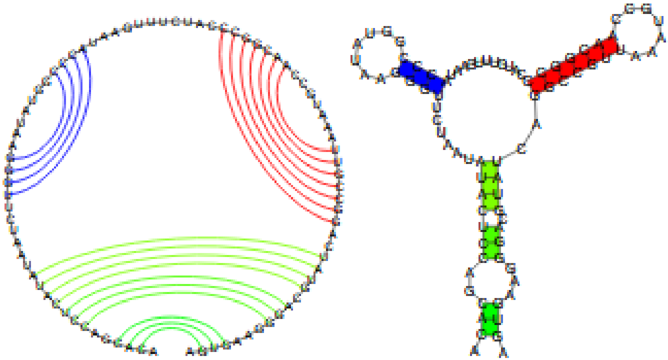
```
(    1)       1     337      10
(    2)      11     326       1
(    3)      12     278       7
(    4)      20      45       2
(    5)      23      42       8
(    6)      59     183       4
(    7)      71     179       5
(    8)      77      89       4
(    9)      91     105       1
(   10)      92     103       4
(   11)     106     174       2
(   12)     111     172       2
(   13)     127     156       4
(   14)     132     151       8
(   15)     187     235       4
(   16)     197     226       6
(   17)     206     220       5
(   18)     242     261       8
(   19)     281     308       2
(   20)     284     305       9
```

RNA Secondary Structure Representation

- String representation

```
ACACGACCUCAUAUAAUCUUGGGAAUAUGGCCCAUAAGUUUCUACCCGGCAACCGUAAAUUGCCGGACUAUGCAGGGAAGUGA
.(((..(((((((......((((.......))))...........(((((((.......)))))))..)))).)))...))).
```

- Graph representation



Tree Representation of the RNA Secondary Structure

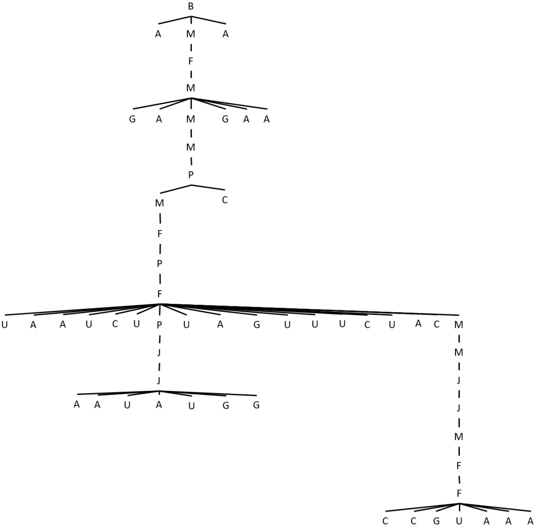
- Base pair labelling

Base Pairs	Label
$A = U$	F
$G \equiv C$	J
$U = A$	P
$C \equiv G$	M

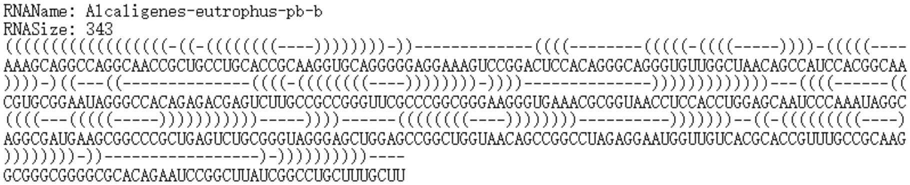
Table: The Label of Base Pair

- Tree representation

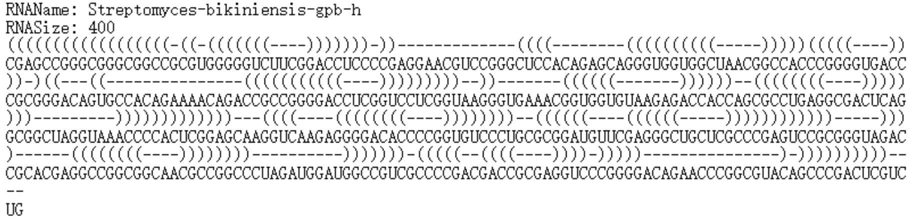
Tree Representation of the RNA Secondary Structure



Experiment

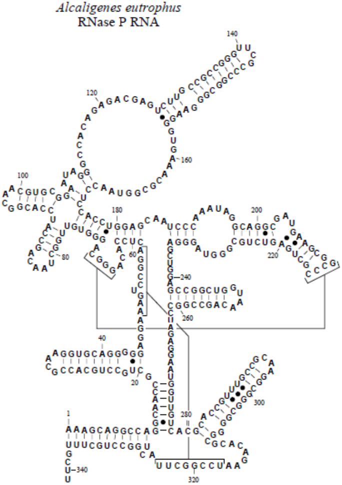


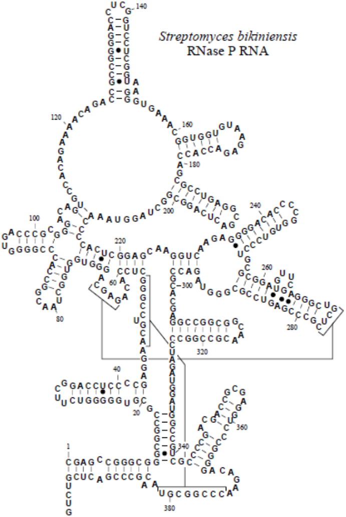
```
RNAName: Alcaligenes-eutrophus-pb-b
RNASize: 343
((((((((((((((((((-((-((((((((----))))))))-))-------------((((--------(((((-((((-----))))-(((((----
AAAGCAGGCCAGGCAACCGCUGCCUGCACCGCAAGGUGCAGGGGGAGGAAAGUCCGGACUCCACAGGGCAGGGUGUUGGCUAACAGCCAUCCACGGCAA
))))-)((---((--------------((((-((((((((----))))))))-))))--------------)))))))))))))---((((------((
CGUGCGGAAUAGGGCCACAGAGACGAGUCUUGCCGCCGGGUUCGCCCGGCGGGAAGGGUGAAACGCGGUAACCUCCACCUGGAGCAAUCCCAAAUAGGC
((((---(((((-----)))))))))))-----))))------((((((((----))))))))----------)))))))--((-(((((((((----)
AGGCGAUGAAGCGGCCCGCUGAGUCUGCGGGUAGGGAGCUGGAGCCGGCUGGUAACAGCCGGCCUAGAGGAAUGGUUGUCACGCACCGUUUGCCGCAAG
))))))))-))-----------------)-))))))))))----
GCGGGCGGGGCGCACAGAAUCCGGCUUAUCGGCCUGCUUUGCUU
```



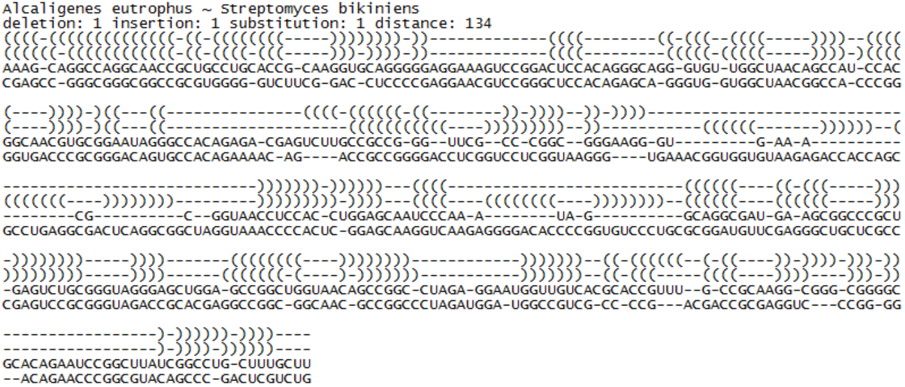
```
RNAName: Streptomyces-bikiniensis-gpb-h
RNASize: 400
((((((((((((((((((-((-(((((((----)))))))-))-------------((((--------((((((((((-----)))))(((((----))
CGAGCCGGGCGGGCGGCCGCGUGGGGGUCUUCGGACCUCCCCGAGGAACGUCCGGGCUCCACAGAGCAGGGUGGUGGCUAACGGCCACCCGGGGUGACC
))-)((---((---------------(((((((((((----)))))))))--))-------((((((-------))))))--((((((((----)))))
CGCGGGACAGUGCCACAGAAAACAGACCGCCGGGGACCUCGGUCCUCGGUAAGGGUGAAACGGUGGUGUAAGAGACCACCAGCGCCUGAGGCGACUCAG
)))---------)))))))))))))---((((----((((((((----))))))))--((((((----((((((-----))))))))))))-----)))
GCGGCUAGGUAAACCCCACUCGGAGCAAGGUCAAGAGGGGACACCCCGGUGUCCCUGCGCGGAUGUUCGAGGGCUGCUCGCCCGAGUCCGCGGGUAGAC
)------((((((((----))))))))----------)))))))-(((((--((((----))))-)))))---------------)-))))))))))--
CGCACGAGGCCGGCGGCAACGCCGGCCCUAGAUGGAUGGCCGUCGCCCCGACGACCGCGAGGUCCCGGGGACAGAACCCGGCGUACAGCCCGACUCGUC
--
UG
```

Experiment


Alcaligenes eutrophus
RNase P RNA
1
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340


Streptomyces bikiniensis
RNase P RNA
1
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380

Experiment Result



```
Alcaligenes eutrophus ~ Streptomyces bikiniens
deletion: 1 insertion: 1 substitution: 1 distance: 134
((((-((((((((((((((-((-((((((((-----))))))))-))-------------((((-------((-(((--((((-----))))--((((
((((((-((((((((((((-((-((((-(((-----)))-))))-))-------------((((--------(((((-(((((-----))))-)((((
AAAG-CAGGCCAGGCAACCGCUGCCUGCACCG-CAAGGUGCAGGGGAGGAAAGUCCGGACUCCACAGGGCAGG-GUGU-UGGCUAACAGCCAU-CCAC
CGAGCC-GGGCGGGCGGCCGCGUGGGG-GUCUUCG-GAC-CUCCCCGAGGAACGUCCGGGCUCCACAGAGCA-GGGUG-GUGGCUAACGGCCA-CCCGG

(----))))-)((---((---------------((((-((((((-((--------))-))))--))-))))---------------------------
(----))))-)((---((--------------------(((((((((((----)))))))))--))-----------((((((------))))))--(
GGCAACGUGCGGAAUAGGGCCACAGAGA-CGAGUCUUGCCGCCG-GG--UUCG--CC-CGGC--GGGAAGG-GU---------G-AA-A---------
GGUGACCCGCGGGACAGUGCCACAGAAAAC-AG----ACCGCCGGGGACCUCGGUCCUCGGUAAGGG----UGAAACGGUGGUGUAAGAGACCACCAGC

----------------------------)))))))-))))))---((((-------------------------((((((----((-(((-----)))
(((((((----))))))))---------)))))))))-))))---((((----((((((((----)))))))--((((((----((((((-----)))
--------CG----------C--GGUAACCUCCAC-CUGGAGCAAUCCCAA-A--------UA-G---------GCAGGCGAU-GA-AGCGGCCCGCU
GCCUGAGGCGACUCAGGCGGCUAGGUAAACCCCACUC-GGAGCAAGGUCAAGAGGGGACACCCCGGUGUCCCUGCGCGGAUGUUCGAGGGCUGCUCGCC

-))))))))-----))))-------((((((((----))))))))-----------)))))))--((-((((((--(-((----))-))))-)))-))
)))))))))-----))))------(((((((-(----)-)))))))----------)))))))--((-(((-----((((----))))----)))-))
-GAGUCUGCGGGUAGGGAGCUGGA-GCCGGCUGGUAACAGCCGGC-CUAGA-GGAAUGGUUGUCACGCACCGUUU--G-CCGCAAGG-CGGG-CGGGGC
CGAGUCCGCGGGUAGACCGCACGAGGCCGGC-GGCAAC-GCCGGCCCUAGAUGGA-UGGCCGUCG-CC-CCG---ACGACCGCGAGGUC---CCGG-GG

-----------------)-))))))-))))----
-----------------)-))))-))))))----
GCACAGAAUCCGGCUUAUCGGCCUG-CUUUGCUU
--ACAGAACCCGGCGUACAGCCC-GACUCGUCUG
```

Evaluation

Tree size: 246 and 282

Algorithm	$\#Rel.sub$	**Time[sec]**
$Zhang - L$	849282	0.08
$Zhang - R$	2039089	0.13
$Demaine$	7175224	0.65
$Our\ Algorithm(Before\ Compression)$	553526	0.04
$Zhang - L(Compressed)$	428766	0.05
$Zhang - R(Compressed)$	686562	0.05
$Our\ Algorithm(After\ Compression)$	291329	0.03

Evaluation

Tree size: 1041 and 1012

Algorithm	$\#Rel.sub$	**Time[sec]**
$Zhang - L$	41889364	3.2
$Zhang - R$	42972591	3.3
$Demaine$	273276733	20.98
$Our\ Algorithm(Before\ Compression)$	32254346	2.3
$Zhang - L(Compressed)$	17769036	0.47
$Zhang - R(Compressed)$	18154025	0.51
$Our\ Algorithm(After\ Compression)$	13546825	0.34

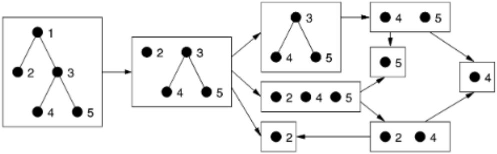
Conclusion

- Tree edit distance quantifies the similarity between two ordered labelled trees.
- Three kinds of root-leaf path are introduced to take the advantages of the overlapping relevant sub-forests and sub-trees in different ways.
- The optimal root-leaf path can be found via dynamic programming and can be used for the tree edit distance problem.
- The vertical reduction on trees can significantly reduced the running time.
- The comparison between RNA secondary structures is an application of the tree edit distance.
- Our strategies on compressed trees are by far the best decomposition strategy, creating the least number of relevant sub-problems.

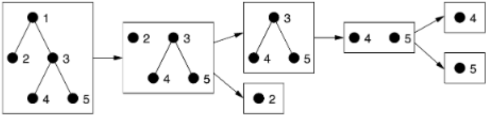
Thank you. Questions?

Path Decomposition

- Leftmost path decomposition



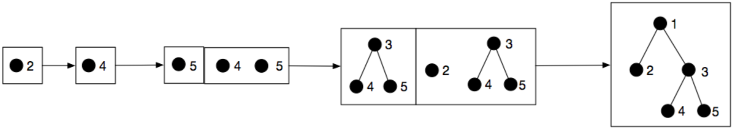
- Rightmost path decomposition



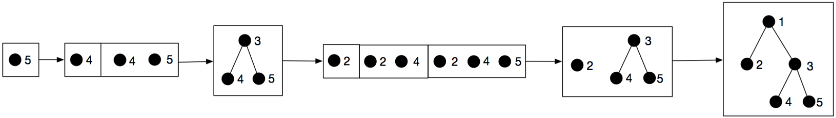
Bottom-up Enumeration

Two orders of the enumeration of the path decomposition:

- LR-postorder enumeration

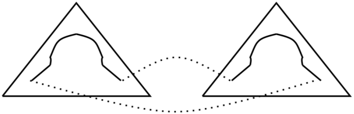


- RL-postorder enumeration

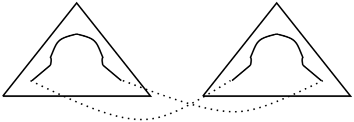


Matching in Trees

- **Input:** Two labelled ordered trees T and P.
- **Constraints:**
  - One-to-one relationship
  - Sibling order is preserved.

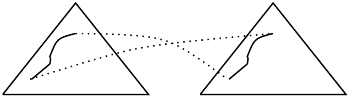


not valid

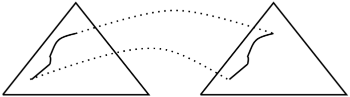


valid

  - Ancestor order is preserved.



not valid



valid

- **Output:** Mapping between T and P.