CS5238 Combinatorial methods in bioinformatics 2003-2004 Semester 1

Lecture 6: RNA structure comparison - September 26, 2003

Lecturer: Prof.Wing-Kin Sung Scribe: Li Guoliang, Nirmala Ramakrishnan, Rohit Joshi, V S Sundararajan

6.1 Introduction

In "Lecture 5: RNA Secondary Structure Prediction", we study how RNA structure can be predicted using energy minimization methods. In this lecture we will learn how to predict structure of RNA sequence given another RNA sequence with known structure and having similar function. In biology we conjecture that if two RNA's look similar then they may have similar function, or if two RNA's have similar function then by comparing their sequences, we can infer their structure. Based on above conjecture we can state three types of RNA structure comparison problems.

1. Deduce structure by comparing several homologous RNA plain sequences.
2. Compare a RNA plain sequence S with a RNA structure T so that the structure of S can be inferred.
3. Compare two RNA structures to know how similar they are.

RNA and protein structures are represented as arc-annotated sequences. Given a sequence S, an arc-annotation (or arc set) P can be defined as a set of arcs,where a pair of positions in S is called an arc. It is assumed that the arcs do not share endpoints. (see Figure 6.1) gives an example.

Figure 6.1: Example of an arc-annotated sequence S = ACGAGACU with set of arcs P = (1, 8), (2, 5), (3, 7)

There are different types of arc-annotated sequences. Given an arc-annotated sequence (S, P), it is:

1. Plain if P $=\phi$. (see Figure 6.2(a))
2. Nested (without pseudo knot)
   if for all ($i_1, j_1) and (i_2, j_2) \in P$,
   $(i_1 < i_2 < j_2 < j_1)$ or $(i_2 < i_1 < j_1 < j_2)$. (see Figure 6.2(b))
3. Crossing, otherwise. (see Figure 6.2(c))

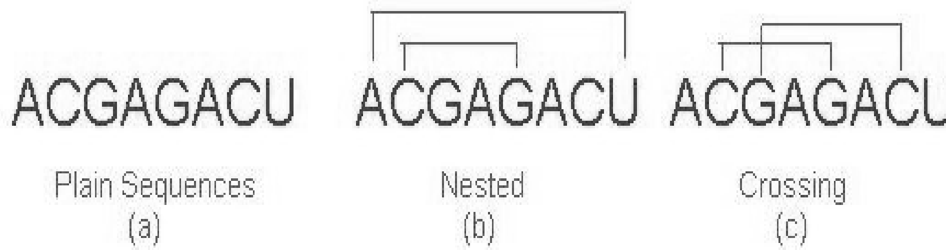

Figure 6.2: Types of arc-annotated sequence

6.2 Deduce RNA Structure By Comparing Several Plain Sequences

Sequence comparison using dynamic programming finds widespread applications in molecular biology, in detection and evaluation of similarities between pair of DNA, RNA or protein sequences. The optimal alignments obtained through this method indicate structural and functional similarities.

Dynamic programming is also the best algorithmic approach to the problem of inferring their secondary structure, given the RNA sequences.

In practice, whether or not they make use of formal algorithms, molecular biologists often carry out RNA sequence alignment and folding analysis, in conjunction, so that the partial information about what regions of two sequences are highly similar, and hence aligned, can be used to constrain the search for a common secondary structure-and vice versa-i.e. the discovery of common folding possibilities among two or more sequences suggests that the pertinent regions are aligned.

David Sankoff used this idea - the mutually informative nature of alignment and folding to solve the following problem:

Given: Two plain sequences A[1..n] and B[1..m]
To find: Common secondary structure
In the following section we discuss the Sankoff algorithm [1].

6.2.1 Sankoff Algorithm

The fundamental idea of Sankoff algorithm is to integrate alignment and folding. We find a common secondary structure for both A and B which minimizes their score, i.e., minimizes alignment cost between A and B + free energy of A + free energy of B

6.2.2 Algorithm

The algorithm finds a score for optimal alignment and free energy. Consider two sequences A[1..n] and B[1..m]. Let us define the terms:
$L(i_1, j_1; i_2, j_2)$:
This is the free energy of the loop enclosed by i_1, j_1 with i_2, j_2. $H(i, j)$:
This is the free energy of the hairpin loop. **Notations:**

1. $D(i_1, j_1; i_2, j_2)$:
   This is the minimum alignment cost between A$[i_1, j_1]$ and B$[i_2, j_2]$. We extend the definition of $D(i_1, j_1; i_2, j_2)$ as follows:
   If $i_1 > j_1$ ($A[i_1, j_1]$ is empty), then $D(i_1, j_1; i_2, j_2)$ equals the cost of inserting the entire sequnce $B[i_2, j_2]$.
   Similarly, if $i_2 > j_2$ ($B[i_2, j_2]$ is empty), then $D(i_1, j_1; i_2, j_2)$ equals to the cost of deleting $A[i_1, j_1]$.

2. $F(i_1, j_1; i_2, j_2)$:
   This is the minimum cost for a pair of equivalent secondary structures S_1 and S_2 on positions $i_1, ..., j_1$ and $i_2, ..., j_2$ of sequences A and B, where the cost is the sum of the free energies and a constrained alignment cost.

3. $C(i_1, j_1; i_2, j_2)$:
   This is the minimum cost given that $(i_1, j_1) \in S_1$and $(i_2, j_2) \in S_2$ [which means that (i_1, j_1) and(i_2, j_2) form base pairs] without considering the alignment costs of A$[i_1]$,B$[i_2]$ and A$[j_1]$,B$[j_2]$.

4. $G(i_1, j_1; i_2, j_2)$:
   This is the minimum cost between A$[i_1, j_1]$ and B$[i_2, j_2]$ given that A$[i_1...j_1]$ and B$[i_2...j_2]$ are sub-regions of a multi-loop.

The optimal score $F(i_1, j_1; i_2, j_2)$ can be computed using dynamic programming by considering the following cases:

- (i_1, j_1) and (i_2, j_2) form base pairs.
  Therefore, $F_1(i_1, j_1; i_2, j_2)$
  $= C(i_1, j_1; i_2, j_2) + D(i_1, i_1; i_2, i_2) + D(j_1, j_1; j_2, j_2)$
  = (minimum score between A$[i_1, j_1]$ and B$[i_2, j_2]$ w/o the alignment score)
  + (cost of aligning A$[i_1]$ and B$[i_2]$) + (cost of aligning A$[j_1]$ and B$[j_2]$)

- Both (i_1, j_1) and (i_2, j_2) are not base pairs.
  We subdivide the problem and Therefore,
  $F_2(i_1, j_1; i_2, j_2)$
  $= \min_{i_1 \leq h_1 \leq j_1, i_2 \leq h_2 \leq j_2} \{F(i_1, h_1; i_2, h_2) + F(h_1 + 1, j_1; h_2 + 1, j_2\}$

- There are no base pairs in these subsequences.
  Therefore, $F_3(i_1, j_1; i_2, j_2) = D(i_1, j_1; i_2, j_2)$= alignment score of A$[i_1, j_1]$ and B$[i_2, j_2]$.

Therefore, $F(i_1, j_1; i_2, j_2) = \min \{ F_1, F_2, F_3 \}$. That is,

$$F(i_1, j_1; i_2, j_2) = \min \begin{cases} C(i_1, j_1; i_2, j_2) + D(i_1, i_1; i_2, i_2) + D(j_1, j_1; j_2, j_2), \\ \min_{i_1 \leq h_1 \leq j_1, i_2 \leq h_2 \leq j_2} \begin{cases} F(i_1, h_1; i_2, h_2) + \\ F(h_1 + 1, j_1; h_2 + 1, j_2) \end{cases}, \\ D(i_1, j_1; i_2, j_2). \end{cases}$$

Time complexity In all, n^4 entries of $F(i_1, j_1; i_2, j_2)$ need to be computed. To compute one entry, we require $O(n^2)$ time - this is because for the 2nd option above, we have n^2 choices for h_1 and h_2 and we need to compute a minimum. In all, the time complexity for the above computation is $O(n^6)$

Now, let us look at how to compute $C(i_1, j_1; i_2, j_2)$.
We can compute this score using dynamic programming by considering the following cases:

- Hairpin loop
  $C(i_1, j_1; i_2, j_2) = \mathrm{H}(i_1, j_1) + \mathrm{H}(i_2, j_2) + \mathrm{D}(i_1 + 1, j_1 - 1; i_2 + 1, j_2 - 1)$
  = Loop energy for the hairpin loop formed at (i_1, j_1) in A
  + Loop energy for the hairpin loop formed at (i_2, j_2) in B
  + Alignment cost of the subsequences enclosed by these hairpin loops

- Stacking pair, internal loop, bulge
  In a stacking pair, internal loop or bulge, we look at the all possible (p_1, q_1) in A and (p_2, q_2) in B such that (p_1, q_1) and (p_2, q_2) are base-pairs enclosed by (i_1, j_1) and (i_2, j_2). Therefore,

$$C(i_1, j_1; i_2, j_2) = \min_{i_1<p_1<q_1<j_1, i_2<p_2<q_2<j_2} \begin{cases} L(i_1, j_1; p_1, q_1) + L(i_2, j_2; p_2, q_2) \\ +C(p_1, q_1; p_2, q_2) + D(i_1+1, p_1; i_2+1, p_2) \\ +D(q_1, j_1-1; q_2, j_2-1) \end{cases}$$

= loop energy of the internal loop formed by $(i_1, j_1; p_1, q_1)$ + loop energy of the internal loop formed by $(i_2, j_2; p_2, q_2)$ + minimum energy of the structure formed by $(p_1, q_1; p_2, q_2)$ + alignment cost of aligning the subsequences $A[i_1+1, p_1], B[i_2+1, p_2]$ and $A[q_1, j_1-1], B[q_2, j_2-1]$.

- Multi-loop

This cost is recursively calculated as follows:

$$C(i_1, j_1; i_2, j_2) = \min_{i_1 \le h_1 \le j_1, i_2 \le h_2 \le j_2} \begin{cases} G(i_1+1, h_1; i_2+1, h_2)+ \\ G(h_1+1, j_1-1; h_2+1, j_2-1) + 2a \end{cases}$$

Therefore,

$$C(i_1, j_1; i_2, j_2) = \min \begin{cases} H(i_1, j_1) + H(i_2, j_2) + D(i_1+1, j_1-1; i_2+1, j_2-1), \\ \min_{i_1<p_1<q_1<j_1, i_2<p_2<q_2<j_2} \begin{cases} L(i_1, j_1; p_1, q_1) + L(i_2, j_2; p_2, q_2) \\ +C(p_1, q_1; p_2, q_2) + D(i_1+1, p_1; i_2+1, p_2) \\ +D(q_1, j_1-1; q_2, j_2-1) \end{cases}, \\ \min_{i_1 \le h_1 \le j_1, i_2 \le h_2 \le j_2} \begin{cases} G(i_1+1, h_1; i_2+1, h_2)+ \\ G(h_1+1, j_1-1; h_2+1, j_2-1) + 2a \end{cases}. \end{cases}$$

Time Complexity In all, n^4 entries of $C(i_1, j_1; i_2, j_2)$ need to be computed. To compute one entry, we require $O(n^2)$ time - this is because for the 2nd option above, we actually have n^4 choices for p_1, q_1 and p_2, q_2.However, to reduce the time complexity, we constrain the choices by applying the following condition:
$p_1 - i_1 + j_1 - q_1 - 2 \le U$
$p_2 - i_2 + j_2 - q_2 - 2 \le U$
and here, U is the maximum size of internal loops, which is a small constant value. Hence we have n^2U^2 choices, i.e. the time complexity for computing a single entry is $O(n^2)$. Therefore, the overall time complexity is $O(n^6)$.

Finally let us look at how $G(i_1, j_1; i_2, j_2)$ is calculated. We have the following 3 cases:

- The multiple loops are aligned, i.e., we find the cost that calculates the folding score between the subsequences $A(i_1, j_1)$ and $B(i_2, j_2)$ and the alignment scores of $A[i_1]$ & $B[i_2]$ and $A[j_1]$ & $B[j_2]$
  $G(i_1, j_1; i_2, j_2) = C(i_1, j_1; i_2, j_2) + 2b + D(i_1, i_1; i_2, i_2) + D(j_1, j_1; j_2, j_2)$

- The multiple loops and external regions are aligned.
  $G(i_1, j_1; i_2, j_2) =$
  $\min_{i_1 \le h_1 \le j_1, i_2 \le h_2 \le j_2}(G(i_1, h_1; i_2, h_2) + (j_1 - h_1 + j_2 - h_2)c + D(h_1+1, j_1; i_2 + 1, j_2)$

$G(i_1, j_1; i_2, j_2) = \min_{i_1 \leq h_1 \leq j_1, i_2 \leq h_2 \leq j_2}((h_1 - i_1 + h_2 - i_2 + 2) + G(h_1 + 1, j_1; h_2 + 1, j_2) + D(i_1, h_1; i_2, h_2))$

- Perform recursion to combine the structurally equivalent parts.
  $G(i_1, j_1; i_2, j_2) = \min_{i_1 \leq h_1 \leq j_1, i_2 \leq h_2 \leq j_2}(G(i_1, h_1; i_2, h_2) + G(h_1 + 1, j_1; h_2 + 1, j_2))$

Therefore,

$$G(i_1, j_1; i_2, j_2) = \min \begin{cases} C(i_1, j_1; i_2, j_2) + 2b + D(i_1, i_1; i_2, i_2) + D(j_1, j_1; j_2, j_2), \\ \min_{i_1 \leq h_1 \leq j_1, i_2 \leq h_2 \leq j_2} \begin{cases} \begin{cases} G(i_1, h_1; i_2, h_2) + (j_1 - h_1 + j_2 - h_2)c \\ +D(h_1 + 1, j_1; i_2 + 1, j_2) \end{cases}, \\ G(i_1, h_1; i_2, h_2) + G(h_1 + 1, j_1; h_2 + 1, j_2), \\ \begin{cases} (h_1 - i_1 + h_2, i_2 + 2)c + \\ G(h_1 + 1, j_1; h_2 + 1, j_2) + D(i_1, h_1; i_2, h_2) \end{cases} \end{cases} \end{cases}.$$

Time Complexity In all, n^4 entries of $G(i_1, j_1; i_2, j_2)$ need to be computed. To compute one entry, we require $O(n^2)$ time - this is because for the 2nd, 3rd and 4th options above, we need have n^2 choices for h_1 and h_2 and we need to compute a minimum. In all, the time complexity for the above computation is $O(n^6)$.

In total the problem can be solved in $O(n^6)$ time.

6.2.3 Initial Condition

We assign the maximum cost if the pair of subsequences do not have a common structure. Therefore,
$C(i_1, j_1; i_2, j_2) = \infty G(i_1, j_1; i_2, j_2) = C(i_1, j_1; i_2, i_2) = \infty$

6.3 Inferring RNA Structure

We have seen, how to predict RNA secondary structure by comparing two homologous RNA plain sequences. Now, we will consider the problem of comparing RNA plain sequence S with a RNA structure T and infer the structure of S.

We are given two RNA sequences $S[1..n]$ and $T[1..m]$ and we have the nested structure "P" for S. The problem is to infer the structure Q of "T" which satisfies the conditions: (S, P)and (T, Q) has the largest weighted common substructure (defined in Section 6.3.1).

6.3.1 Score Function

If we align two RNA sequences $S[1..n]$ and $T[1..m]$, then we induce a common substructure on T. To indicate the relationship between the corresponding matching arcs or matching bases, we define two score functions as follows:

- For $u, v \in \Sigma$, let $\chi(u, v)$ be the similarity of u and v.

For example, $\chi(u,v) = 1$ if $u = v$ and 0 otherwise. For instance, $\chi(A,A) = 1$; $\chi(A,G) = 0$.)

- For $a, b \in \Sigma \times \Sigma$, let $\delta(a,b)$ be the similarity of a and b.

  For example, $\delta(a,b) = 1$ if both a and b are complementary base-pairs and 0 otherwise. (For instance, $\delta((A,U),(C,G)) = 1$.)

We have an example (see Figure 6.3) to show the score functions clearly. There are two matching arcs and one matching base between two RNA sequence("AUGCCA"and "CUUCAA").
$\chi(C,C) = 1$; $\delta((U,A),(U,A)) = 1$; $\delta((G,C),(U,A)) = 1$.

The best common substructure has score = 3. This infer a secondary structure for the second RNA sequence("CUUCAA"). Once we get the best alignment, we get the largest common substructure of the two RNA sequence.

Figure 6.3: best common substructure

6.3.2 Preliminaries

Denote $D[i,i';j,j']$ be the score of the largest weighted common substructure between $(S[i,i'], P[i,i'])$ and $(T[j,j'], Q[j,j'])$ among every possible structure Q. $D(i,i-1;j,j-1) = D(i,i';j,j-1) = D(i,i-1;j,j') = 0$ for $1 \leq i < i' \leq n$ and $1 \leq j < j' \leq m$.

We have the following definitions, which is illustrated in Figure 6.4.

- For any arc $u \in P$, u_ℓ and u_r are left and right endpoint of u. $u(i)$ is the arc in P incident on position i and $u(i)$ is ϕ if position i is free.

A position i is defined to be covered by an arc $u = (i_1, j_1) \in P$ if there is no arc $(i_2, j_2) \in P$ such that $i_1 < i_2 < i < j_2 < j_1$; and either i is free or $i = u(i)_r$.

Denote C_u as the set of positions covered by arc u. For example, in Figure 6.5, all the nodes covered by $u = (1,16)$ are of color gray. In other word, $C_u = \{2,6,11,12,14,15\}$. Note that, every node is only covered by one arc, so $\sum_u C_u \leq n$.

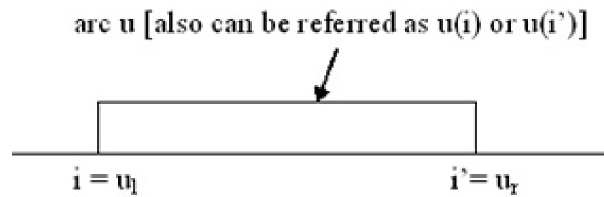

Figure 6.4: definition

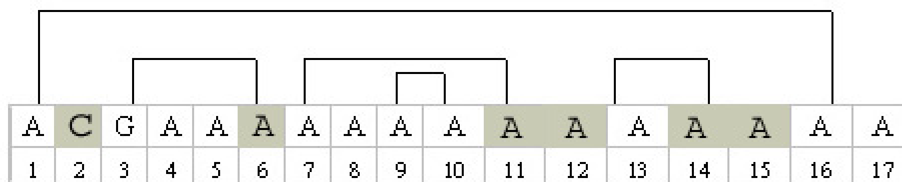

Figure 6.5: covered positions

6.3.3 Transformation

To ease the discussion, we make the following transformation:

- Insert x before and after S(i.e. $S = xSx$) where x is an extra symbol not exists in the alphabet. Similar for T, that is, set T be xTx.
- Include an arc between the two x into P.

Then we get the transformed S which contains an arc between the first and the last symbols. The inferred RNA structure for the problem are the same. We do this transformation to include the matching bases outside the outermost arc. Figure 6.6 shows the example to explain why they are the same. After we get the secondary structure of $xTx[1..m]$, we must remove the first and the last x from xTx to recover the solution.

6.3.4 Algorithm and Complexity

Algorithm: We use dynamic programming to solve this problem. The basic idea is as follows: for each arc $u = (i_1, i_2) \in P$ from inner to outer, we compute $D(i_1, i_2, j, j')$ for $1 \leq j \leq j' \leq m$ using two steps.

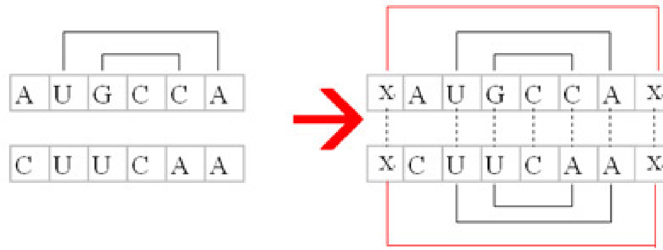

Figure 6.6: transformation

1. Step 1: Let $i = i_1 + 1$. For every $i' \in C_u$, compute $D(i, i', j, j')$ for $1 \leq j \leq j' \leq m$.
2. Step 2: Based on the values in Step 1, compute $D(i_1, i_2, j, j')$ for $1 \leq j \leq j' \leq m$

Finally, $D(i_1, i_2, 1, m)$, where (i_1, i_2) is the outermost arc of S, stores the length of the longest common substructure. The actual structure can be found through tracing back the matrix D.
Step 1: Let $i = i_1 + 1$. For every $i' \in C_u$, in increasing order, computer $D(i, i'; j, j')$ for $i \leq j \leq j' \leq m$.

If i' is free, we have the following recursive equation.

$$D(i, i'; j, j') = \max \begin{cases} D(i, i' - 1; j, j' - 1) + \chi(S[i'], T[j']), & \text{T[j'] is not a arc base} \\ D(i, i' - 1; j, j'), & \text{S[i] maps to a space} \\ D(i, i'; j, j' - 1). & \text{T[j'] maps to a space} \end{cases}$$

- D(i, i'-1;j, j'-1)+χ(S[i'], T[j']):When both S[i'] and T[j'] are unpaired base, ie. i' and j' are both free, we have the score function to check if S[i']=T[j'].
- D(i, i'-1;j, j'): S[i'] maps to a space.
- D(i, i';j, j'-1): T[j'] maps to a space.

If i'= $u(i')_r$, we partition the problem into two subproblems. Figure 6.7 shows how we subdivide the problem.
$D(i, i'; j, j') = \max_{j \leq j'' \leq j'+1} \{ D(i, u(i')_\ell - 1; j, j'' - 1) + D(u(i')_\ell, i'; j'', j')$

- $\max\{D(i, u(i')_\ell - 1; j, j'' - 1) + D(u(i')_\ell, i'; j'', j')\}$: When $i' = u(i')_r$, we divide $S[i..i']$ into two parts, which is $S[i, u(i')_\ell - 1]$ and $S[u(i')_\ell..i']$. Either $S[u(i')_\ell..u(i')_r]$ does not mapped to anything or it mapped to a partial secondary structure generated from interval (j'', j') for some j''.

Step2: Based on the values in Step 1, we compute $D(i_1, i_2, j, j')$ for $1 \leq j \leq j' \leq m$. We can generate the largest common substructure between S and T

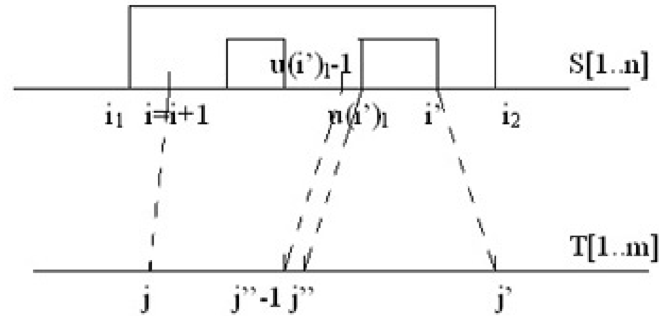

Figure 6.7: subdivision of Step1

through backtracking. The base pairs of T in the common substructure form the secondary structure of T.

$$D(i_1, i_2; j, j') = \max \begin{cases} D(i_1+1, i_2-1; j+1, j'-1) + \delta((S[i_1], S[i_2]), (T[j], T[j'])), \\ \text{if}(S[i_1], S[i_2]) \text{ and } (T[j], T[j']) \text{ are complementary} \\ D(i_1+1, i_2-1; j, j'), \\ D(i_1, i_2; j, j'-1), \\ D(i_1, i_2; j+1, j'). \end{cases}$$

- $D(i_1+1, i_2-1; j+1, j'-1) + \delta((S[i_1], S[i_2]), (T[j], T[j']))$: In this case, both $(S[i_1], S[i_2])$ and $(T[j], T[j'])$ are two complementary base-pairs.
- $D(i_1+1, i_2-1; j, j')$: We drop the arc here.
- $D(i_1, i_2; j, j'-1)$: $T[j']$ maps to a space.
- $D(i_1, i_2; j+1, j')$: $T[j]$ maps a space.

We find the maximum among these cases.

Time Complexity: We analyze the time complexities of two steps separately.

Step1: For every arc $u \in P_1$, we need to fill in $|C_u|m^2$ entries. Each entry can be computed in $O(m)$ time. For all arcs, the total number of entries is $\sum_u(|C_u|m^2) = nm^2$. So the total time for Step 1 is $O(nm^3)$.

Step2: We need to fill in nm^2 entries. Each entry takes $O(1)$ time. So total time for Step2 $= O(nm^2)$.

6.4 Comparing two RNAs

Similar to DNA sequence comparison problems, we can use edit operations to compare RNAs.Three edit operations can be performed, which are defined as following:

- Relabel: relabel an unpaired base or a base-pair
- Delete: delete an unpaired base or a base-pair
- Insert: insert an unpaired base or a base-pair

An edit operation is represented as $x \to y$, where x and y are either a base-pair or an unpaired base. Λ means an empty position. For example:

- Relabel: $(A, U) \to (G, C)$, $A \to C$
- Delete: $(A, U) \to \Lambda$, $A \to \Lambda$
- Insert: $\Lambda \to (C, G)$, $\Lambda \to A$

These operations can be operated on base pair or unpaired base. When operating to a pair, in sequence level, this means the two bases changes at the same time. When inserting a pair, we require that the elements in S are non-crossing after the insertion.

6.4.1 Edit Distance

We can transform U_1 to U_2 using a sequence S of edit operations $s_1, s_2, \ldots, s_k$. We define γ(x$\to$y) be the cost of an edit operation, then the cost of a sequence S of edit operations γ(S)=$\Sigma_{i=1..n}(s_i)$. The edit distance between U_1 and U_2 is the minimum cost of edit operation sequence that transforms U_1 to U_2. Table 6.1 shows the time complexity that needed to compare plain, nested and crossing sequences.

- When both U_1 and U_2 are plain sequences, ordinary sequence comparison can be done in $O(nm)$ time;
- When U_1 is plain and U_2 is nested or crossing, we have to delete all the arcs in U_2 and convert it to a plain sequence, then we compare the transformed U_2 with U_1 using a ordinary sequence comparison. This can be done in $O(nm)$ time;
- When both U_1 and U_2 are nested, they can be represented as an ordered tree, which will be discussed in the next section.
- When both U_1 and U_2 are crossing, it becomes a NP-complete problem.

Below is the table that shows the time complexities for different possible scenarios in RNA comparison problem.

	Plain	Nested	Crossing
Plain	O(mn)		
Nested		$O(m^2 n \log n)$ OR $O(mn\ colldepth(U_1) colldepth(U_2))$	
Crossing			NP-complete

Table 6.1 Transforming Time Complexity

6.4.2 Ordered Tree

Trees are among the most common and well-studied combinatorial structures [10]. In computational biology, computing the similarity between trees under various distance measures is used in the comparison of RNA secondary structures [7, 6]. In this section, we will show how an ordered tree is used for RNA structures comparison.
Let T be a ordered tree. Ordered tree is a rooted tree in which every node is labelled with a number called the order. The order of the nodes in an ordered tree is significant. The definition of the nodes' order is recursive: for all nodes in an ordered tree, first label its children from left to right, and then label the root of the node. Then the root always has the largest order.

In this section, the tree edit distance problem is based on simple primitive operations applied to ordered tree. We can define the tree edit operations similar to the sequence edit operation as following:

- Relabel $(x \to y)$: change the label of a node in T from x to y
- Delete $(x \to \Lambda)$: Delete a non-root node x in T, making the children of the node x become the children of the parent of x and then removing x.
- Insert $(\Lambda \to x)$: complement of delete, inserting a node x as a child of node y, and making x being the parent of the original children of node y

Tree edit distance We use $\gamma(x \to y)$ to denote the cost of the tree edit operations. Ordered tree edit distance problem is defined as follows: given two ordered trees T_1 and T_2, we can transform T_1 to T_2 by a sequence of tree edit operations. The ordered tree edit distance between T_1 and T_2 is the minimum cost of tree edit operation sequence that transforms T_1 to T_2.

Figure 6.8 shows an example of ordered tree transformation. First one node is relabelled from C to A, and then a node labelled with "AU" is deleted. If the cost of each operation is 1, the edit distance between T_1 and T_2 is 2.

6.4.3 Nested Sequence VS Ordered Tree

Ordered Tree is a powerful tool for structure comparison. Nested sequence can be modelled as an Ordered Tree as follows.

- Each base-pair of a nested sequence is represented by a node; and unpaired base is also represented by a node
- Node y is a descendent of node x in Ordered Tree if the region represented by y is a sub-region of the region represented by x in nested sequence. The order of the nodes should preserve the order of the corresponding regions.

Examples of nested sequence and its corresponding ordered tree are shown in Figure 6.8.

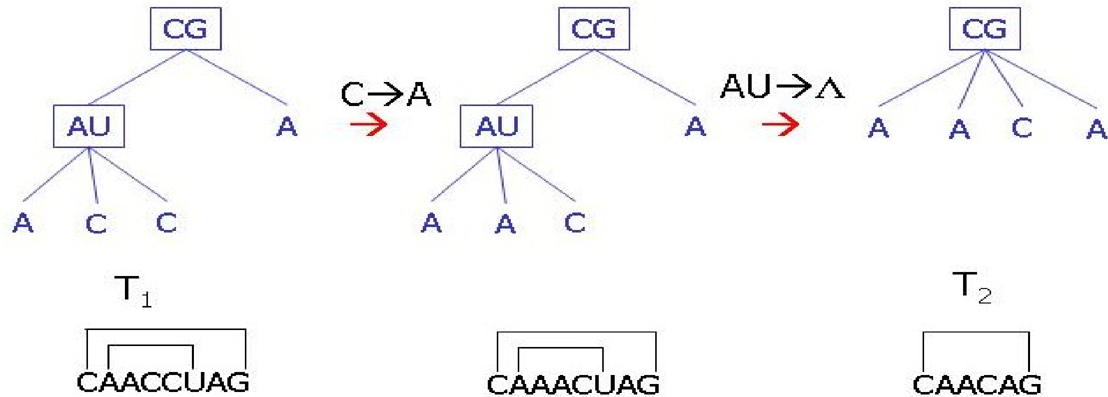

Figure 6.8: Transformation of Nested Sequences VS Ordered Tree

Examining each of the operations defined on the nested sequences and ordered trees in Figure 6.8, we can observe that there is an one-to-one correspondence between the tree edit operations defined on the ordered tree and the sequence edit operations defined on the sequence. Therefore, we get the following lemma from the observation.

Lemma 6.1 *The edit distance between the two nested sequences = the edit distance between the corresponding ordered trees*

6.5 Algorithm for Ordered Tree Edit Distance

This section describes one common-used algorithm for ordered tree edit distance problem. Suppose there are two trees T_1 and T_2.

Let's define $n = |T_1|, m = |T_2|$, assuming $n < m$, and define $L_i = l[T_i]$ {l is the number of leaves}, $D_i = m[T_i]$ {m is the depth} .

The ordered tree edit distance problem was introduced by Tai [8] as a generalization of the well-known string edit distance problem [9]. Tai presented an

algorithm for the ordered tree using $O(nmL_1^2L_2^2)$ time and space. Subsequently, Zhang and Shasha [7] improved Tai result by presenting an algorithm which uses $O(nmcolldepth(T_1)colldepth(T_2))$ time, where $colldepth(T) = min\{depth(T), leaves(T)\}$ and $O(nm)$ space. This algorithm was modified by Klein [Kle98] to get a better worst case time bound of $O(n^2mlogm)$ under the same space bounds. Recently, Chen [Che01] has presented an algorithm using $O(nm + L_1^2m + L_1^{2.5}L_2)$ time and $O((n + L_1^2)min(L_2; D_2) + m)$ space. This algorithm uses results on fast matrix multiplication and is more complex than all of the above algorithms.

This lecture gives a simple algorithm first, and then Zhang and Shasha's algorithm is introduced to improve the time complexity.

6.5.1 Some definitions

- **Postorder numbering:** For any tree T, T[i] is referred to be the i-th node according to the left-to-right postorder numbering. Figure 6.9 shows an example, where the tree is being numbered from left sub-tree, right sub-tree and finally the root.

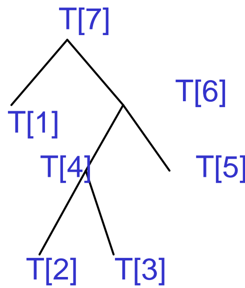

Figure 6.9: Postorder Numbering

- **Ordered Forest:** An ordered forest is a set of sub-trees with order from left to right. For any $i < j$, $T[i..j]$ defines an ordered forest. Note that $T[i]$ is the leftmost leaf of the leftmost tree in the forest $T[1..j]$ while $T[j]$ is the root of the rightmost tree in the forest $T[1..j]$. Figure 6.10 shows some examples.

- $l(i)$**:** For any i, denote $l(i)$ be the leftmost leaf of the subtree rooted at i.

  Note that $T[l(i)..i]$ represents the subtree rooted at i. For example, $l(6) = 2$ for the tree in Figure 6.9. Figure 6.10(b) shows the subtree $T[l(6)..6]$

T[1] T[4] T[5]

T[2] T[3]

(a) T[1..5]

T[6]

T[4] T[5]

T[2] T[3]

(b) T[2..6]

Figure 6.10: Ordered Forest for the tree in Figure 6.9

6.5.2 A simple algorithm

Let $fdist(i, i'; j, j')$ be the edit distance between $T_1[i..i']$ and $T_2[j..j']$. The edit distance between T_1 and T_2 is $fdist(1, n; 1, m)$ where n = $|T_1|$ and m = $|T_2|$. The following two lemmas show the base case and the recursive case for $fdist(i, i'; j, j')$.

Lemma 6.2

- $fdist(i, i'; j, j') = 0$ *if* $i > i'$ *and* $j > j'$
- $fdist(i, i'; j, j') = fdist(i, i' - 1; j, j') + \gamma(T_1[i'], \Lambda)$ *for* $i \leq i', j > j'$
- $fdist(i, i'; j, j') = fdist(i, i'; j, j' - 1) + \gamma(\Lambda, T_2[j'])$ *for* $i > i', j \leq j'$

Proof: $T_1[i..i']$ and $T_2[j..j']$ are both ordered forests, which implies that they follow the postorder numbering. This means $i \leq i'$ and $j \leq j'$. Therefore for Case 1, if $i > i'$ and $j > j'$, this means the trees are empty. Then the algorithm does not apply and the cost equals 0. For Cases 2 and 3, there is one empty tree in each case. It will recursively delete another tree to an empty tree respectively. ■

Lemma 6.3 *For* $i \leq i'$, $j \leq j'$,

$$fdist(i, i'; j, j') = \min \begin{cases} fdist(i, i' - 1; j, j') + \gamma(T_1[i'], \Lambda) \text{ Delete } T_1[i'] \\ fdist(i, i'; j, j' - 1) + \gamma(\Lambda, T_2[j']) \text{ Insert } T_2[j'] \\ fdist(i, l(i') - 1; j, l(j') - 1) + fdist(l(i'), i' - 1; l(j'), j' - 1) + \\ \gamma(T_1[i'], T_2[j']) \text{ Otherwise, replace } T_1[i'] \text{ by } T_2[j'] \end{cases}$$

Proof: The objective is to find out the edit distance between $T_1[i..i']$ and $T_2[j..j']$ with the minimum cost, therefore the algorithm select the minimum cost from the above 3 cases. For Cases 1 and 2, the cost will be a delete operation and insert operation respectively between $T_1[i..i']$ and $T_2[j..j']$, and the formula are straightforward. If we don't delete $T_1[i']$ or insert $T_2[j']$, then we must relabel $T_1[i']$ to $T_2[j']$, which is Case 3. Case 3 requires operation for both forests T_1 and

T_2. Aside from relabelling one node in each tree, we need to perform separate lookups for the forests to the left of T_1[i'] and T_2[j'] and for the descendants of T_1[i'] and T_2[j']. This gives the third recursive equation. Figure 6.11 below illustrates the operation for Case 3.

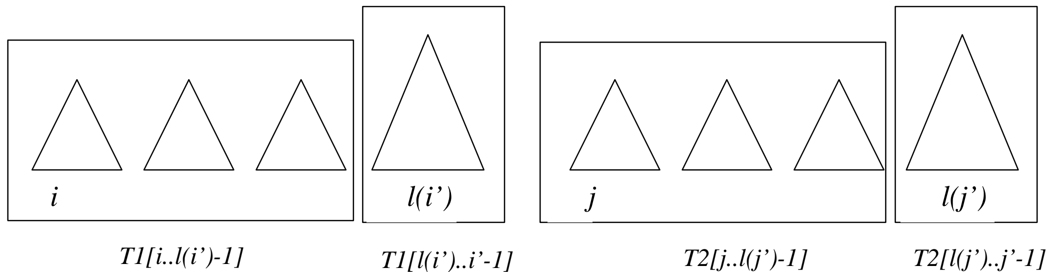

Figure 6.11: Recursive algorithm illustration

■

Time complexity Since $n = |T_1|$ and $m = |T_2|$, table *fdist* has n^2m^2 entries. Each entry can be computed in $O(1)$ time. Therefore the total time is $O(n^2m^2)$.

6.5.3 Zhang and Shasha's algorithm: Improving the time complexity

Lemma 6.2 and 6.3 define a dynamic program to solve the ordered tree edit distance problem with $O(n^2m^2)$ time. In fact, the table *fdist* is sparse and we do not need to fill all the entries in the table. Based on this observation, Zhang and Shasha developed an algorithm [7] to improve the time complexity.

Let's denote $anc(i)$ be the set of ancestors of i. We find that it is unnecessary to compute $fdist(i, i'; j, j')$ for all $i, i' \in T_1$ and $j, j' \in T_2$. Instead, we only need to compute $fdist(l(i'), i; l(j'), j)$ for all $i \in T_1, j \in T_2, i' \in anc(i)$, and $j' \in anc(j)$.

Note that for every nodes $i \in T_1$ and $j \in T_2$, we have $|anc(i)| \leq depth(T_1)$ and $|anc(j)| \leq depth(T_2)$. Thus, we only need to compute $O(nm\ depth(T_1)depth(T_2))$ $fdist$ values!

Denote $tdist(i, j)$ be the edit distance between subtrees of T_1 and T_2 rooted at i and j respectively. Also note that $tdist(i, j) = fdist(l(i), i; l(j), j)$ The aim of the algorithm is to compute $tdist(n, m)$. The following two lemmas show the recursive equations to compute $fdist(l(i'), i; l(j'), j)$ for $i' \in anc(i), j' \in anc(j)$.

When $i = i'$ and $j = j'$, we have the following lemma.

Lemma 6.4

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

Proof: Based on Lemma 6.3, we have

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

For the last case, the term $fdist(l(i),l(i)-1;l(j),l(j)-1)$ is 0 because $l(i)-1 < l(i)$ and $l(j)-1 < l(j)$. Thus, the lemma follows. ■

When $i \neq i'$ or $j \neq j'$, we have the following lemma.

Lemma 6.5

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

Proof: The following two inequalities are correct, since the former distance corresponds to the cost of a minimum cost mapping while the latter expression stands for a particular and possibly suboptimal mapping between T_1 and T_2.

$$fdist(l(i'),i;l(j'),j) \leq fdist(l(i'),i-1;l(j'),j-1) + tdist(i,j) \quad (6.1)$$
$$tdist(i,j) \leq fdist(l(i),i-1;l(j),j-1) + \gamma(T_1[i],T_2[j]). \quad (6.2)$$

Based on Lemma 6.3, we have

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

According to Inequality (6.1), we introduce Case 4 to the recursive equation.

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

Based on Inequality (6.2), Case 4 of the recursion equation is always smaller than Case 3 of the recursion equation. The recursion equation can be simplified by removing Case 3. The lemma follows.

■

Time complexity For every $i \in T_1, j \in T_2$, for $i' \in anc(i), j' \in anc(j)$, $fdist(l(i'),i;l(j'),j)$ can be computed in $O(1)$ time. And for every node in an ordered tree, it has at most $depth(T)$ ancestors obviously. Then we only need to

compute $O(nm\ depth(T_1)depth(T_2))$ fdist-values, the time complexity is $O(nm\ depth(T_1)depth(T_2))$. The analysis here is simplified. Actually, the time complexity should be $O(nm\ colldepth(T_1)colldepth(T_2))$. The details can be found in [7]. This algorithm has a very high time complexity in the worst case, however, it runs fast and takes about $O(nm)$ time in practice, since generally $colldepth(T)$ is quite small. Although some algorithms have a lower time complexity in the worse case theoretically, such as those in [Kle98] and [Che01], they run slower in practice, or more complex techniques are required. This makes Zhang and Shasha's algorithm common-used in practice.

References

[1] DAVID SANKOFF, "Simultaneous solution of the RNA Folding, Alignment and Protosequence Problmes", *SIAM Journal Appl. Math*, 45(5), 810-825, 1985.

[2] JASON T. L. WANG, BRUCE A. SHPIRO, DENNIS SHASHA, KAIZHONG ZHANG and KATHLEEN M. CURREY, "An Algorithm for Finding the Largest Approximately Common Substructures of Two Trees", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.20, No.8, August 1998.

[3] KAIZHONG ZHANG, LUSHENG WANG, BIN MA, "'Computing Similarity between RNA Structures"', *CPM 1999: 281-293.*

[4] DAVID SANKOFF, *"The early introduction of dynamic programming into computational biology", Bioinformatics, Vol. 16, No.1, 2000, pp. 41-47.*

[5] CARSTEN ISERT, *"The Editing Distance Between Trees".*

[6] TAO JIANG, LUSHENG WANG, KAIZHONG ZHANG, *"Alignment of Trees - An Alternative to Tree Edit", TCS, 143(1): 137-148 (1995)*

[7] KAIZHONG ZHANG, DENNIS SHASHA, *"Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems", SIAM J. Comput, 18(6): 1245-1262 (1989)*

[8] KUO-CHUNG TAI, *"The tree-to-tree correction problem", Journal of the Association for Computing Machinery (JACM),26:422-433, 1979.*

[9] ROBERT A. WAGNER AND MICHAEL J. FISCHER, *"The string-to-string correction problem", Journal of the ACM (JACM), 21:168-173, 1974.*

[10] PHILIP BILLE, *"Tree Edit Distance, Alignment Distance and Inclusion", University Technical Report Series TR-2003-23, ISSN 1600-6100, March 2003.*

[11] GUO-HUI LIN, BIN MA, KAIZHONG ZHANG, *"'Edit distance between two RNA structures"', RECOMB, 2001.*

[12] P.A. EVANS, *"'Algorithms and Complexity for Annotated Sequence Analysis"', PhD thesis, University of Victoria, 1999.*

[13] P.A. EVANS, *"'Finding common subsequences with arcs and pseudoknots"'. In Proceedings of 10th Annual Symposium on Combinatorial Pattern Matching (CPM99), LNCS 1645, pages 270280, 1999.*

[14] P. N. KLEIN, *"'Computing the edit-distance between unrooted ordered trees"', Algorithms -ESA 98, 6th Annual European Symposium Gianfranco Bilardi, Giuseppe F. Italiano,Andrea Pietracaprina, and Geppino Pucci, Eds. Lecture Notes in Computer Science, 1461, 91-102, Springer-Verlag, Berlin New York, 1998.*

[15] WEIMIN CHEN, *"'New Algorithm for Ordered Tree-to-Tree Correction Problem"'. J. Algorithms 40(2), 135-158, 2001.*

[16] KAIZHONG ZHANG, *"'Computing Similarity between RNA Secondary Structures"', IEEE International Joint Stmposia on Intelligence and Systems, 126-132, 1998.*

[17] ZHUOZHI WANG, KAIZHONG ZHANG, *"'Alignment between Two RNA Structures"', MFCS 690-702, 2001.*

[18] V. BAFNA, S. MUTHUKRISHNAN, R. RAVI, *"'Computing Similarity between RNA Strings"'. CPM 1-16, 1995.*

[19] TAO JIANG, GUO-HUI LIN, BIN MA, KAIZHONG ZHANG, *"'The Longest Common Subsequence Problem for Arc-Annotated Sequences"', CPM 154-165, 2000.*

[20] GUO-HUI LIN, ZHI-ZHONG CHEN, TAO JIANG, JIANJUN WEN, *"'The Longest Common Subsequence Problem for Sequences with Nested Arc Annotations"', ICALP, 444-455, 2001.*