

9.1 Introduction

9.1.1 Phylogenetic Tree

Phylogeny, or an evolutionary tree, is a model of the evolutionary history for a set of species. In the previous two lectures, we have learned a number of methods to construct phylogeny for the same set of species, which is one of fundamental tasks of computational molecular biology. This is because the evolutionary relationship of species provides a great deal of information about their function.

9.1.2 Phylogenetic Tree Comparison

However, in practice, different phylogenetic trees are reconstructed in different situation. Below discuss 3 main reasons:

- Different Kind of Data
The data used to represent the same species for tree reconstruction is not always the same, which will definitely lead to different result. For example, different segments of the genomes are used as the input to the tree reconstruction algorithm.
- Different Kind of Model
Every phylogenetic tree reconstruction algorithm is based on some models (or assumptions), such as: Cavender-Felsenstein Model, Jukes-Cantor Model etc. There is no consensus which model is the best.
- Different Kind of Algorithm
We use different kind of reconstruction algorithm for a set of species. It is not surprising that various algorithms do not always give the same answer on the same input.

Surely, the resulting trees may agree in some parts and differ on others. Tree comparison helps us to gain the similarity and dissimilarity information from multiple trees.

9.1.3 Tree Comparison Type

In this lecture, we will discuss two types of comparison and the corresponding techniques.

- **Similarity Measurement:** The similarity measurement determines the common structure among the given trees. One popular similarity measurement is Maximum Agreement Subtree (MAST). Since information extracted is agreed among the trees, the information is trustable to a certain extent.
- **Dissimilarity Measurement:** The dissimilarity measurement determines the difference/distance among the given trees. There are three popular dissimilarity measurement: Robinson-Founds distance, nearest neighbor interchange (NNI) and subtree transfer distance (STT).

9.2 Maximum Agreement Subtree

9.2.1 Restricted Subtree

A *restricted subtree* of a tree is its subtree formed by restricting the leaves of the subtree to a subset of the leaves of the tree. To derive a restricted subtree from a tree, remove all the leaves that are not restricted on and remove all internal nodes with a single child. Figure 9.1 illustrates this process.

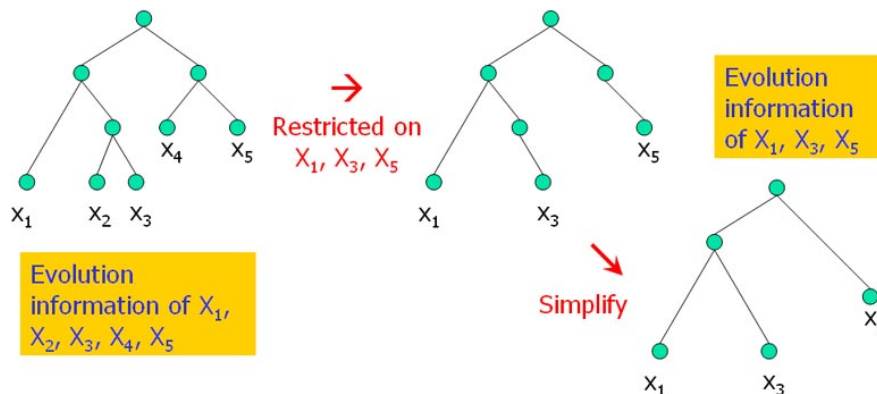


Figure 9.1: Formation of a Restricted Subtree

9.2.2 Maximum Agreement Subtree

Definition 9.1 Given two trees T_1 and T_2 , an agreement subtree of T_1 and T_2 is a common restricted subtree derived from both trees. A maximum agreement subtree (MAST) of two trees is the agreement subtree of the two trees with the largest possible number of leaves.[1].

Since an agreement subtree reflects common information agreed by both trees, the evolution information represented by the agreement subtree is more reliable. Refer to Figure 9.2 for an illustration.

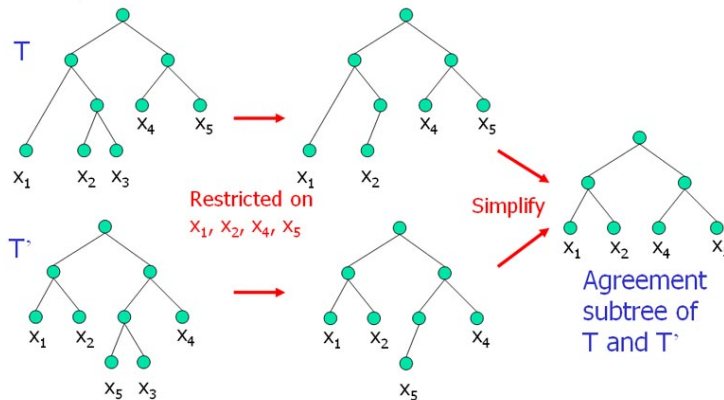


Figure 9.2: Agreement Subtree

9.2.3 Computing MAST by Dynamic Programming

Here, we give an algorithm to compute the MAST of two binary rooted trees using dynamic programming:

Denote the number of leaves in the MAST of two binary rooted trees T_1 and T_2 to be $MAST(T_1, T_2)$. For a tree T and a node u , define T^u to be the subtree of T rooted at u .

Using dynamic programming, the MAST of T_1 and T_2 is computed using the following recurrence formula:

$$MAST(T_1^u, T_2^v) = \max \begin{cases} MAST(T_1^a, T_2^c) + MAST(T_1^b, T_2^d) \\ MAST(T_1^a, T_2^d) + MAST(T_1^b, T_2^c) \\ MAST(T_1^a, T_2^v) \\ MAST(T_1^b, T_2^u) \\ MAST(T_1^u, T_2^c) \\ MAST(T_1^u, T_2^d) \end{cases}$$

Figure 9.3 illustrates the details.

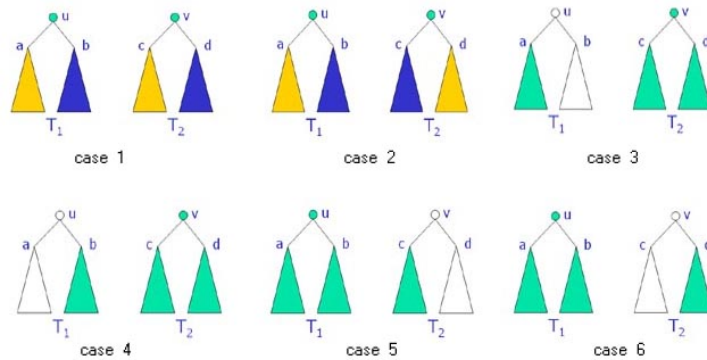


Figure 9.3: Six Cases for MAST

- $MAST(T_1^a, T_2^c) + MAST(T_1^b, T_2^d)$: The species appear in both subtrees of T_1 and T_2 , and we try to match the left subtree of T_1 with the left subtree of T_2 , match the right subtree of T_1 with the right subtree of T_2 . In this case, u and v match.
- $MAST(T_1^a, T_2^d) + MAST(T_1^b, T_2^c)$: The species appear in both subtrees of T_1 and T_2 , and we try to match the left subtree of T_1 with the right subtree of T_2 , match the right subtree of T_1 with the left subtree of T_2 . In this case, u and v match.
- $MAST(T_1^a, T_2^v)$: All the species appears in the left subtree of T_1 , and we try to match left subtree of T_1 with the tree T_2 .
- $MAST(T_1^b, T_2^u)$: All the species appears in the right subtree of T_1 , and we try to match right subtree of T_1 with the tree T_2 .

- $MAST(T_1^u, T_2^e)$: All the species appears in the left subtree of T_2 , and we try to match left subtree of T_2 with the tree T_1 .
- $MAST(T_1^u, T_2^d)$: All the species appears in the right subtree of T_2 , and we try to match right subtree of T_2 with the tree T_1 .

Now we analyze the time complexity of the algorithm. Suppose T_1 and T_2 are rooted phylogenies for n species. We have to compute $MAST(T_1^u, T_2^v)$ for every u in T_1 and v in T_2 . Thus, we need to fill in n^2 entries. Each entry can be computed in $O(1)$ time. In total, the time complexity is $O(n^2)$.

In fact, we can compute $MAST$ of two degree- d rooted trees T_1 and T_2 with n leaves in $O(\sqrt{dn} \log(\frac{n}{d}))$ time (Journal of Algorithm 2001). We will not discuss this algorithm here.

9.2.4 MAST for Unrooted Trees

In real life, we normally want to compute $MAST$ for unrooted trees. While the set of evolutionary trees may be large in practice, the trees usually have very small degrees, typically no larger than three. For unrooted degree-3 trees U_1 and U_2 , $MAST(U_1, U_2)$ can be computed in $O(n \log n)$ time (STOC 97) [4]. For general unrooted trees U_1 and U_2 , $MAST(U_1, U_2)$ can be computed in $O(n^{1.5} \log n)$ time (SIAM J. of Comp 2000) [1]. We will not cover algorithms for unrooted trees here. However, we will discuss the $MAST$ relationship between unrooted trees and rooted trees.

For any unrooted tree U , for any edge e in U , we denote U^e is as the rooted tree rooted at the edge e . Figure 9.4 illustrates an example for converting an unrooted tree U to a specified rooted tree U^e .

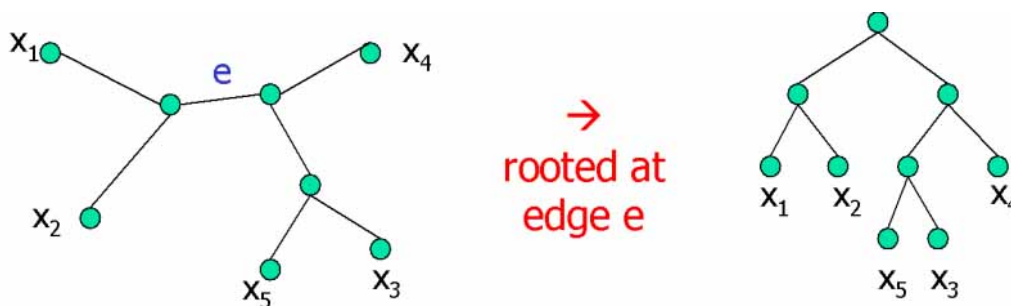


Figure 9.4: Converting an Unrooted Tree U to a Rooted Tree U^e at edge e

Lemma 9.2 For any edge e of U_1 , $MAST(U_1, U_2) = \max\{MAST(U_1^e, U_2^f) \mid f \text{ is the an edge of } U_2\}$.

Proof: The proof of this lemma is left as an exercise. ■

The above lemma gives the relation from unrooted trees to rooted trees. Using this lemma with dynamic programming algorithm for finding MAST of two rooted trees, we can compute the MAST of two unrooted trees.

9.3 Robinson-Foulds Distance

In previous section, we covered how to compute the similarity of two phylogenetic trees, including both unrooted and rooted trees. Now we discuss how to compute the dissimilarity of two phylogenetic trees. One intuitive way to define the dissimilarity is based on the number of edges that are not agreed by two trees, which is called the Robinson-Foulds distance [5]. The Robinson-Foulds distance metric d defines the distance, $d(T_1, T_2)$ between any two trees T_1 and T_2 as the smallest number of transformations required to obtain the topology of T_2 from the topology of T_1 . The metric d has the following properties:

1. $d(T_1, T_2) > 0$ if T_1 is not identical to T_2
2. $d(T_1, T_2) = 0$ if T_1 is identical to T_2
3. $d(T_1, T_2) = d(T_2, T_1)$
4. $d(T_1, T_3) \leq d(T_1, T_2) + d(T_2, T_3)$

There are several operations that can be performed on phylogenetic trees, in the following section, we shall describe their definitions in detail.

9.3.1 Definitions

- **Partition of a tree:** Each edge of a tree can partition the species into two subsets. This definition tells us that if we take away an edge from the phylogenetic tree, the species in the tree is partitioned into two disjoint set of species. Figure 9.5 shows an example.
- **Good and bad edge:** Consider two unrooted trees T and T' , an edge x in T is called a good edge if there exists an edge x' in T' such that both of them form the same partitions. Similarly, x' is also called a good edge. Otherwise, the edge x is called a bad edge. Figure 9.6 illustrates an example.

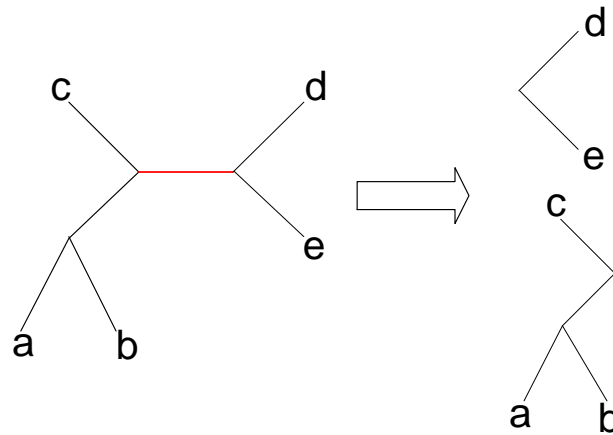


Figure 9.5: The red edge partitions the species into two sets $\{a, b, c\}$ and $\{d, e\}$.

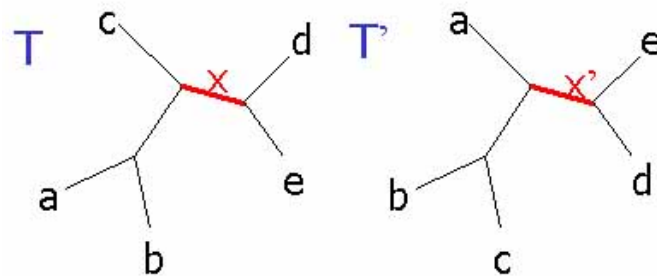


Figure 9.6: x partitions the species of T into $\{a, b, c\}$ and $\{d, e\}$. x' also partitions the species of T' into $\{a, b, c\}$ and $\{d, e\}$. Thus, x and x' form the same partition and we call x and x' the good edges.

- **Lemma 9.3** *Leaf edges are always good.*

Proof: As a leaf edge always divides a tree into two parts, one is a single leaf node and the other is a set of rest nodes. Then in the other tree, there must also be another edge which makes the same partition. An example is given in Figure 9.7. ■

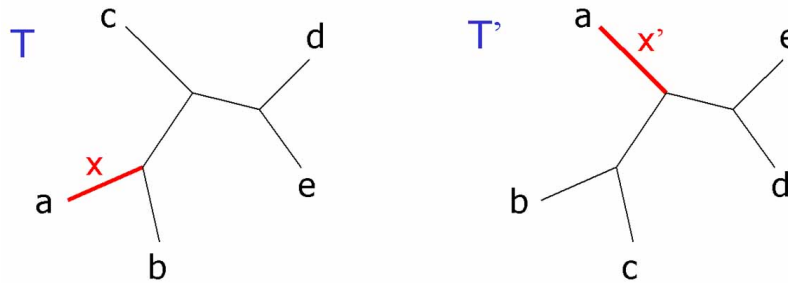


Figure 9.7: x in T and x' in T' are also good edges

9.3.2 Robinson-Foulds (RF) Distance

Robinson-Foulds Distance is defined as follows.

Given two phylogeny trees T and T' , the RF distance between these two trees is: RF distance = (number of bad edges in T with respect to T' + number of bad edges in T' with respect to T)/2.

From this definition, we can have the criteria for measuring the similarity of two phylogeny trees. We say the two trees are similar when RF distance between T and T' is small. This is because the RF distance is based on the number of bad edges. If this distance is small, it implies that the majority of edges between each other is good edge, which means the two trees are similar. In order to have a clear knowledge of this definition, Figure 9.8 gives an example: In this example, there is only 1 bad edge in each phylogeny tree. According to the definition, the RF distance is $(1 + 1)/2 = 1$. Since this value is small, it implies that these two trees are similar.

When the phylogenies are of degree 3, we have the following property for RF distance.

Lemma 9.4 *When both T and T' are of degree-3, the number of bad edges in T with respect to $T' =$ the number of bad edges in T' with regard to T .*

Proof: As both trees are of degree 3, we know that they have the same number of edges. Also we know that the number of good edges in T with respect to T' is equal to the number of good edges in T' with respect to T .

Since the number of bad edge is equal to the number of total edges minus the number of good edges, the lemma follows. ■

Here we present a brute-force algorithm to compute the RF distance.

For every edge e in T , if the partition formed by e is the same as the partition formed by some edge e' in T' , e is a good edge.

This is a general algorithm for finding the set of good edges. We just do an exhaustive search for all the edges of T and T' . After finishing the search, a set of good edges in T with respect to T' is found.

For the time analysis, Note that for every edge e in T , the algorithm checks every edge e' in T' to determine whether e and e' form the same partition of species. Such checking takes $O(n)$ time. Since there are n edges in T , the algorithm takes $O(n^2)$ time.

As a phylogeny tree can be quite huge, such time complexity is far beyond acceptable. Then we need some faster algorithm to solve this problem.

9.4 Day's algorithm

As described in the previous section, the brute-force algorithm to find the set of good edges takes $O(n^2)$ time. In 1985, W. H. E. Day proposed an algorithm which can find the set of good edges in T with respect to T' with using $O(n)$ time[6].

9.4.1 Problem description

- Input: Two unrooted phylogenies T_1 and T_2 for the same set of species.
- Output: The set of good edges in T_1 with respect to T_2 .

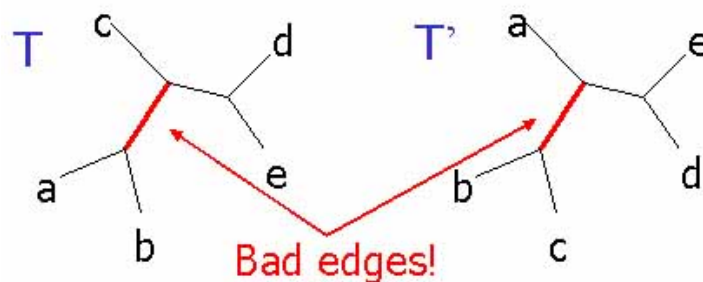


Figure 9.8: The computation of RF distance between T and T'

The idea of Day's algorithm is to build a special data structure which enables constant time checking whether a particular partition of leaves exists in T_1 .

9.4.2 Day's algorithm

- **Step 1:** Root T_1, T_2 at the leaf species with number n
Figure 9.9 shows the two trees T_1 and T_2 , and Figure 9.10 shows the procedure to label trees T_1 and T_2 . Time complexity for this step: $O(n)$.

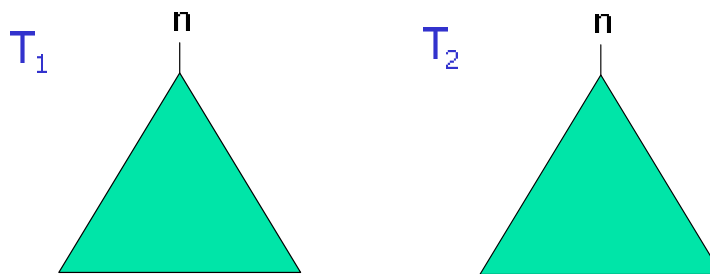


Figure 9.9: Step 1: Two degree-3 T_1 and T_2

- **Step 2:** Relabel the leaves of the trees
Relabel the leaves of T_1 in increasing order. And change the labels of T_2 correspondingly. Then for every internal node x of T_1 , the set of leaf labels in the subtree of x form an interval $[i..j]$. Figures 9.11 and 9.12 show the relabelled trees T_1 and T_2 . Time complexity for this step is $O(n)$.
- **Step 3:** Create a hash table to store the intervals
For every node x in T_1 , we store the corresponding interval $[i_x..j_x]$ in $H[i_x]$ or $H[j_x]$
 - Store $[i_x..j_x]$ in $H[j_x]$ if x is the leftmost child of its parent in T_1 ;
 - Otherwise, store the interval $[i_x..j_x]$ in the entry $H[i_x]$.

Figure 9.13 shows the hash table for the given example. It takes $O(n)$ to construct this hash table.

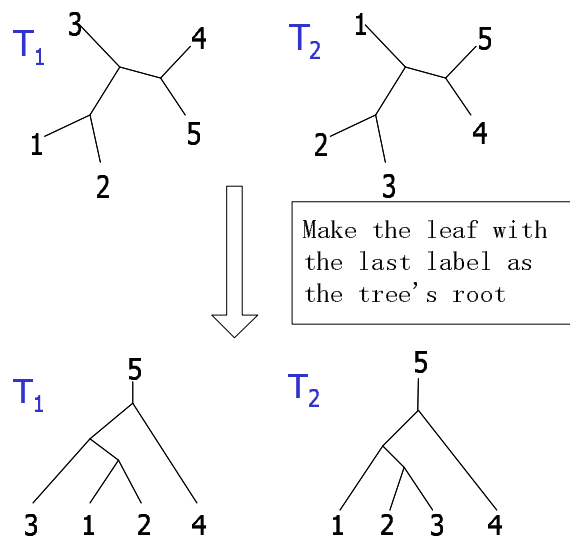


Figure 9.10: Example for Step 1: Label all leaves of T_1 and T_2

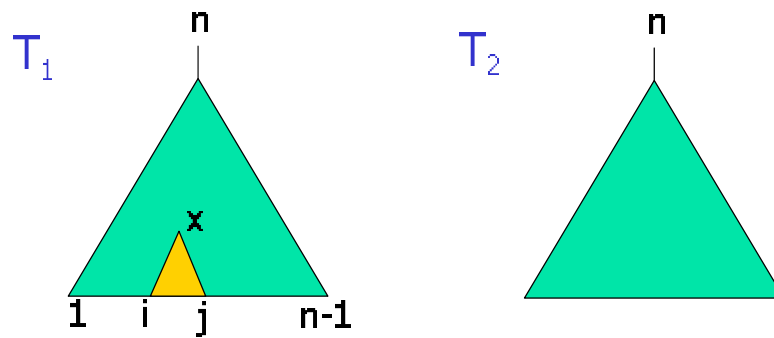


Figure 9.11: Step 2: Relabel all leaves of T_1 in increasing order.

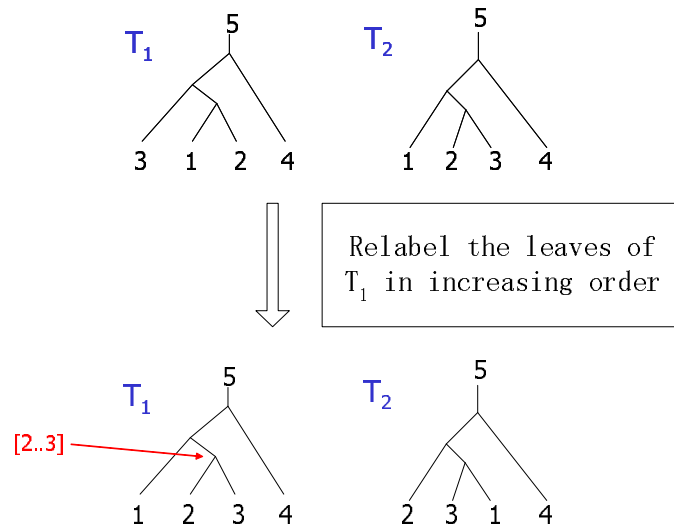


Figure 9.12: Example for Step 2: Relabel all leaves of T_1 in increasing order.

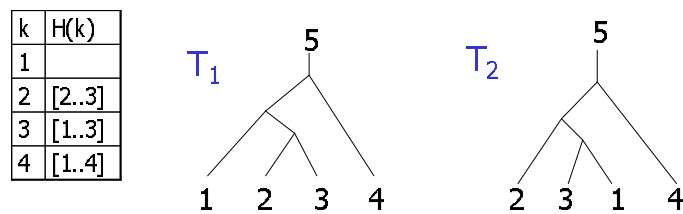


Figure 9.13: Example for Step 3: Hash table to store the intervals for the internal nodes in T_1 .

The problem is that whether we will store two intervals in the same entry $H[i]$ in the hash table H . If we use the method stated above, below lemma implies that collision is impossible.

Lemma 9.5 *If we store the intervals according to the rules stated above, we will store at most one interval in each entry in H .*

Proof:

- By contrary, suppose $H[i]$ contains two intervals which are represented by internal nodes x and y .
- By definition, i should be the endpoints of the intervals represented by x and y . Thus, x and y should satisfy the ancestor-descendent relationship. WLOG, assume x is the ancestor of y . Then, y 's interval should be the subinterval of x 's interval.
- So, we can have either
 - 1 . x 's interval = $[j..i]$ and y 's interval = $[j'..i]$ for $j < j'$; or
 - * This means that both x and y are the leftmost children of their parents.
 - * The right endpoint of x 's interval should not be i !
 - * So we get a contradiction!
 - 2 . x 's interval = $[i..j]$ and y 's interval = $[i..j']$ for $j > j'$. Similar to the above case, we can arrive at a contradiction.
- In conclusion, we can store at most one interval in each entry in H .

■

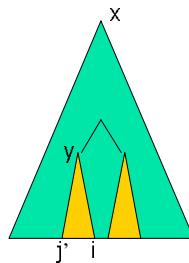


Figure 9.14: Example for Step 2: Relabel all leaves of T_1 in increasing order.

Given the hash table H , we can check whether an interval $[i..j]$ exists in T_1 by checking $H[i]$ and $H[j]$. If one of them equals to $[i..j]$, then the interval $[i..j]$ does exist in T_1 .

- **Step 4:** Traverse tree T_2 to find the good edges
 - For T_2 , by traversing the tree, for each internal node u in T_2 , we can find (1)the minimum(min_u) and the maximum(max_u) leaf labels, and (2)the number of leaves($size_u$) in the subtree rooted at u .
 - If($max_u - min_u + 1 = size_u$), then we can know that the leaves labels in the subtree of node u form an interval $[min_u..max_u]$. then we check whether $H[min_u]$ or $H[max_u]$ equals $[min_u..max_u]$. If yes, (u, v) is a good edge where v is the parent of u in T_2 .

Figure 9.15 shows the results of computing the min_u , max_u , and $size_u$. Time complexity for this step is $O(n)$.

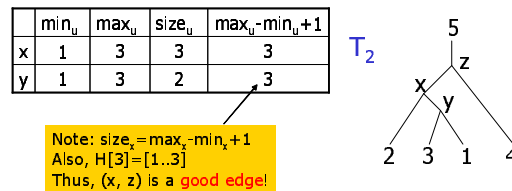


Figure 9.15: Example for Step 4: Compute the minimum, the maximum leaf labels, and the number of leaves.

Time complexity analysis: All 4 steps can correctly recover the good edges, and they can be computed in $O(n)$ time. So the total time complexity is $O(n)$.

9.5 Nearest Neighbor Interchange Distance (NNI)

Given two unrooted binary trees T_1 and T_2 , the distance $NNI(T_1, T_2)$ between those trees is the smallest number of nearest neighbor interchanges (NNI) required to transform one tree into another [13].

Definition 9.6 Below defines the NNI operation and the NNI distance.

- **NNI Operation:** A Nearest Neighbor Interchange (NNI) is the interchanging of two of the subtrees incident to an internal edge (=branch) in a binary tree. Two such interchanges are possible for each internal edge. Figure 9.16 gives an example of the two possible NNI operations across an edge. Nodes x and y are adjacent to edge e , nodes a and b are incident to node x , and nodes c and d are incident to node y . Interchanging nodes b and c results in the top tree and interchanging nodes b and d results in the bottom tree.

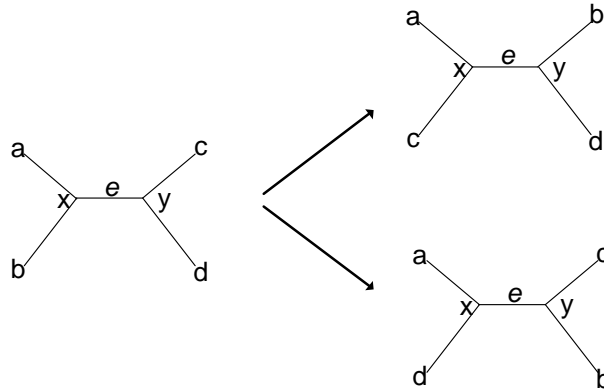


Figure 9.16: NNI Operation Example

- **NNI Distance:** Given two unrooted, degree-3 tree T_1 and T_2 , the NNI Distance is defined as the minimum number of NNI operations required to convert T_1 to T_2 , which is notated as $NNI\text{-}dist(T_1, T_2)$. Figure 9.17 shows an example.

Below shows some properties of NNI-dist.

Lemma 9.7 $NNI\text{-}dist(T_1, T_2) = NNI\text{-}dist(T_2, T_1)$

Proof: Since the nearest neighbor interchange (NNI) distance measures the minimum number of NNIs required to change T_1 into T_2 . If one NNI is required to convert T_1 to T_2 , then obviously one NNI is required to convert T_2 to T_1 , hence, the lemma follows. ■

Lemma 9.8 $NNI\text{-}dist(T_1, T_2) \geq \text{number of bad edges in } T_1 \text{ with respect to } T_2$

Proof: To remove one bad edge, we required at least one NNI-operation. ■

Computing NNI-dist is NP-hard. However, there exists a polynomial time $O(\log n)$ -approximation algorithm. This approximation algorithm makes use of certain merge-sort technique, which will not be discussed in this lecture.

9.6 Subtree Transfer Distance (STT)

Definition 9.9 Below defines the STT operation and the STT distance.

- **STT Operation:** Given an unrooted, degree-3 tree T , a subtree transfer operation is the operation of detaching a subtree and reattaching it to the middle of another edge. There are a few different methods to charge the STT operation. In this lecture, the STT operation is charged by the number of nodes the subtree is transferred. Figure 9.18 gives an example of STT operation.
- **STT Distance:** Given two unrooted, degree-3 tree T_1 and T_2 , the STT Distance is defined as the minimum cost of STT operations required to transfer T_1 to T_2 , which is notated as $STT\text{-dist}(T_1, T_2)$.

Below we show a property of STT distance.

Lemma 9.10 $STT\text{-dist}(T_1, T_2) \leq NNI\text{-dist}(T_1, T_2)$.

Proof:

- $STT\text{-dist}(T_1, T_2) \leq NNI\text{-dist}(T_1, T_2)$ because each NNI operation is an STT operation.
- $STT\text{-dist}(T_1, T_2) \leq NNI\text{-dist}(T_1, T_2)$ because each STT operation of cost k can be simulated by k NNI-operations.

■

Based on the previous property of SST-dist, the SST distance equals NNI distance. Therefore, computing $STT\text{-dist}(T_1, T_2)$ is also a NP-hard problem. However, there also exists a polynomial time $O(\log n)$ approximation algorithm to compute $STT\text{-dist}(T_1, T_2)$.

As we can see from the definition of STT distance, the STT approach of phylogenetic tree comparison mainly depends on the cost mechanism of STT operation. According to our current cost mechanism, we have the property that STT distance equals to NNI distance. If we use different cost mechanism, this property may not hold. This is also the reason that we need both NNI and STT approach.

9.7 References

- [1] M.Y. KAO, T.W. LAM, W.K. SUNG, H.F. TING, "Cavity matchings, label compressions, and unrooted evolutionary trees", *SIAM Journal of Computing*, 30(2):602-624, 2000.

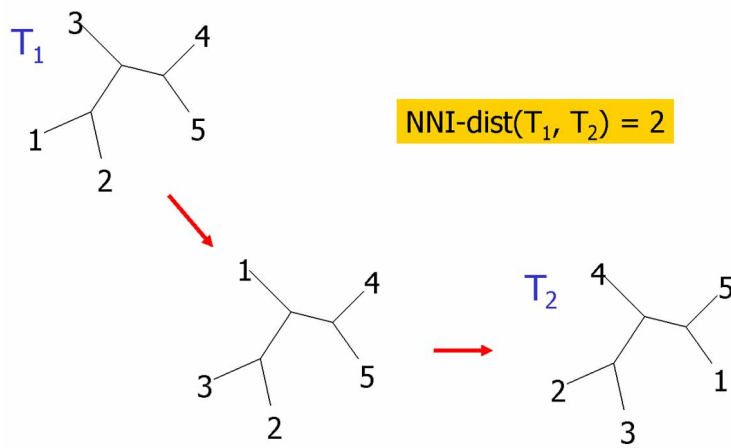


Figure 9.17: The NNI Distance of T_1 and T_2 is 2.

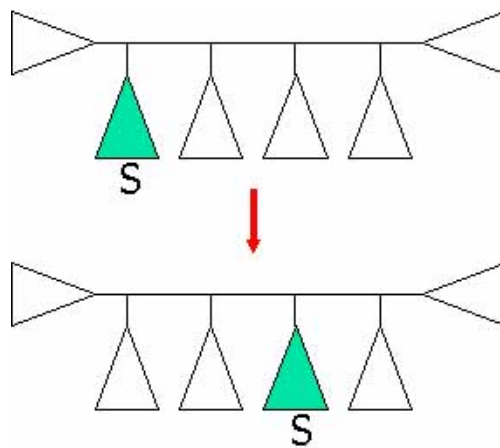


Figure 9.18: The cost of the above STT Operation is 2

- [2] C. R. FINDEN, A. D. GORDON, "Obtaining common pruned trees.", *Journal fo Classification*, 2:255-276, 1985.
- [3] M. Y. KAO, T. W. LAM, W. K. SUNG, H. F. TING, "Fast Labeled Tree Comparisons via Unbalanced Bipartite Matchings.", *Journal of Algorithms*, 40(2):212-233, 2001.
- [4] M. Y. KAO, T. W. LAM, W. K. SUNG, T. M. PRZYTYCKA, H. F. TING, "General techniques for comparing unrooted evolutionary trees.", *In Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC 1997)*, pages 54-65, 1997.
- [5] D. F. ROBINSON, L. R. FOULDS, "Comparison of phylogenetic trees.", *Mathematical Biosciences*, 53:131-147, 1981.
- [6] W. H. E. DAY, "Optimal algorithms for comparing trees with labeled leaves.", *Journal of Classification*, 2:7-28, 1985.
- [7] M. LI, J. TROMP, L. X. ZHANG, "Some notes on the nearest neighbour interchange distance.", *Journal of Theoretical Biology*, 182:463-467, 1996.
- [8] B. DASGUPTA, X. HE, T. JIANG, M. LI, J. TROMP, "On the linear-cost subtree-transfer distance between phylogenetic trees.", *Algorithmica*, 25(2):176-195, 1999.
- [9] B. DAS GUPTA, X. HE, T. JIANG, M. LI, J. TROMP, L. ZHANG, "On distance between phylogenetic trees.", *In Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 427-436, 1997.
- [10] J. HEIN, "Reconstructing evolution of sequences subject to recombination using parsimony.", *Mathematical Biosciences*, 98:185-200, 1990.
- [11] J. HEIN, "A heuristic method to reconstruct the history of sequences subject to recombination.", *Journal of Molecular Evolution*, 36:396-405, 1993.
- [12] G. W. MOORE, M. GOODMAN, J. BARNABAS, "An iterative approach from teh standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets.", *Journal of Theoretical Biology*, 38:423-457, 1973.

- [13] D. F. ROBINSON, "Comparison of labeled trees with valency three.", *Journal of Combinatorial Theory*, 11:105-119, 1971.