Combinatorial Methods in Bioinformatics

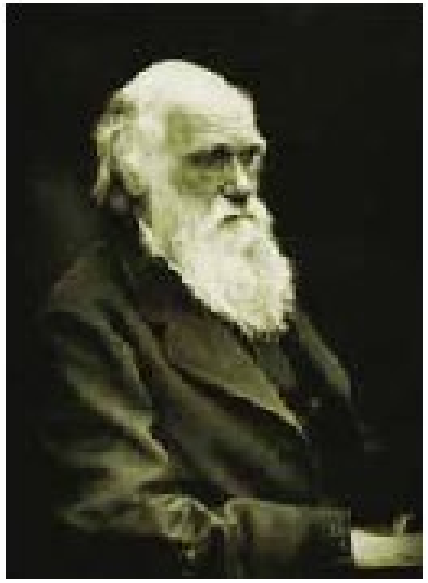
Phylogenetic Trees Reconstruction
Character-Based Methods

Wing-Kin Sung, Ken 宋永健
ksung@comp.nus.edu.sg

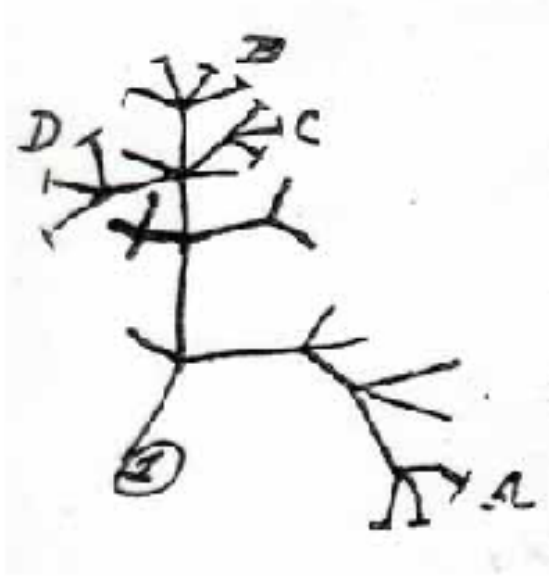
Evolution

- DNA encodes the information of life.
- Living things pass the DNA information to their children.
- Due to mutation, the DNA is changed by a little bit.
- After a long time, different species evolved.
- Phylogenetics studies the genetic relationship among different species!

Definition of Phylogeny



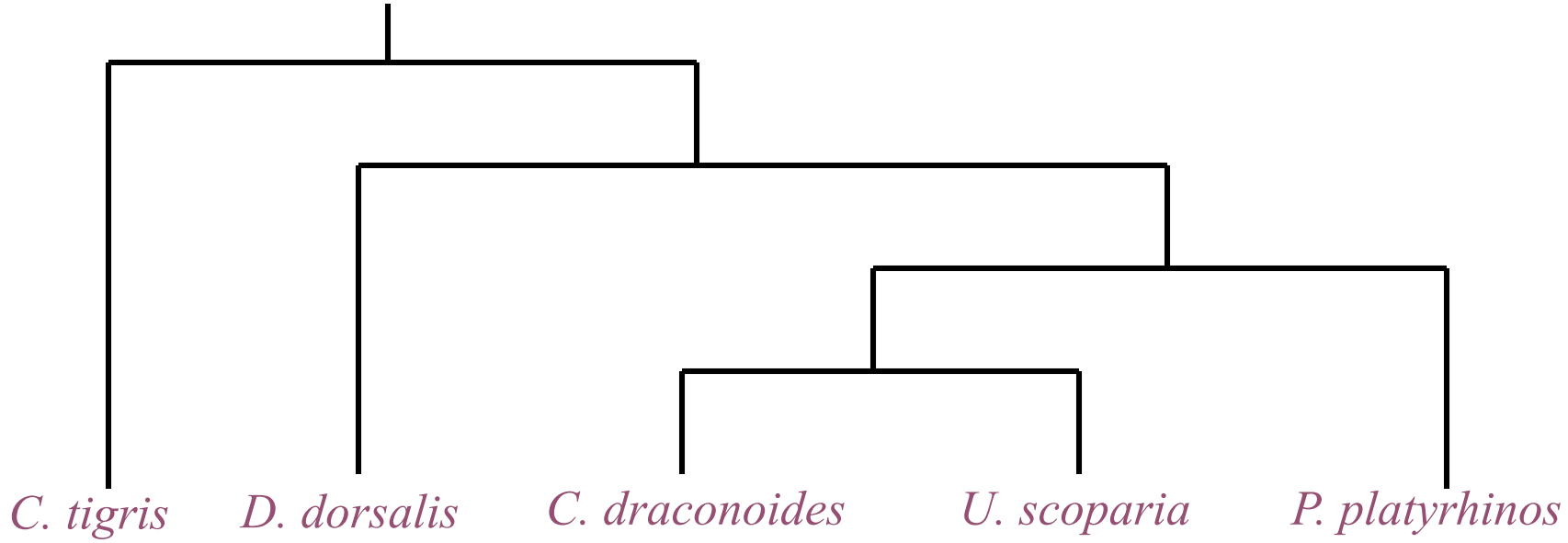
- Phylogeny (or Phylogenetic tree): Reconstruction of the evolutionary history of a set of species.
- Usually, it is a leaf-labeled tree where the internal nodes refer the hypothetical ancestors and the leaves are labeled by the species
- The edges of the tree represent the evolutionary relationships



First Notebook on Transmutation of Species, 1837.

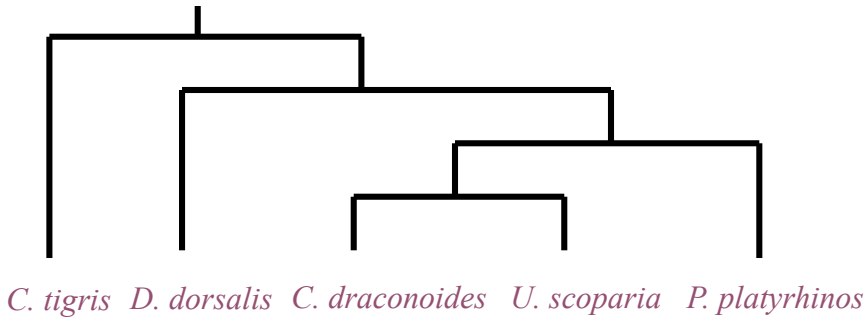
Example of phylogeny

- Phylogeny for lizards

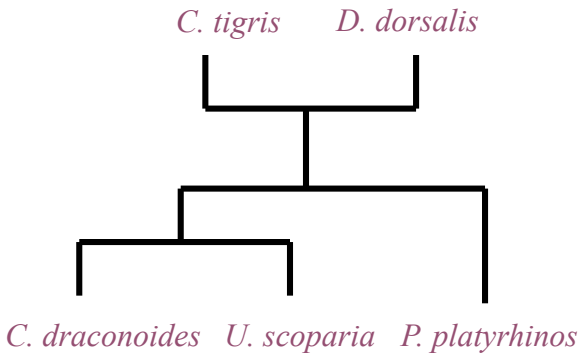


Rooted and Unrooted Tree

- A phylogeny is rooted.
- However, since estimating the root is scientifically difficult, the reconstructed tree may be unrooted.



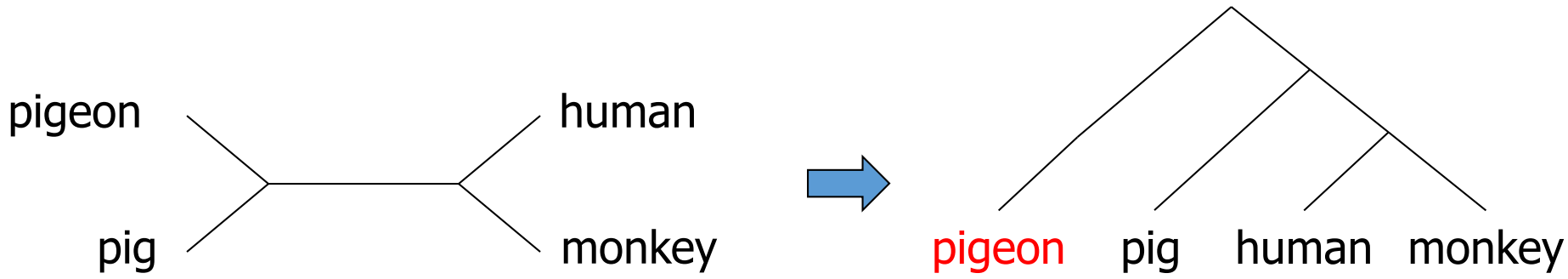
Rooted



Unrooted

Rooted a phylogeny by outgroup

- Rooted tree can be reconstructed by systematic biologists based on using outgroup.
  - Outgroup is a species which is clearly less related with all other species in the phylogeny
  - E.g. build the phylogenetic tree for pigeon, pig, human and monkey. Then, most probably, pigeon is the outgroup since it is the only non-mammal.



Human evolution

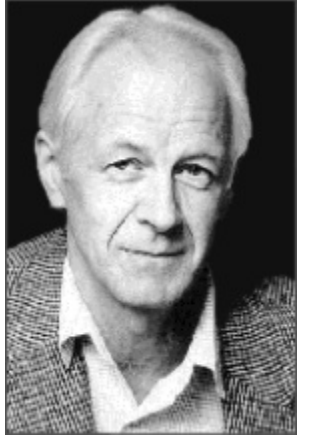
- As an example, we can understand the human evolution through phylogenetic study.
- Below, we illustrate the phylogenetic study of
  - mitochondrial Eve
  - Y chromosome Adam

About mitochondrial Eve

- Human mitochondrial DNA (mtDNA)
  - Circular double-stranded consisting of 16,500 base pairs
  - Everyone inherits the mtDNA from his/her mother (because mitochondria exists in egg, not in sperm)
  - The pointwise mutation substitution rates of mtDNA is roughly 10 times faster than nuclear DNA
  - Every cell has many mtDNAs.
  - Apparently lack of recombination.

- Therefore, we all inherit the mtDNA from the mother of human (Eve)!

Genetics helps finding the origin of human



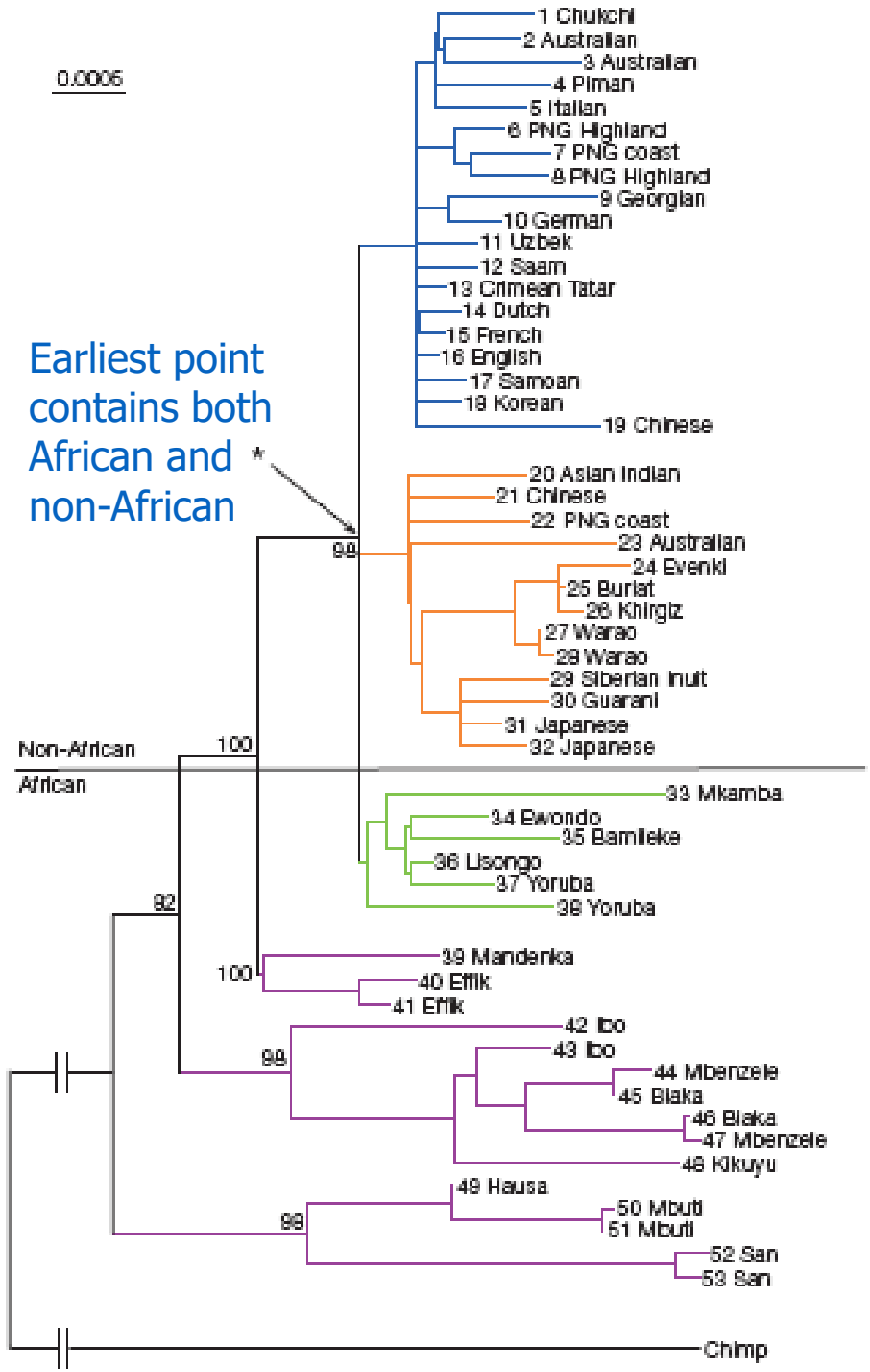
- By carrying out a statistical analysis of mtDNAs extracted from the placental tissue of 147 women of different races and from different countries
  - Alan Wilson's group and others construct a phylogenetic tree under the assumption of a constant molecular clock.
  - Such phylogenetic tree implies that the common ancestor of modern human appear roughly 100,000-200,000 years ago. (about 143,000 years ago)

Vigilant, L., Stoneking, M., Harpending, H., Hawkes, K. &Wilson, A. C. African populations and the evolution of human mitochondrial DNA. Science 253, 1503-1507 (1991).
Cann, R. L., Stoneking, M. & Wilson, A. C. Mitochondrial DNA and human evolution. Nature 325, 31-36 (1987).

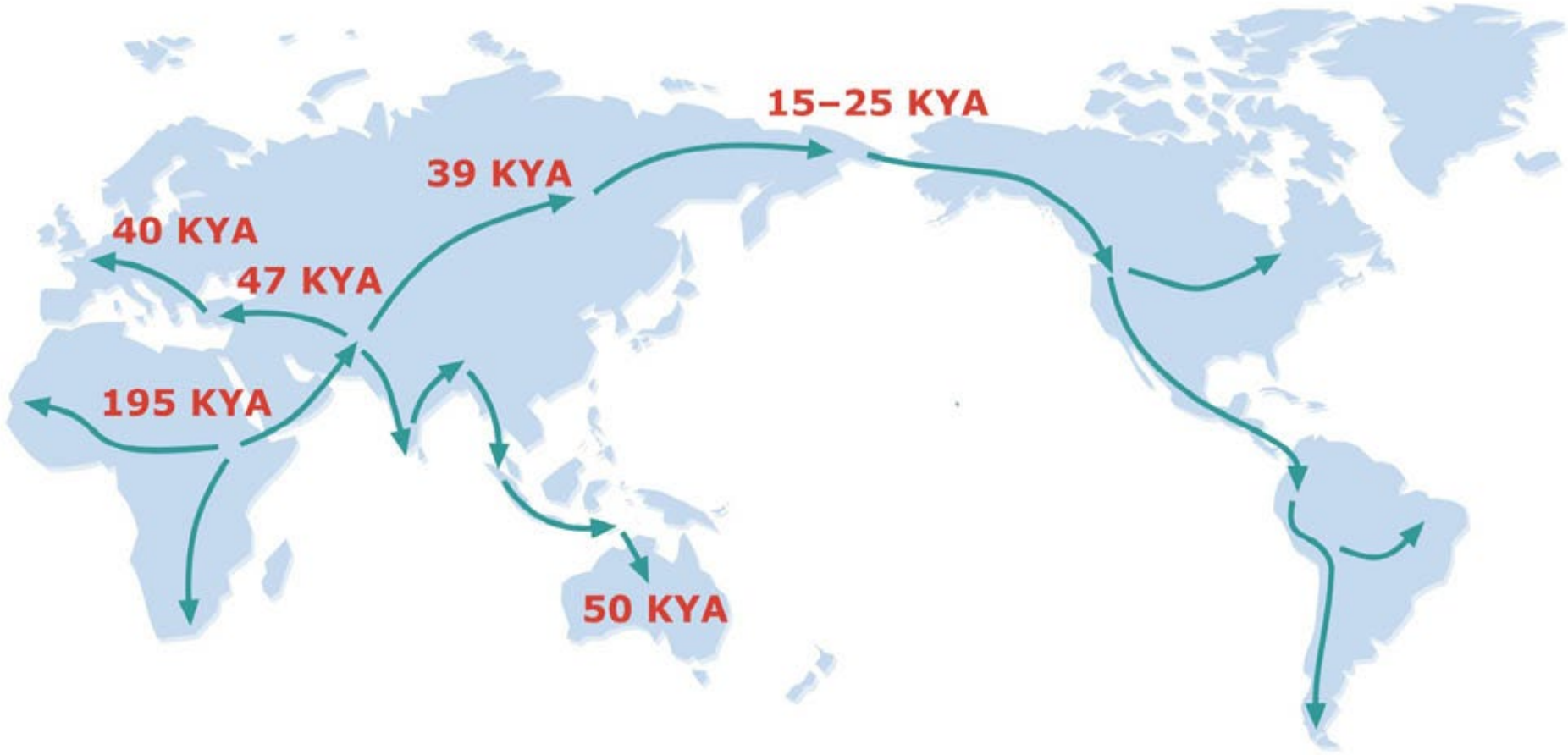
Eve tree

- Tree constructed using neighbour-joining for 53 humans and 1 chimp.
  - chimp is outgroup!
- Complete mtDNA excluding the D-loop.

- M. Ingman, H. Kaessmann, S. Paabo, and U. Gyllensten. Mitochondrial genome variation and the origin of modern humans. Nature, 2000.



Moving out of Africa



About Y chromosome Adam (I)

- Y chromosome is unique to males and it can help to find the father of human.
- However, since the mutation rate of Y chromosome is not as fast as mtDNA,
  - we need more samples to study the evolution of Y chromosome

About Y chromosome Adam (II)

- In Science 1997, at least 93 polymorphic sites have been identified in Y chromosomes of 900 men scanned.
- For one of the site,
  - 15% Khoisan people have A
  - 5-10% of Ethiopians and Sudanese have A
  - Most africans and people outside Africa have T
- This suggested that
  - Khoisan, Ethiopians, and Sudanese (in Africa) may be the closest living relatives to the Y chromosome Adam

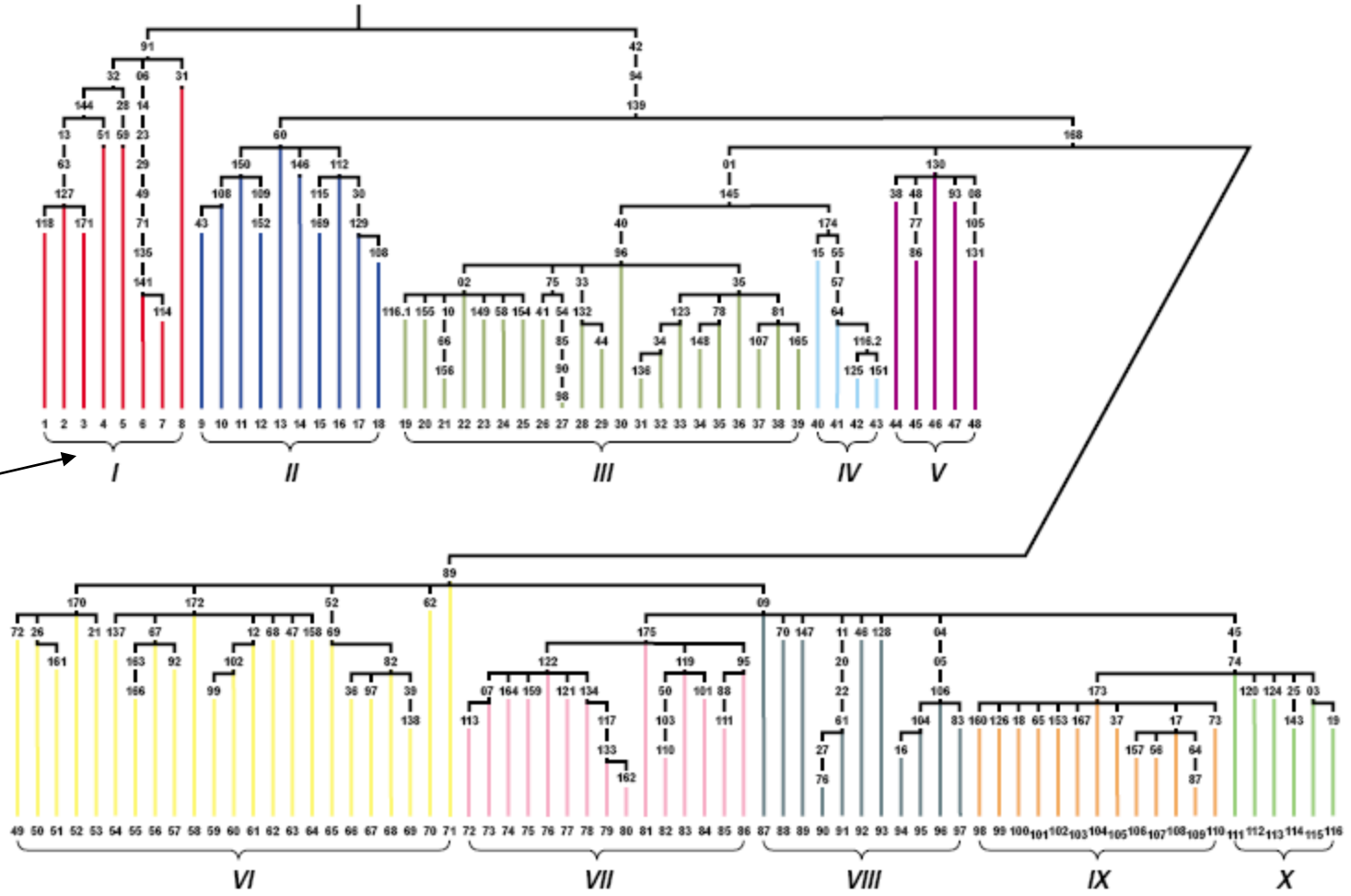
About Y chromosome Adam (III)

- In Nature genetic 2000, by studying Y chromosome of 1062 males from 22 different geographic areas,
  - They identify 167 haplotypes.
  - The common ancestor of the 167 haplotypes is estimated to appear around 59,000 years old.

- Underhill et al. Y chromosome sequence variation and the history of human populations. Nature Genetic, 26:358-361, 2000.

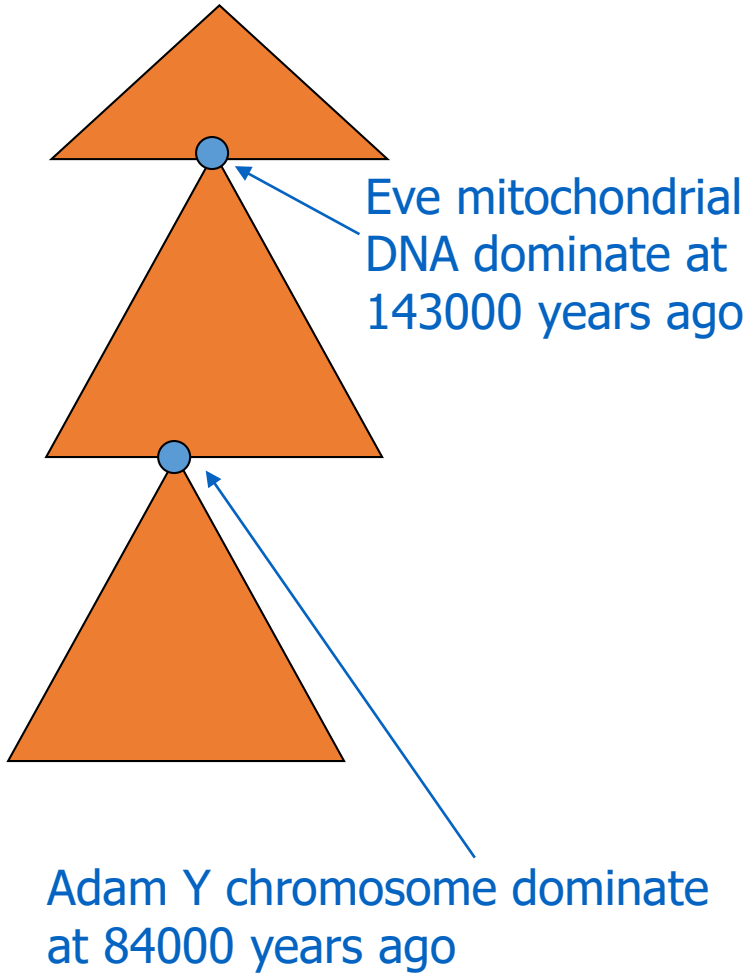
Adam tree

Minority of Africans—mainly Sudanese, Ethiopians and Khoisans



Explanation why Adam and Eve appear in different time

- In around 143,000 years ago,
  - Among different mitochondrial DNA sequences in human population, the Eve mitochondrial DNA had advantages and started to dominate.
  - All other versions of mitochondrial DNA eventually disappear.
- In parallel, different versions of Y chromosomes appear in human population.
  - It took another 84,000 years before the Adam Y chromosome started to take over in the human population.



Applications of Phylogeny

- Apart from understanding the history of life, there are many other applications
  - Understanding rapidly mutating viruses (like HIV)
  - Help to predict protein/RNA structure
  - Help to do multiple sequence alignment
  - Explaining and predicting gene expression
  - Explaining and predicting ligands
  - Help to design enhanced organisms (like rice, wheat)
  - Help to design drug

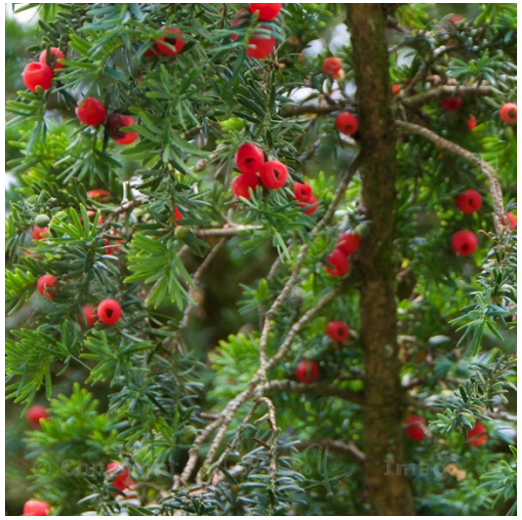
Drug discovery example

- Scientist discovered Pacific Yew produces a compound called taxol that is helpful to treat certain kind of cancer (a kind of chemotherapy drugs).

- However, taxol is extracted from the bark of the Pacific Yew.
  - Once taxol is extracted, the tree would die. This makes taxol production difficult since Pacific Yew is a slow growing tree.

- Through evolutionary relationships among yew species, they find European Yew is closely related with Pacific Yew.

- They discovered that the leaves of European Yew contain a related compound that can be efficiently produced. Through semisynthesis, taxol is produced.



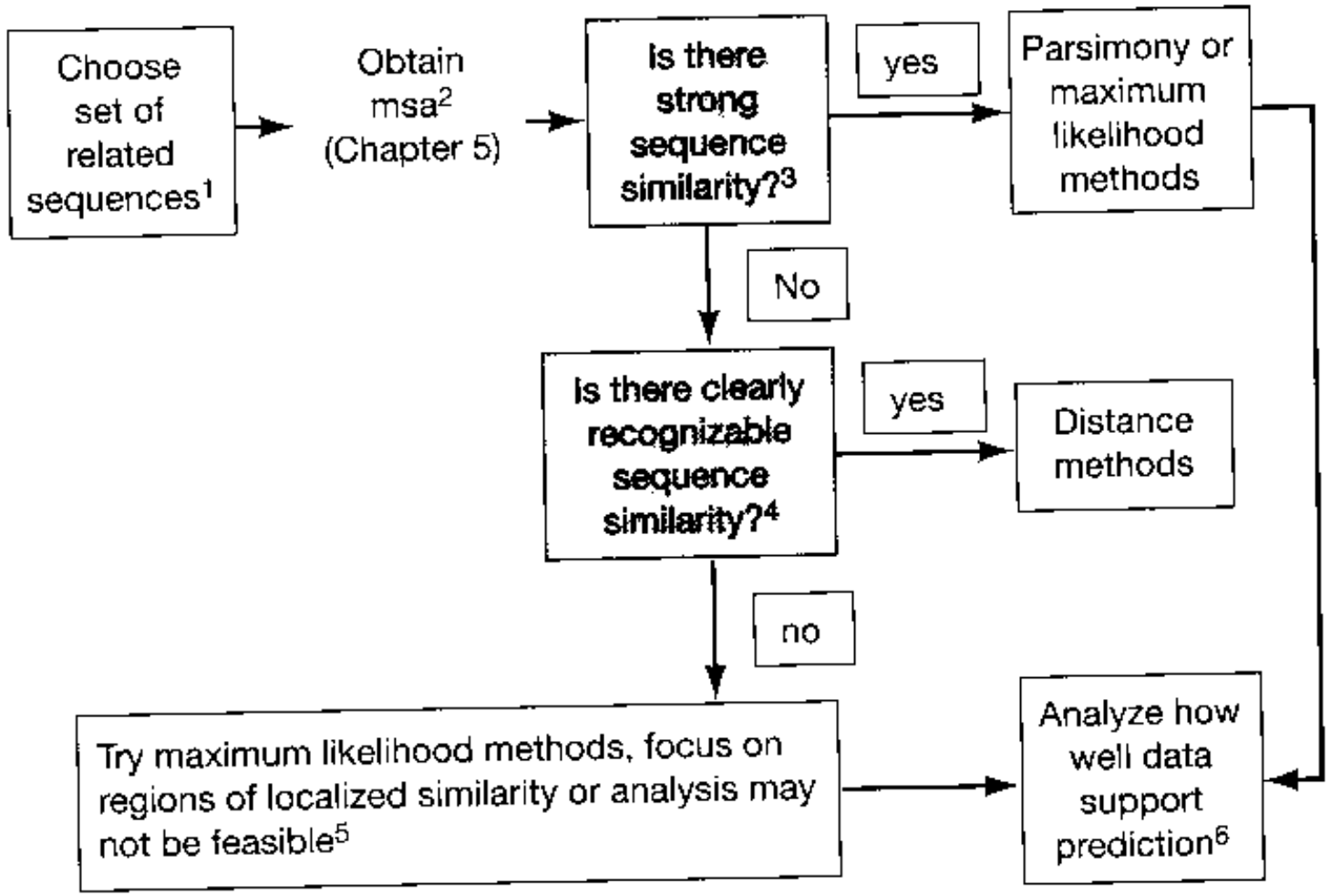
Pacific Yew

Phylogeny reconstruction

- Depending on the input, there are two computational problems for reconstructing the phylogeny:
  - Character based
  - Distance Based

- Below, we first describe character based method.

How to choose a phylogenetic prediction method?



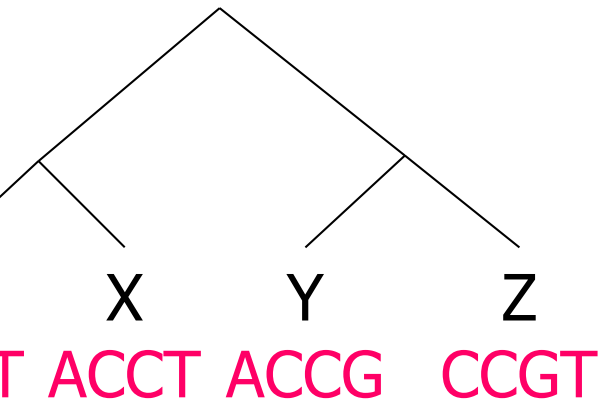
Source: D.W.Mount, Bioinformatics: Sequence and Genome Analysis, Cold Spring Harbor Press, 2004

Character Based Phylogenetic Tree Reconstruction

Character Based

- Input: each species is described by a set of characters
  - A character can be a base in a specific position in its DNA sequence, the number of eyes of the species, etc
- Output: a tree which best explain the input

	1	2	3	4
W	A	C	G	T
X	A	C	C	T
Y	A	C	C	G
Z	C	C	G	T



Outline for Character based methods

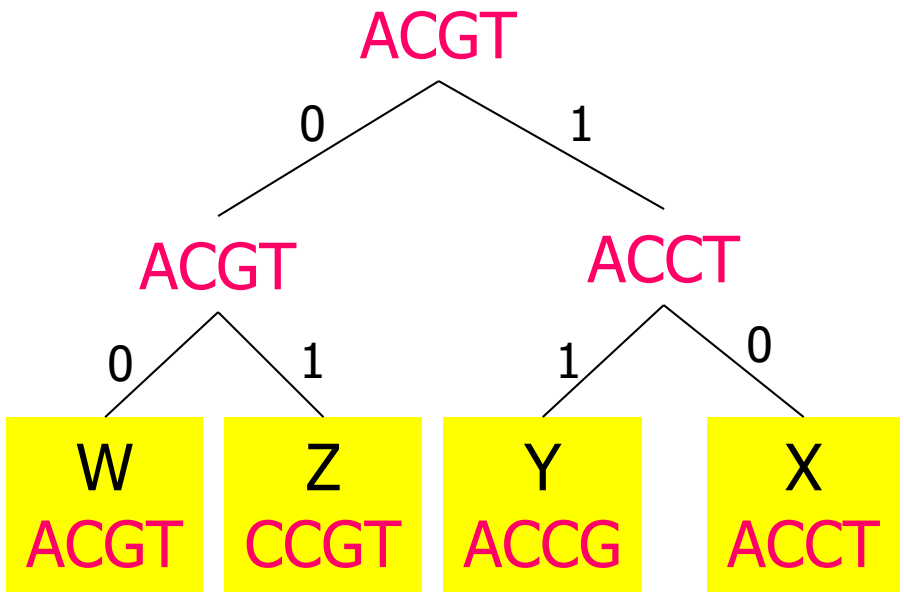
- Parsimony
- Compatibility
- Maximum Likelihood

Parsimony

- Idea: Build a phylogeny with the fewest point mutations
- Formal Definition:
  - Let S be a set of (DNA or Protein) sequences
  - Denote H(x, y) be the hamming distance between two sequences x and y
  - The most parsimonious tree is a tree T leaf-labeled by S and each internal node is assigned a sequence such that $H(T) = \sum_{(x, y) \in E(T)} H(x, y)$ is minimized. Note that H(T) is called the parsimony length of T

Example (4 species, each is represented by a sequence of 4 characters)

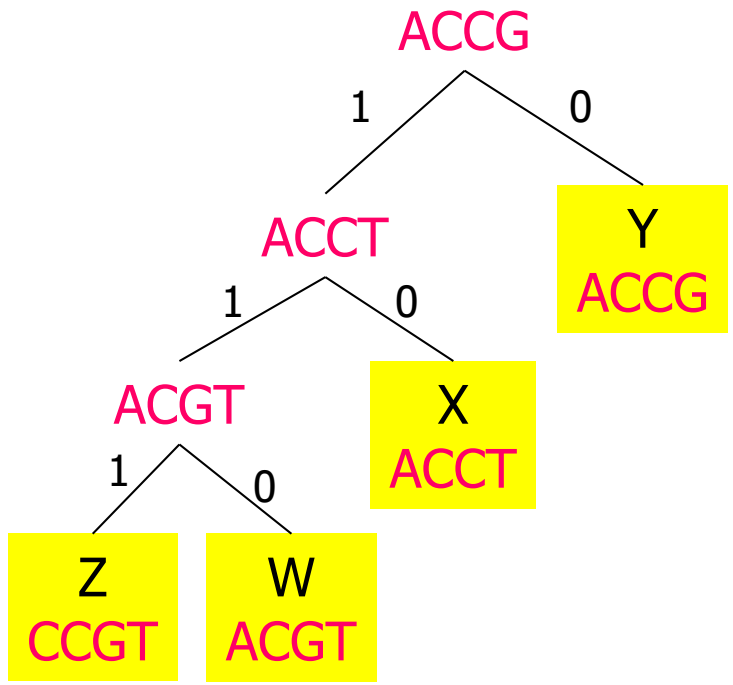
	1	2	3	4
W	A	C	G	T
X	A	C	C	T
Y	A	C	C	G
Z	C	C	G	T



This is the most parsimonious tree.
Its parsimony length is 3.

Example (4 species, each is represented by a sequence of 4 characters)

	1	2	3	4
W	A	C	G	T
X	A	C	C	T
Y	A	C	C	G
Z	C	C	G	T



This is another most parsimonious tree.
Its parsimony length is 3.

Computational Problems

- Computing the most parsimonious tree can be divided into two subproblems.

- Small Parsimony problem: Given a tree topology T, we want to find its parsimony length.
- Large Parsimony problem: Among all trees, find the most parsimonious tree.

Small Parsimony Problem

- Input: A set S of aligned sequences and the topology of a rooted phylogeny T with leave labeled by S
- Goal: Find parsimony length of T

- This problem can be solved in polynomial time using Fitch's algorithm

Simple case: each sequence only has one character

- Input: a leaf-labeled tree T where each leaf v is labeled by a single character v_c
- Output: a fully-labeled tree which is also the most parsimonious tree of T

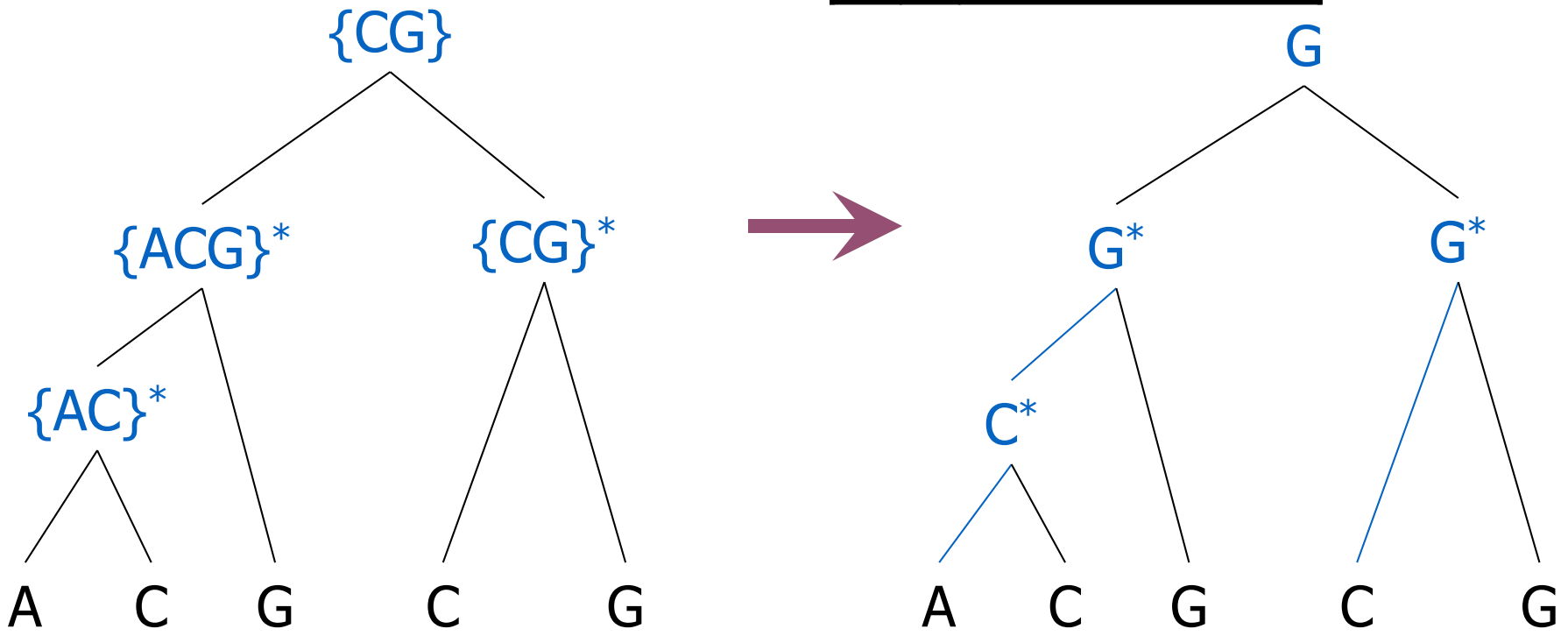
1. For every leaf v, let $S_v = \{v_c\}$.
2. For every internal node v with children u, w, let

$$S_v = \begin{cases} S_u \cap S_w & \text{if } S_u \cap S_w = \emptyset \\ S_u \cup S_w & \text{otherwise} \end{cases}$$

3. For every node v in preorder,
   - Let u be its parent. If $u_c \in S_v$, set $v_c \leftarrow u_c$; otherwise, assign any character in S_v to v_c.

An example

	1	…
S_1	A	…
S_2	C	…
S_3	G	…
S_4	C	…
S_5	G	…



- Each asterisk(*) requires a change in one of the edges to its children
- Time complexity: O(nk) where k is the size of the alphabet (which is 4 for DNA and 20 for protein)

Each sequence has m characters

- Note that the i^{th} character and the j^{th} character are independent for any i and j.
- Thus, this problem can be solved using m instance of the simple case problem.
- Time complexity is O(mnk).

Large Parsimony Problem

- Input: A set S of aligned sequences
- Output: the most parsimonious tree

- Large Parsimony Problem is NP-hard
- Exact solution for large parsimony
- Heuristics for large parsimony
- Large Parsimony Problem can be 2-approximated in polynomial time

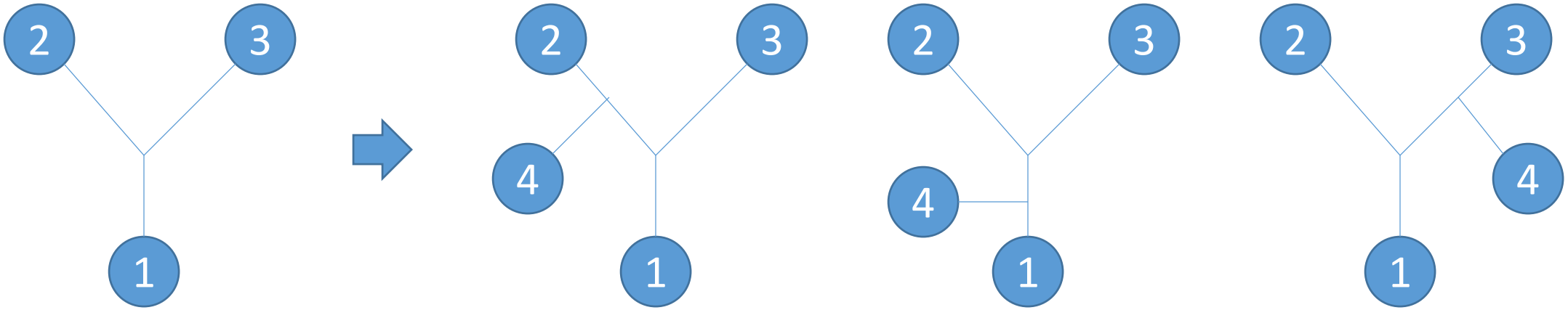
Exact solution

- 1. Generate all possible trees
- 2. For each tree T, we solve the small parsimony problem and compute the score w(T)
- 3. Report the tree with the smallest score.

- As shown in the following slides, there are exponential possible trees. This solution runs in exponential time.

Number of unrooted phylogenetic trees with 3 or 4 leaves

- Let c_k be the number of unrooted phylogenetic trees with k leaves.
- There is only one unrooted tree with 3 leaves. Hence, $c_3=1$.
- An unrooted phylogenetic tree with 4 leaves can be formed by inserting the 4^{th} leaf to one of the edge. Hence, $c_4=3$.



Number of unrooted phylogenetic trees with n leaves

- In general, an unrooted phylogenetic tree with k leaves has (2k-3) edges.
- Hence, $c_n = (2(n-1)-3)c_{n-1} = (2n-5)c_{n-1} = (2n-5)(2n-7)c_{n-2}$.
- Then, $c_n = (2n-5)*(2n-7)*\ldots*3*1 = \frac{(2n-5)*(2n-6)*(2n-7)*\cdots*3*2*1}{(2n-6)*(2n-8)*\cdots*2}$, which equals $\frac{(2n-5)!}{2^{n-3}(n-3)!} = 2^{\Theta(n \lg n)}$.

- Example: For 10 taxa, the number of trees is 2,027,025.

Heuristics

- Given a tree T, Neighbor(T) is the set of trees that are generated by branch-swapping.

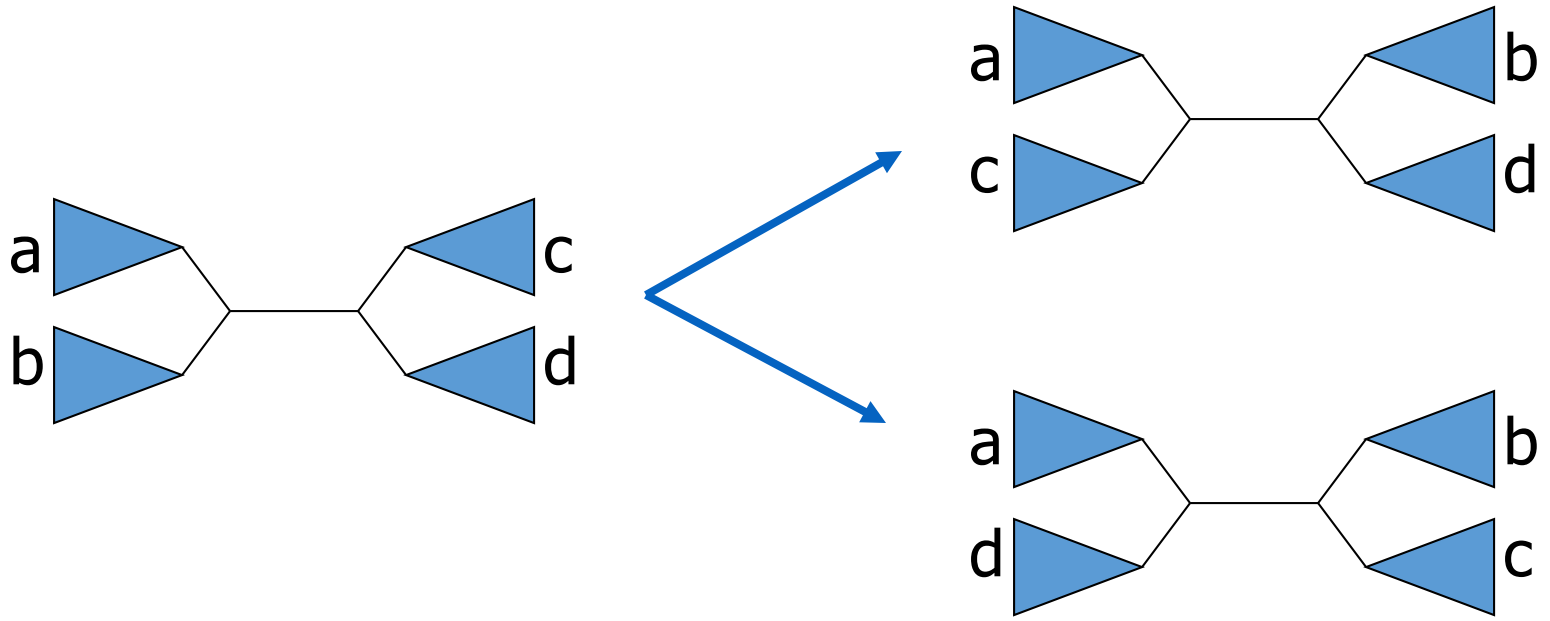
- Start with a random tree T
- Iterate t times
  - Find $T' \in$ Neighbor(T) such that $w(T') < w(T)$ [w(T) is the parsimony score]
  - Set T = T'
- Time analysis = t * O(nm) * Size_Of_Neighbor.

branch-swapping

- Three possible branch-swappings:
  - Nearest neighbor interchange (NNI)
    - A tree T with n leaves has 2(n-3) possible NNIs
  - Subtree pruning and regrafting (SPR)
    - A tree T with n leaves has $O(n^2)$ possible SPRs
  - Tree bisection and reconnection (TBR)
    - A tree T with n leaves has $O(n^3)$ possible TBRs
- Note: NNI $\subseteq$ SPR $\subseteq$ TBR.

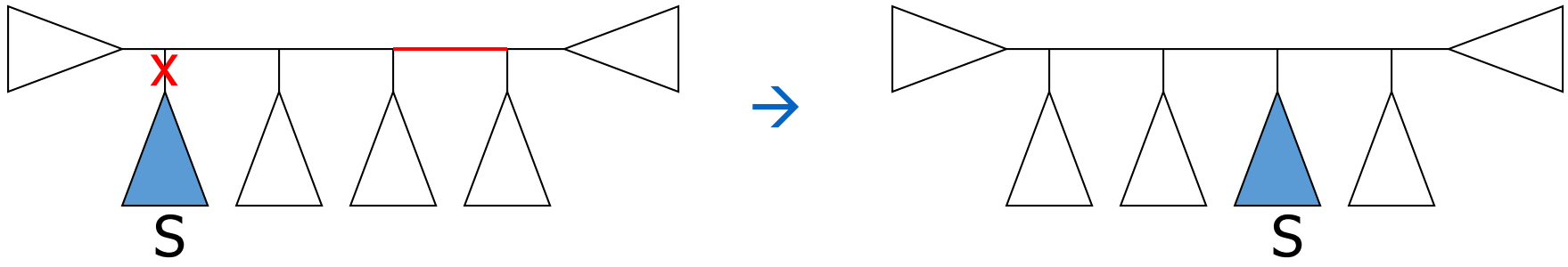
Finding optimal trees - heuristics

- Nearest neighbor interchange (NNI): exchanges two subtrees across an edge.



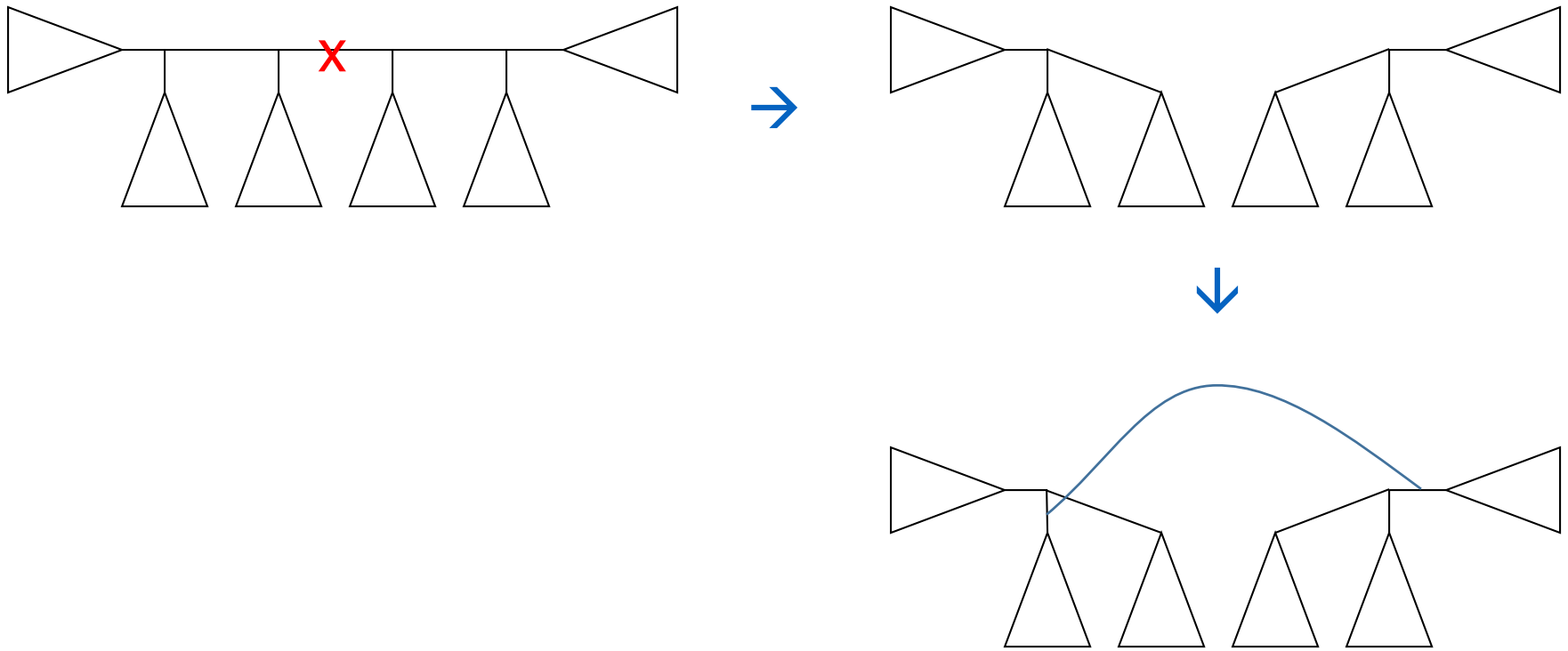
Finding optimal trees - heuristics

- Subtree pruning and regrafting (SPR): the operation of detaching a subtree and reattached it to the middle of another edge exchanges two subtrees across an edge.



Finding optimal trees - heuristics

- Tree bisection and reconnection (TBR): Bisect the tree into two; then, join two edges from the two trees.

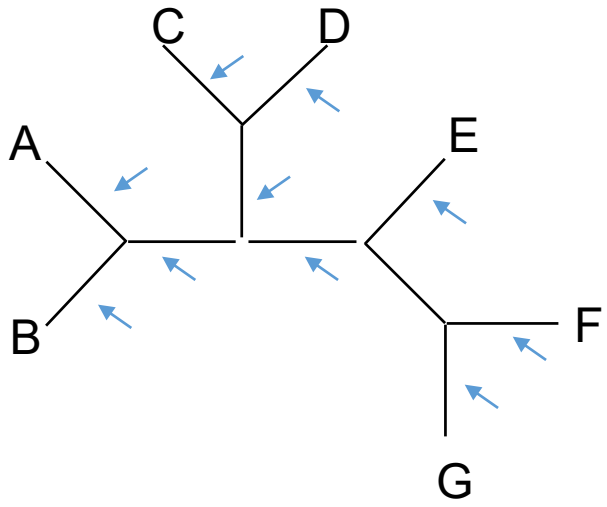


How to generate an initial random tree?

- Possible ways to generate an initial random tree:
  - Generate random tree by inserting taxa one by one randomly.
  - Stepwise addition heuristics
  - Distance-based method (like Neighbor-joining method)

Stepwise addition heuristics

- 1. Start with a star tree T with random 3 taxa
- 2. While we still have some taxon not included in T,
  - Randomly select an uninserted taxon x;
  - If T has k edges, there are k ways to insert x
    - set T be the one with the minimum parsimony score
- 3. Report T

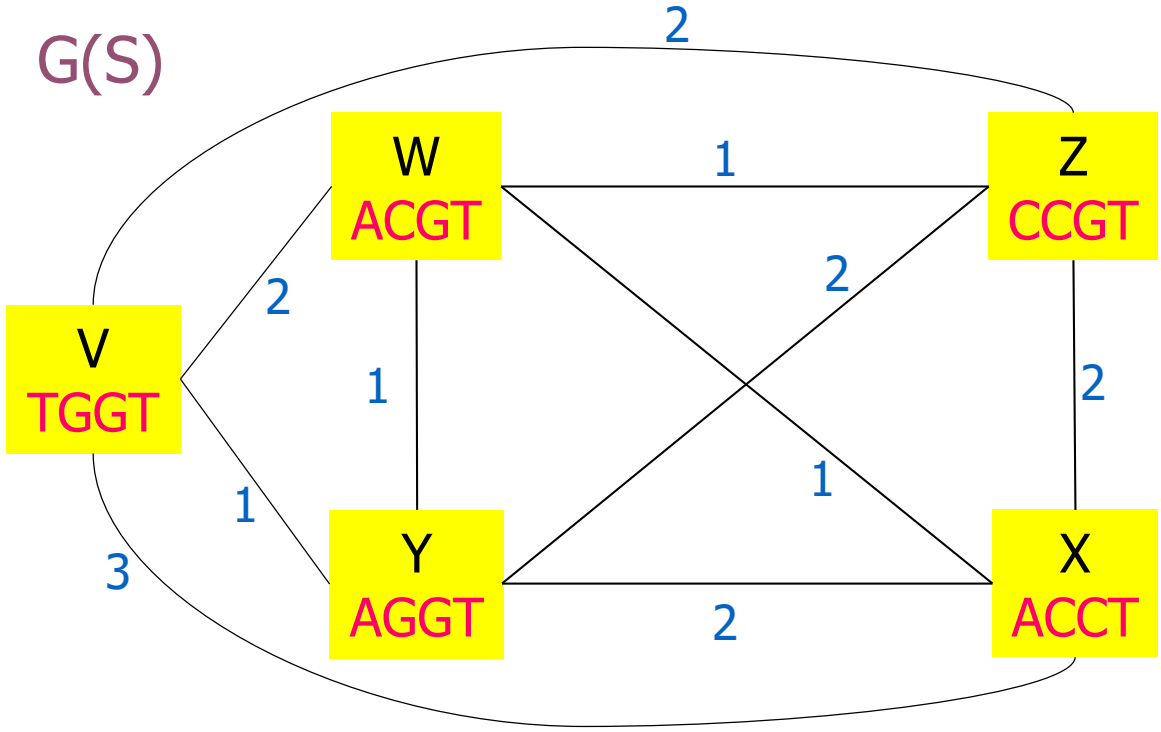


Approximation algorithm

- Given a set S of sequences, define G(S) be a weight complete graph whose nodes are labeled by S and each edge (i, j) has weight H(i, j).

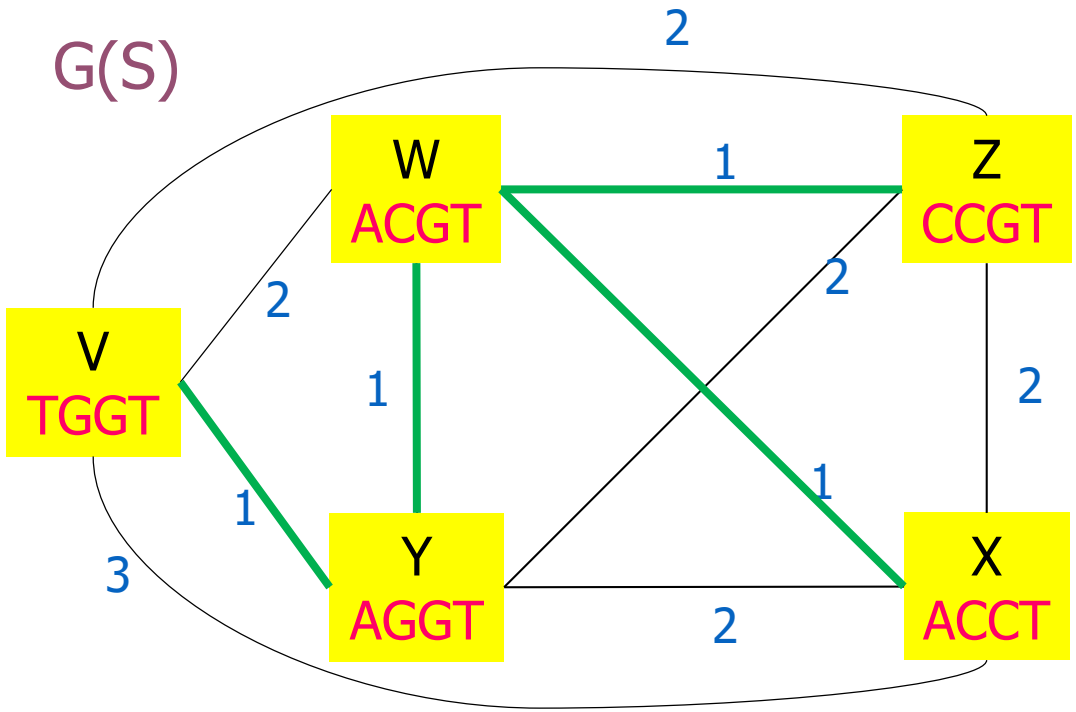
	1	2	3	4
V	T	G	G	T
W	A	C	G	T
X	A	C	C	T
Y	A	G	G	T
Z	C	C	G	T

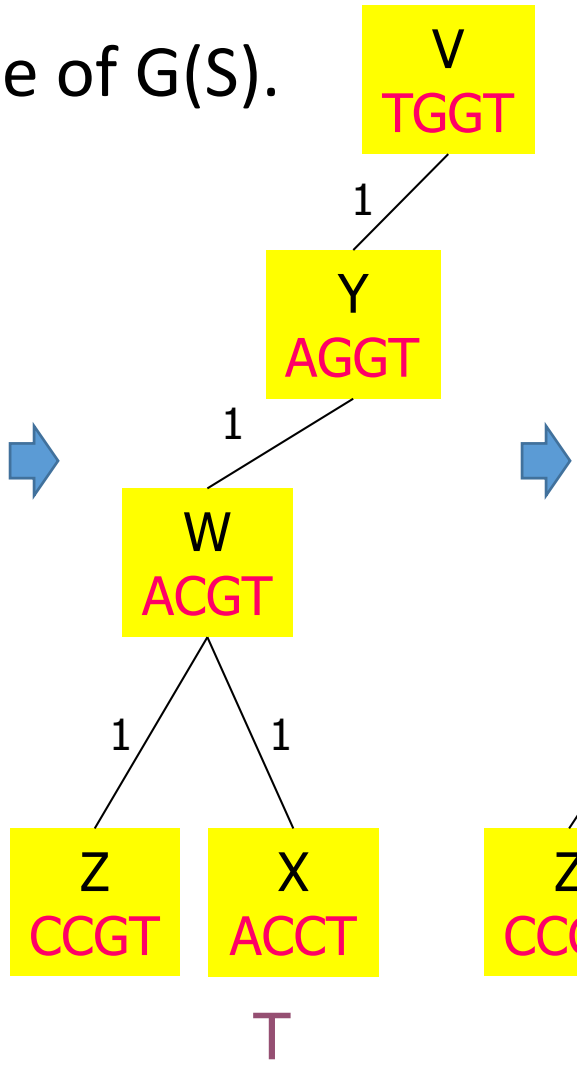
S

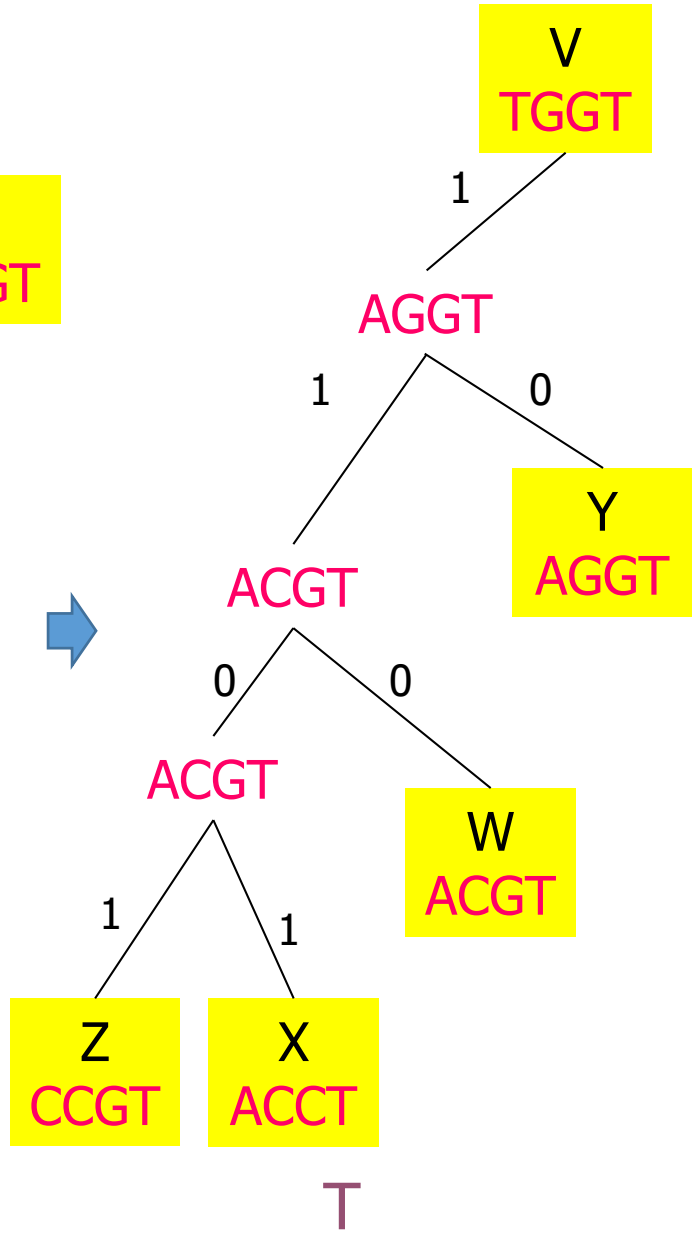


Approximation algorithm (II)

- Let T be a minimum spanning tree of G(S).







Approximation algorithm (III)

- Theorem: Let T be a minimum spanning tree of G(S). Then, the parsimony length of T is at most twice that of the most parsimonious tree.
  - Proof: Let T* be the most parsimonious tree.
  - Let C be an Euler cycle of T*.
  - Let P contains only the nodes of G(S) ordered in the way in which they appear in C.
  - $w(T) \leq w(P) \leq w(C) = 2\, w(T^*)$

Application of Maximum Parsimony: Predicting evolution of influenza

- Influenza is a fast evolving virus.
- Bush and Fitch et al. show that phylogenetic analyses of the human influenza A (subtype H3) virus can be used to make predictions about the evolutionary course of future human influenza strains.
- The predicted strains of flu virus is included in the vaccine prepared each year to protect against the upcoming influenza season.
  - Bush, R. M., C. A. Bender, et al. (1999) "Predicting the evolution of human influenza A." Science 286: 1921-1925.

How to build the influenza tree?

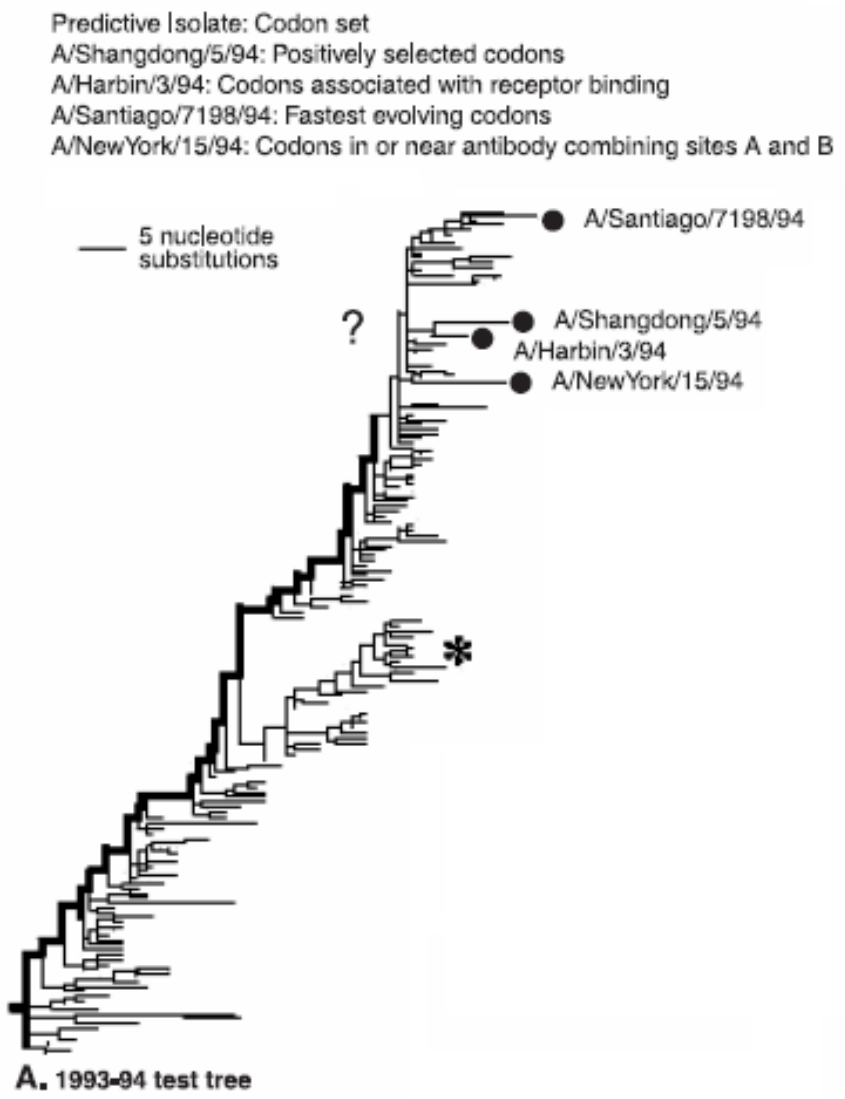
- The HA1 domain of the hemagglutinin gene of human influenza A subtype H3

- The HA1 domains are aligned using multiple sequence alignment algorithm. Then, we get the input matrix.

- By maximum parsimony, we build the tree.

Observation from the influenza tree

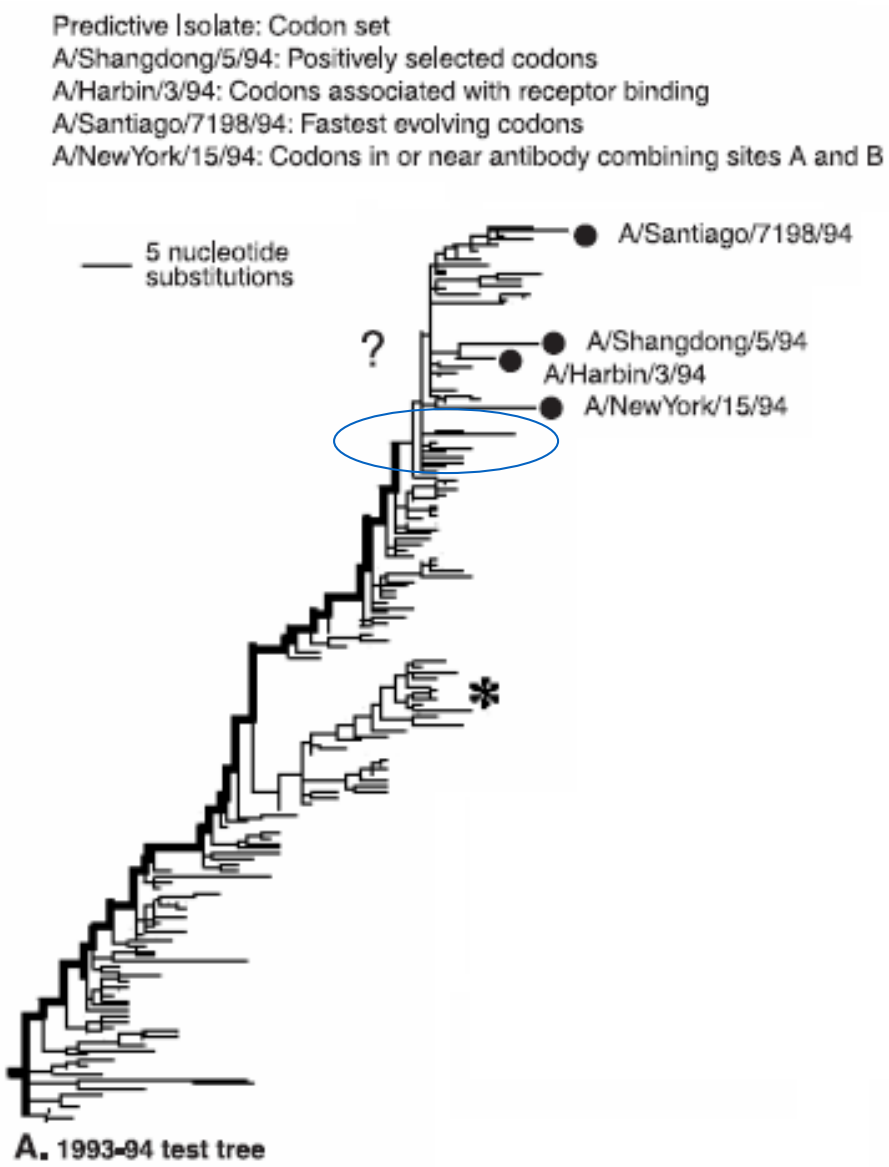
- The tree shows the evolution of HA1 domain of the hemagglutinin gene of human influenza A subtype H3
  - Build by Maximum Parsimony using isolates from 1983-1994
- There is a selection stress. (The tree is skew.)
  - The bold path shows the single evolutionarily successful linkage.
- At least 18 of the 329 H3 HA1 codons have been under positive selection.



Question: What is the trend of the evolution lineage?

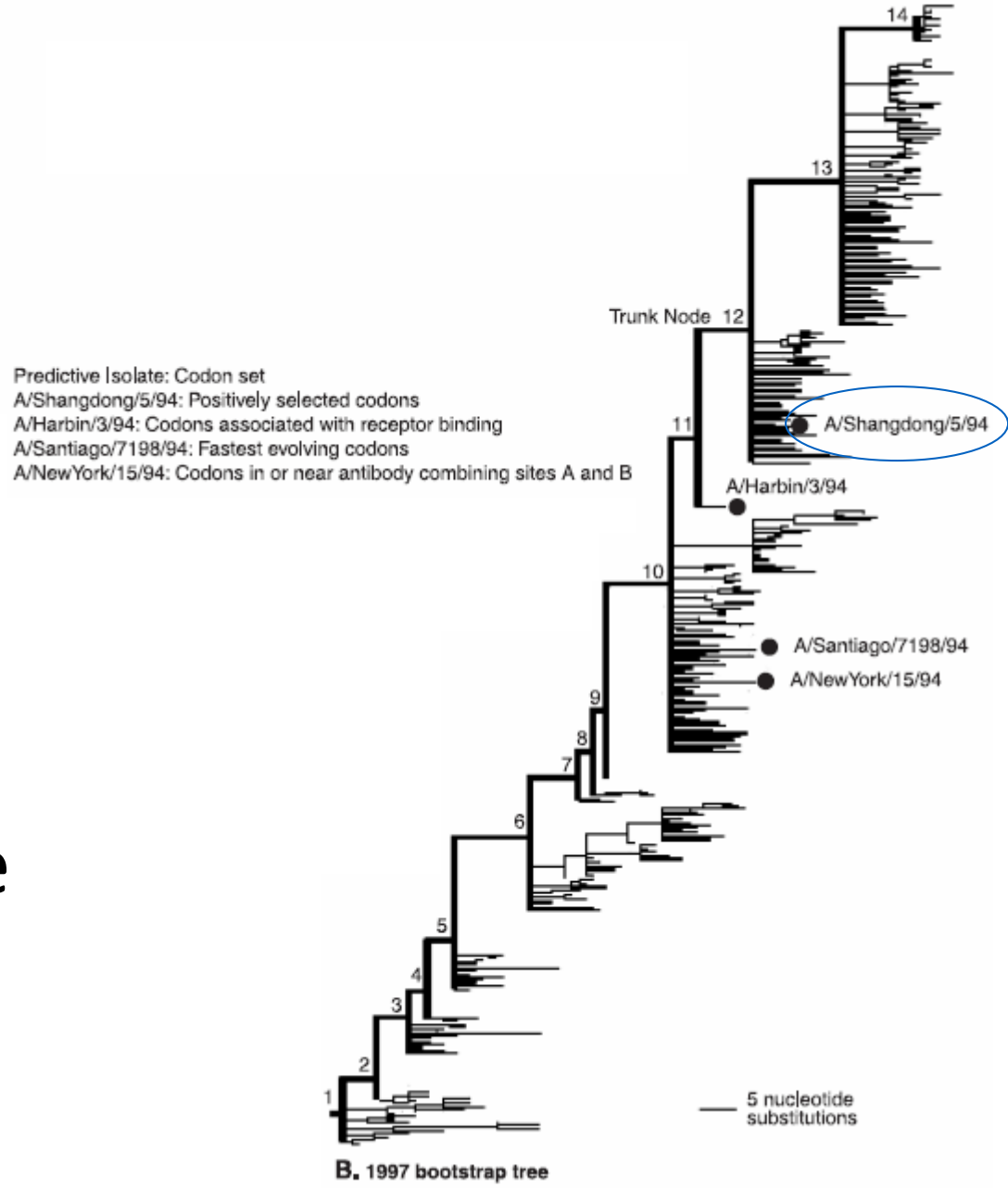
- Hypothesis:
  - If the selective pressure were to evade the host immune response, then viruses sustaining mutations at these 18 codons in the past should have been more fit than other coexisting viruses.

- Based on this idea, the authors predict the future influenza looks similar to A/Shangdong/5/94.



Is the prediction accurate?

- The right tree is reconstructed from the influenza in 1985-1997.
- A/Shangdong/5/94 is relative more fit to isolates in the future influenza seasons.

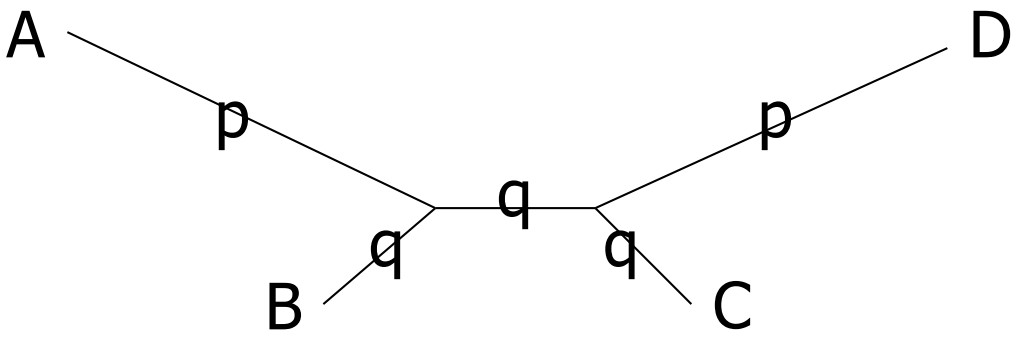


Advantages and disadvantages of parsimony

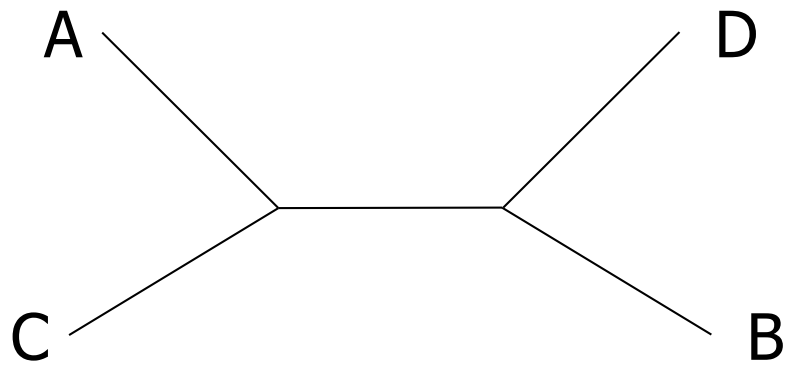
- Advantages
  - Does not depend on an explicity model of evolution
  - It not only predicts the tree, but also predicts the internal nodes
  - Parsimony can predict an accurate tree if homoplasy is rare or widely randomly distributed.
    - Homoplasy is a shared character between two or more animals that did not arise from a common ancestor. (E.g. both bird and bat have wing. But birth and bat have different ancestors.)
- Disadvantages
  - Underestimate branch lengths
  - Statistically inconsistent
    - This means that given long enough sequences, maximum parsimony may not be able to recover the true tree with arbitrarily high probability.

Why not statistically consistent?

- Felsenstein (1973) observes the issue of long branch attraction.



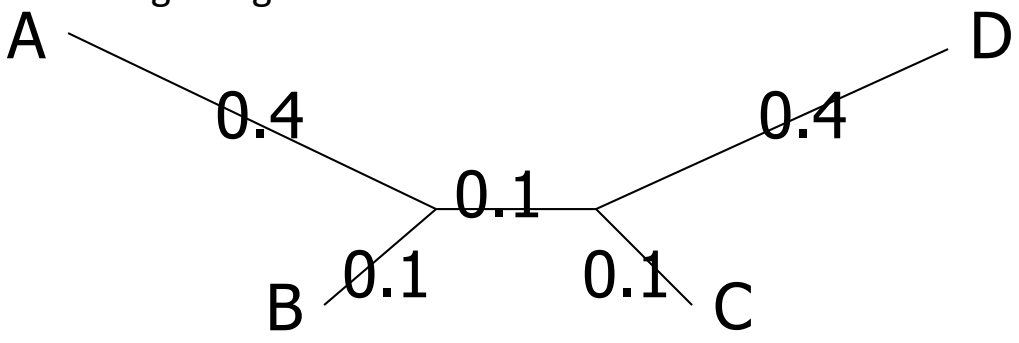
Model tree
p is much longer than q



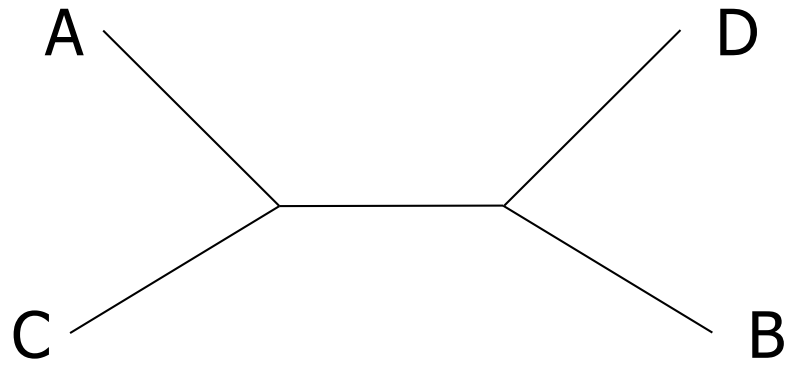
Parsimony tree

Example

- Felsenstein (1973) considers the following CF model.
- Under CF model,
  - Probability of (ABCD = 1100 or 0011): 0.0682
  - Probability of (ABCD = 1010 or 0101): 0.1242
  - Probability of (ABCD = 1001 or 0110): 0.0522
- The probabilities of other vectors do not affect the tree topology.
- Since ABCD = 1010 or 0101 is the most common, the parsimonious tree is converged to AC|BD, which is incorrect.
- Hence, if a character-based matrix is generated from this model tree, as you have more characters, the probability of getting the correct tree trends to 0.



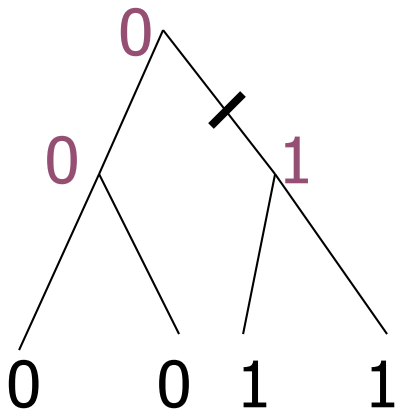
Model tree



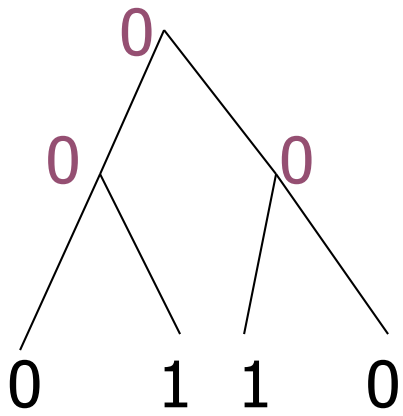
Parsimony tree

Compatibility

- Compatibility is a simplification of parsimony.
- Definition:
  - A binary character c is compatible to a leaf-labeled tree T if and only if there exist an assignment of states to the internal nodes of T such that a change of status exists in exactly one edge



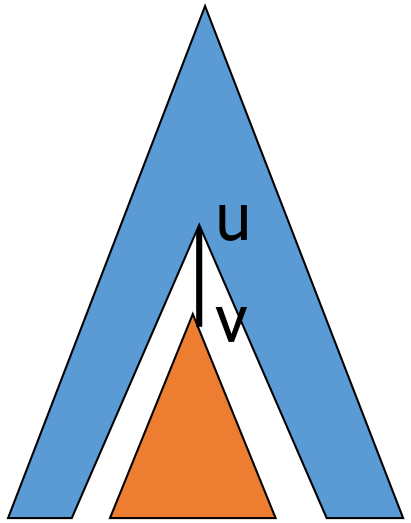
One status change!
c is compatible to T



Two status changes!
c is not compatible to T

More on compatibility

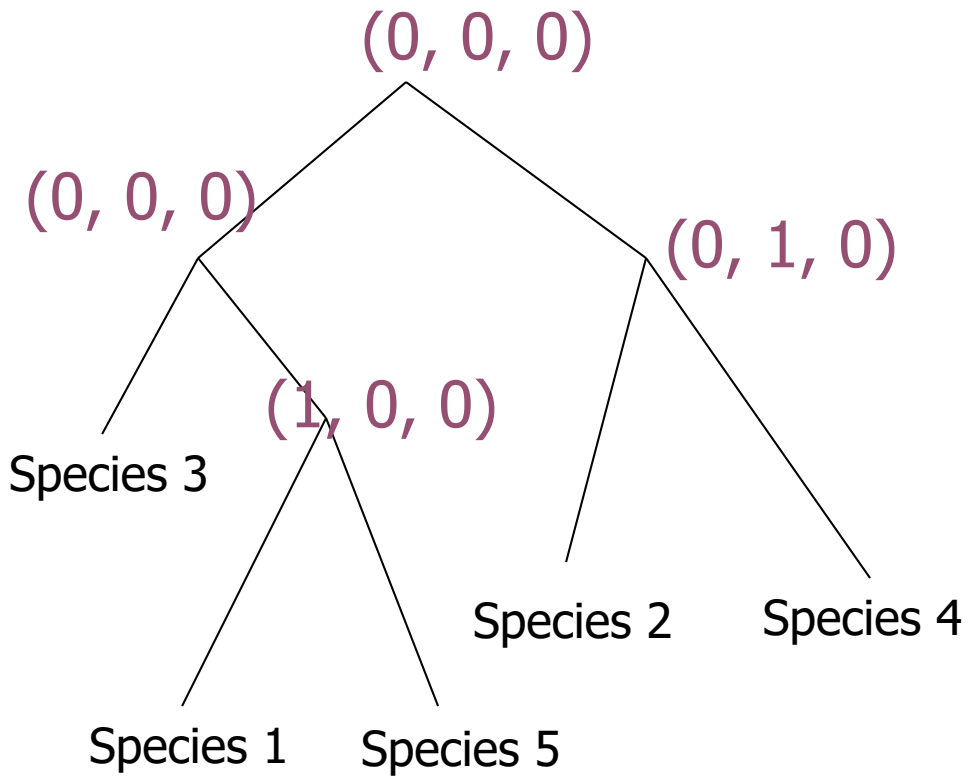
- In fact, if character c is compatible to a tree T, we can identify an edge (u, v) in T so that
  - The leaves in the subtree of v have state s for character c
  - The other leaves have state (1-s) for character c



Example

- Characters 1, 2, and 3 are all compatible!

M	X_1	X_2	X_3
Species 1	1	0	1
Species 2	0	1	0
Species 3	0	0	0
Species 4	0	1	0
Species 5	1	0	0



Perfect phylogeny

- Input: n species, each is characterized by m binary characters.
  - This input can be represented using a binary matrix M with n rows and m columns.
- M admits a perfect phylogeny if
  - there exists a rooted tree T for the n species such that all m characters are compatible.

Computational Problems

- Input: Given n species, each characterized by m binary characters. (Represented using a binary matrix M.)
- Compatibility Problem
  - Check whether this set of species admits a perfect phylogeny.
- Perfect Phylogeny Problem (Large Compatibility Problem)
  - Find a maximum set of characters which admits a perfect phylogeny

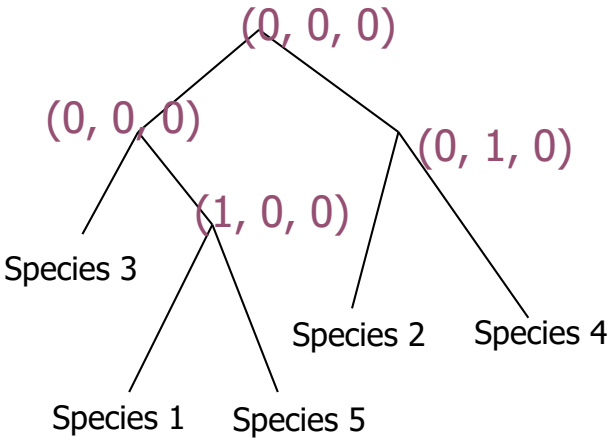
Compatibility problem

- Divide the discussion into two parts:
  1. Check whether M admits a perfect phylogeny
  2. If M admits a perfect phylogeny, recover the tree

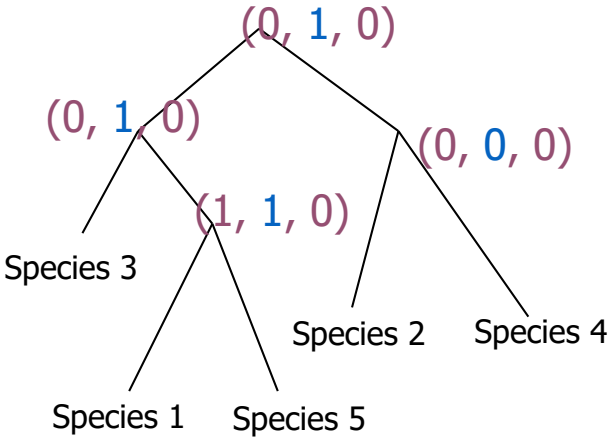
Observation

- If M admits a perfect phylogeny T, after exchanging 0 and 1 in any column, the resulting matrix M' still admits the same perfect phylogeny T.

M	X_1	X_2	X_3
Species 1	1	0	1
Species 2	0	1	0
Species 3	0	0	0
Species 4	0	1	0
Species 5	1	0	0



M'	X_1	X_2	X_3
Species 1	1	1	1
Species 2	0	0	0
Species 3	0	1	0
Species 4	0	0	0
Species 5	1	1	0



Assumption on the input matrix M

- Based on the previous slide, we assume for every column of M,
  - The number of state 1 < the number of state 0.

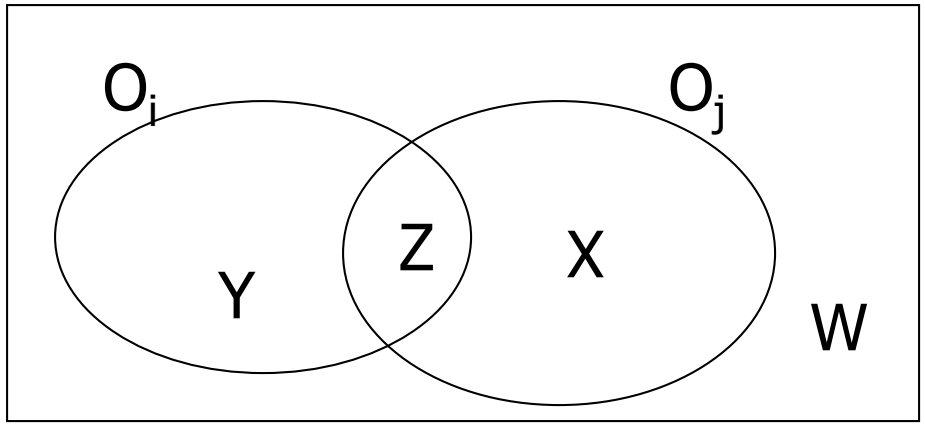
- Otherwise, we exchange 0 and 1 and such transformation has no effect on compatibility!

Main lemma

- For every character i, let O_i be the set of species with state 1.
- Characters i and j are pairwise compatible if
  - O_i and O_j are disjoint or one of them contains the other.
  - (Note: pairwise compatible ≠ compatible!)
- Lemma: M admits a perfect phylogeny if and only if for every characters i and j, they are pairwise compatible.

- This lemma is also called 4-gamete test.

Proof(➔)



- Given that M admits a perfect phylogeny
- Note that, for every character i, $|O_i| \leq n/2$.
- Assume that character i and j are not pairwise compatible.
- That is, there exists three species X,Y,Z such that $Y,Z \in O_i$, $X \notin O_i$ and $X,Z \in O_j$, $Y \notin O_j$.
- Since $O_i \cap O_j$ is non-empty, $|O_i \cup O_j| = |O_i| + |O_j| - |O_i \cap O_j| < n$.
  - Thus, there exists a species $W \notin O_i, O_j$.
- By character i, Y and Z are in the same partition in T, while X and W are in another partition
- By character j, X and Z are in the same partition in T and W and Y are in the same partition in T.
- Impossible! We arrived at contradiction!

Proof (🡐)

- Exercise!

Simple solution for compatibility

- Based on the previous lemma, we get the following algorithm.

Algorithm

- For every characters i and j,
  - Check whether i and j are pairwise compatible.
  - If no, return “cannot admit a perfect phylogeny”!
- Return “admits a perfect phylogeny”!

- Time complexity: $O(m^2 n)$

Can we get a better algorithm?

- Yes! We can have an O(mn) time algorithm
- Idea:
  - The previous algorithm is slow since it takes long time to check, for every i, j, whether O_i and O_j are disjoint or one of them contains the other
  - Observation: If $|O_i| \geq |O_j|$,
    - we only need to check if ether (1) $O_i \cap O_j = \Phi$ or (2) $O_j \subseteq O_i$.
    - We don't need to check if $O_i \subseteq O_j$.

Step 1

- Relabel the characters so that $|O_i| \geq |O_j|$ if $i<j$

M	X_1	X_2	X_3
Species 1	1	0	1
Species 2	0	1	0
Species 3	0	0	0
Species 4	0	1	0
Species 5	1	0	0

$|O_1|=2$,
$|O_2|=2$,
$|O_3|=1$

Step 2

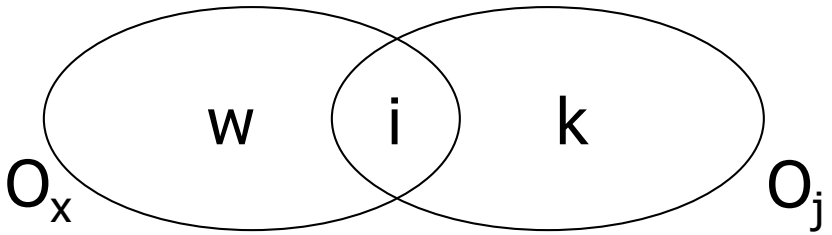
- For every species i and character j,
  - If $M_{ij}=1$, let L_{ij} be the biggest $k<j$ such that $M_{ik}=1$. If no such k, $L_{ij} = -1$
  - If $M_{ij}=0$, let $L_{ij}=0$.

M	X_1	X_2	X_3
Species 1	1	0	1
Species 2	0	1	0
Species 3	0	0	0
Species 4	0	1	0
Species 5	1	0	0

L	X_1	X_2	X_3
Species 1	-1	0	1
Species 2	0	-1	0
Species 3	0	0	0
Species 4	0	-1	0
Species 5	-1	0	0

Technical Lemma

- Lemma: For some character j, if there exist two nonzero entries L_{ij} and L_{kj} such that $L_{ij} \neq L_{kj}$, then M does not admit a perfect phylogeny.
- Proof:
  - Suppose $L_{ij}=x$ and $L_{kj}=x'$. WLOG, $x>x'$.
  - By definition, $M_{ij}=M_{kj}=1$, $M_{ix}=1$, $M_{kx}=0$
  - O_j contains species i and species k and O_x contains species i, but not species k. It means that (1) $O_j \cap O_x \neq \emptyset$ and (2) $O_j \not\subseteq O_x$.
  - Note that $j>x$. Thus, $|O_x| \geq |O_j|$
  - As $k \notin O_x$, O_x should contain some species w which does not appear in O_j. So, (3) O_x is not subset of O_j.
  - By previous lemma, M does not admit a perfect phylogeny.



M

	x'		x	…	j	
	…		…		…		…
i	…		…	1	…	1	…
	…		…		…		…
k	…		…	0	…	1	…
	…		…		…		…

L

	x'		x	…	j	
	…		…		…		…
i	…		…		…	x	…
	…		…		…		…
k	…		…		…	x'	…
	…		…		…		…

Step 3

- For every character j, check if there exist i and k such that $L_{ij} \neq L_{kj}$ and both L_{ij} and L_{kj} are nonzero.
- If yes, return "does not admit a perfect phylogeny".
- Otherwise, "admits a perfect phylogeny".

L	X_1	X_2	X_3
Species 1	-1	0	1
Species 2	0	-1	0
Species 3	0	0	0
Species 4	0	-1	0
Species 5	-1	0	0

For every character j (column j), we can't find two nonzero entries which are different. So, for all i, j, O_i and O_j are disjoint or one of them contains the other

Time complexity

- Step 1 takes O(mn) time (by radix sort)
- Steps 2 and 3 can be computed in O(mn) time!
- Thus, we can decides whether M admits a perfect phylogeny or not in O(mn) time.

Tree reconstruction

Algorithm

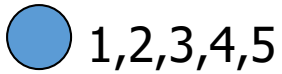
Input: A character-state matrix M with $O_i \geq O_j$ for $1 \leq i < j \leq n$

- Let T be a tree containing the single root node r. N(r)={1,…,n}
- For every character j where j=1 to m
  - Find a leaf $v \in T$ such that
    - N(v) can be partitioned into two non-empty sets N_0 and N_1 where N_s={ $x \in N(v)$ | character j of species x is of state s} for s=0,1
    - /* Note: we can only split one leaf v */
  - Create two children v_0 and v_1 for v
  - Set $N(v_0) = N_0$, $N(v_1) = N_1$
  - Set $N(v) = \Phi$
- For every leaf v s.t. N(v) is nonempty,
  - If |N(v)|>1, let the species in N(v) be the children of v
  - If |N(v)|=1, leaf v represents the species in N(v)

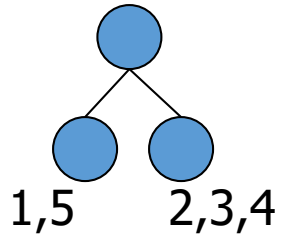
Example

Assume the columns of M is reordered such that $|O_i| \geq |O_j|$ for $i<j$.

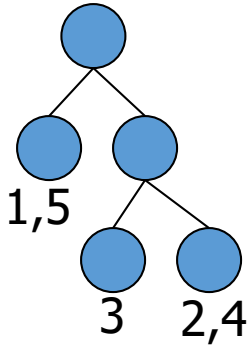
M	X_1	X_2	X_3
Species 1	1	0	1
Species 2	0	1	0
Species 3	0	0	0
Species 4	0	1	0
Species 5	1	0	0



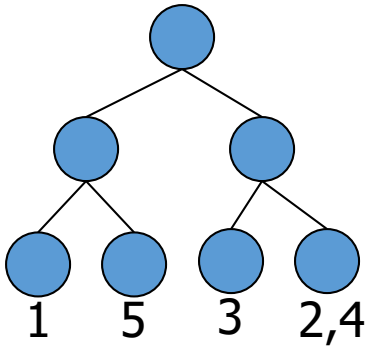
Initial case



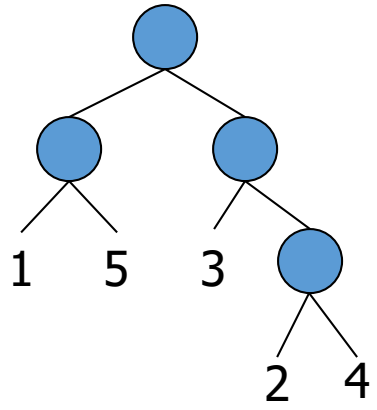
character 1



character 2



character 3



final

Time analysis

- For every character j, it takes O(n) time to identify a node and to split the node
- Thus, the total time is O(nm)

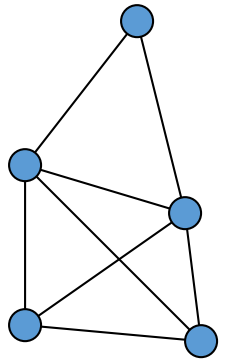
Large Compatibility Problem

- Find the maximum set of characters which admits a perfect phylogeny!
- This problem is NP-hard!

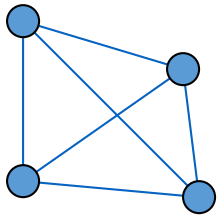
- We discuss how to solve Large Compatibility Problem by transforming it to CLIQUE Problem.

CLIQUE Problem

- Given a graph G, the problem tries to find the maximum size subgraph H such that H is a complete graph.



G



H

- Note: this is an NP-complete problem

Large Compatibility Problem vs CLIQUE Problem

- Given an instance of M, define a graph G where
  - Each vertex i in G corresponds to a character in M
  - (i, j) is an edge in G if i and j are pairwise compatible.
- Note:
  1. G can be constructed in polynomial time
  2. G contains a clique of size B if and only if M contains a subset of compatible characters whose size is B.
- Thus, we transforms the large compatibility problem to a CLIQUE problem.

Algorithm for solving large compatibility problem

Input: M

1. Obtain G based on M
2. Find a maximum clique C in G
3. Build the perfect phylogeny of C (as C is a maximum set of compatible characters)

- The bottleneck is step 2. So, the time complexity is exponential.

Finding maximum clique

- The fastest algorithm for finding the maximum clique is by Robson (2001) which runs in $O(2^{n/4})$ time.
- Below, we describe the Bron–Kerbosch algorithm (1973), which is fast in practice. (Running time is $O(3^{n/3})$.)
  - The Bron-Kerbosch algorithm enumerates all maximal cliques.
  - From these maximal cliques, we identify the maximum clique.

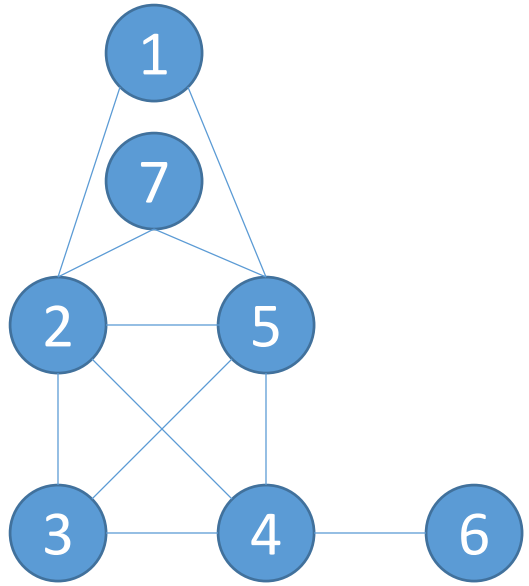
C. Bron, J. Kerbosch. Finding all cliques of an undirected graph, Communication of ACM, 1973.

J. M. Robson. Finding a maximum independent set in time $O(2^{0.25n})$. 2001.

Simple clique finding algorithm

- Consider a graph G=(V,E).
- Given a clique R of G, let S be the set of nodes that have edges to all nodes in R.
- Note that S is the set of nodes that can be used to expand R into a bigger clique. In other words, $R \cup \{v\}$ is a clique for any v $\in S$.

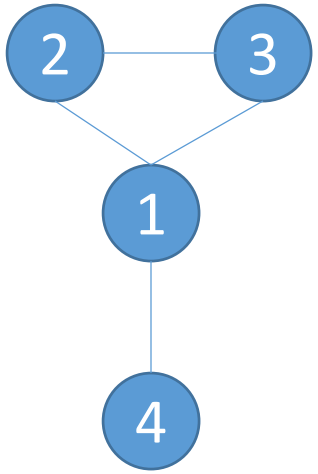
- Example:
- Suppose $R = \{2,5\}$. Then, $S = \{1,3,4,7\}$.
- Suppose $R = \{2,5,7\}$. Then, $S = \emptyset$.



Simple clique finding algorithm

- Consider a graph G=(V,E).
- Given a clique R of G, let S be the set of nodes that have edges to all nodes in R.
- $SimpleClique(R, S)$ returns all maximal cliques C where $R \subseteq C \subseteq R \cup S$.
- All maximal cliques of G can be found by calling $SimpleClique(\emptyset, V)$.
- Define $N(v)$ be the neighbors of v in the graph G=(V,E).

- Base case:
  - $SimpleClique(R, \emptyset) = \{R\}$
- Recursive case:
  - $SimpleClique(R, S) = \bigcup_{v \in S} SimpleClique\big(R \cup \{v\}, S \cap N(v)\big)$



Example: $SimpleClique(\emptyset, \{1,2,3,4\})$

$SimpleClique(\emptyset, \{1,2,3,4\})$

$SimpleClique(\emptyset \cup \{1\}, \{1,2,3,4\} \cap N(1))$

$SimpleClique(\{1\}, \{2,3,4\})$

$SimpleClique(\emptyset \cup \{2\}, \{1,2,3,4\} \cap N(2))$

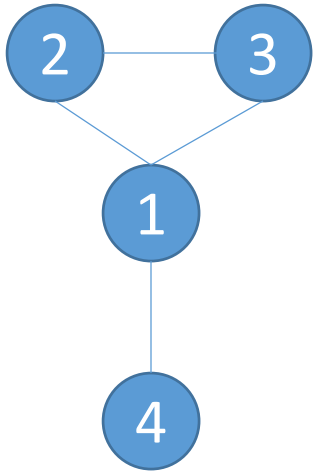
$SimpleClique(\{2\}, \{1,3\})$

$SimpleClique(\emptyset \cup \{3\}, \{1,2,3,4\} \cap N(3))$

$SimpleClique(\{3\}, \{1,2\})$

$SimpleClique(\emptyset \cup \{4\}, \{1,2,3,4\} \cap N(4))$

$SimpleClique(\{4\}, \{1\})$



Example: $SimpleClique(\emptyset, \{1,2,3,4\})$

$SimpleClique(\emptyset, \{1,2,3,4\})$

- $SimpleClique(\{1\}, \{2,3,4\})$
  - $SimpleClique(\{1,2\}, \{3\})$ → $SimpleClique(\{1,2,3\}, \emptyset)$
  - $SimpleClique(\{1,3\}, \{2\})$ → $SimpleClique(\{1,3,2\}, \emptyset)$
  - $SimpleClique(\{1,4\}, \emptyset)$
- $SimpleClique(\{2\}, \{1,3\})$
  - $SimpleClique(\{2,1\}, \{3\})$ → $SimpleClique(\{2,1,3\}, \emptyset)$
  - $SimpleClique(\{2,3\}, \{1\})$ → $SimpleClique(\{2,3,1\}, \emptyset)$
- $SimpleClique(\{3\}, \{1,2\})$
  - $SimpleClique(\{3,1\}, \{2\})$ → $SimpleClique(\{3,1,2\}, \emptyset)$
  - $SimpleClique(\{3,2\}, \{1\})$ → $SimpleClique(\{3,2,1\}, \emptyset)$
- $SimpleClique(\{4\}, \{1\})$ → $SimpleClique(\{4,1\}, \emptyset)$

SimpleClique is very slow

- $SimpleClique(\emptyset, V)$ enumerates all permutations of all cliques of G.
- For previous example, we enumerate all permutations of all cliques of {1,2,3} and {1,4}.

- The worst case happens when G is a complete graph of n nodes.
- Running time is at least n! since we need enumerate all n! permutations of {1, 2, …, n}.

Can we speedup SimpleClique?

- Basic Bron-Kerbosch algorithm avoids enumerating all permutations of the same clique.

- Idea: Try to avoid reporting the same clique multiple times.

Basic Bron-Kerbosch algorithm

- Consider a graph G=(V,E). Given a clique R of G, let S be the set of all nodes that have edges to all nodes in R. We partition S into 2 disjoint sets P and X such that
  - P is the set of nodes we used to expand the cliques
  - X is the set of vertices that will not be included to expand the cliques.
- BasicClique(R, P, X) returns all **maximal** cliques C of G where $R \subseteq C \subseteq R \cup P$.
  - Note that $S = P \cup X$ is the set of all nodes that have edges to all nodes in R.
  - X seems redundant. It is actually used to check if a clique is maximal.
- Our aim is to compute $BasicClique(\emptyset, V, \emptyset)$.

Recurrence for BasicClique

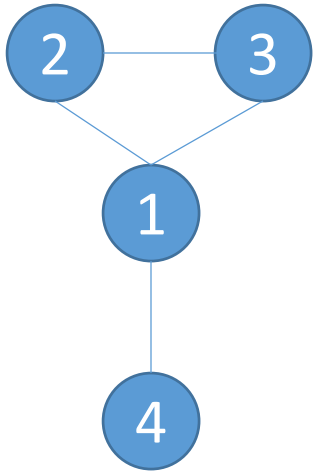
- Base case: $BasicClique(R, \emptyset, \emptyset) = \{R\}$, $BasicClique(R, \emptyset, X) = \emptyset$

- Suppose $P = \{v_1, v_2, \dots, v_k\}$.
- Recursive case:

$$BasicClique(R, P, X)$$
$$= BasicClique\big(R \cup \{v_1\}, \{v_1, \dots, v_k\} \cap N(v_1), X \cap N(v_1)\big)$$
$$\cup\, BasicClique\big(R \cup \{v_2\}, \{v_2, \dots, v_k\} \cap N(v_2), (X \cup \{v_1\}) \cap N(v_2)\big) \cup \cdots$$
$$\cup\, BasicClique\big(R \cup \{v_k\}, \{v_i, \dots, v_k\} \cap N(v_i), (X \cup \{v_1, \dots, v_{i-1}\}) \cap N(v_i)\big) \cup \cdots$$
$$\cup\, BasicClique\big(R \cup \{v_k\}, \{v_k\} \cap N(v_k), (X \cup \{v_1, \dots, v_{k-1}\}) \cap N(v_k)\big)$$
$$= \bigcup_{i=1..k} BasicClique\big(R \cup \{v_i\}, \{v_i, \dots, v_k\} \cap N(v_i), (X \cup \{v_1, \dots, v_{i-1}\}) \cap N(v_i)\big)$$

Basic Bron-Kerbosch algorithm

- BasicClique(R, P, X)
  - If both P and X are empty, report R;
  - If P is empty and X is not empty, report nothing;
  - For each vertex v in P,
    - BasicClique($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
    - $P = P \setminus \{v\}$
    - $X = X \cup \{v\}$



$BasicClique(\emptyset, \{1,2,3,4\}, \emptyset)$

Observe that BasicClique generates every clique exactly once.

$BasicClique(\emptyset, \{1,2,3,4\}, \emptyset)$

$BasicClique(\{1\}, \{1,2,3,4\} \cap N(1), \emptyset \cap N(1))$

$BasicClique(\{1\}, \{2,3,4\}, \emptyset)$

$BasicClique(\{1,2\}, \{3\}, \emptyset)$ → $BasicClique(\{1,2,3\}, \emptyset, \emptyset)$ → $\{1,2,3\}$

$BasicClique(\{1,3\}, \emptyset, \{2\})$ → $\emptyset$

$BasicClique(\{1,4\}, \emptyset, \emptyset)$ → $\{1,4\}$

$BasicClique(\{2\}, \{2,3,4\} \cap N(2), \{1\} \cap N(2))$

$BasicClique(\{2\}, \{3\}, \{1\})$ → $BasicClique(\{2,3\}, \emptyset, \{1\})$ → $\emptyset$

$BasicClique(\{3\}, \{3,4\} \cap N(3), \{1,2\} \cap N(3))$

$BasicClique(\{3\}, \emptyset, \{1,2\})$ → $\emptyset$

$BasicClique(\{4\}, \{4\} \cap N(4), \{1,2,3\} \cap N(4))$

$BasicClique(\{4\}, \emptyset, \{1\})$ → $\emptyset$

Running time of basic Bron-Kerbosch algorithm

- Observe that $BasicClique(\Phi, V, \Phi)$ enumerates all cliques of V.

- The worst case happens when G is a complete graph of n nodes.
- Running time is $\sum_{i=0}^{n} \binom{n}{i} = 2^n$.

Pivot Bron-Kerbosch algorithm

- To further improve the maximal clique finding algorithm, we observe that we don't need to expand R by every node in P.

- **Lemma**: For u in P∪X, any maximal clique must include either u or one of its non-neighbor.

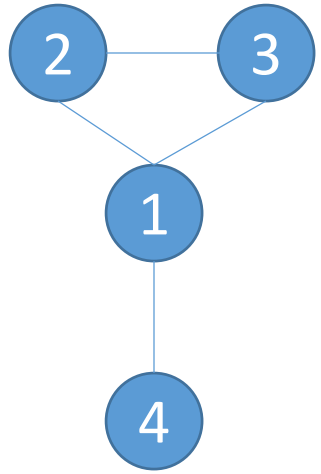
- Proof: By contrary, suppose a maximal clique C does not include both u and any of its non-neighbor.
- Then, C is a subset of N(u).
- This implies C∪{u} is also a clique.
- Hence, C is not maximal and we arrive at contradiction.

Pivot Bron-Kerbosch algorithm

- For a fixed u in $P \cup X$, previous lemma shows that every maximal clique C, which is superset of R, must contain either u or some non-neighbors of u.
- If we don't extend R by $v \in N(u)$, we will not miss any maximal clique!

- PivotClique(R, P, X)
  - If both P and X are empty, report R as a maximal clique;
  - Choose a pivot u in $P \cup X$ which minimizes $P \setminus N(u)$;
  - For each vertex v in $P \setminus N(u)$,
    - PivotClique($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
    - $P = P \setminus \{v\}$
    - $X = X \cup \{v\}$

Illustration of $PivotClique(\emptyset, \{1,2,3,4\}, \emptyset)$

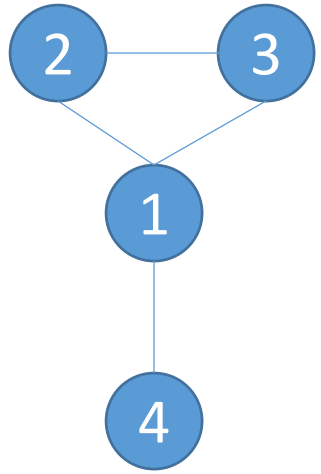


- {1,2,3,4}\N(1)={1}
- {1,2,3,4}\N(2)={2,4}
- {1,2,3,4}\N(3)={3,4}
- {1,2,3,4}\N(4)={2,3,4}
- We choose 1 as pivot.
- Hence, we only expand R with v in {1,2,3,4}\N(1)={1}.

$PivotClique(\{1\}, \{1,2,3,4\} \cap N(1), \emptyset \cap N(1))$

$PivotClique(\emptyset, \{1,2,3,4\}, \emptyset) \longrightarrow PivotClique(\{1\}, \{2,3,4\}, \emptyset)$

Illustration of $PivotClique(\emptyset, \{1,2,3,4\}, \emptyset)$



- {2,3,4}\N(2)={2,4}
- {2,3,4}\N(3)={3,4}
- {2,3,4}\N(4)={2,3,4}
- We choose 2 as pivot.
- Hence, we only expand R with v in {2,3,4}\N(2)={2,4}.

$PivotClique(\{1,2\}, \{2,3,4\} \cap N(2), \emptyset \cap N(2))$

$PivotClique(\emptyset, \{1,2,3,4\}, \emptyset) \rightarrow PivotClique(\{1\}, \{2,3,4\}, \emptyset)$

$PivotClique(\{1,2\}, \{3\}, \emptyset) \rightarrow PivotClique(\{1,2,3\}, \emptyset, \emptyset)$

$PivotClique(\{1,4\}, \emptyset, \emptyset)$

$PivotClique(\{1,4\}, \{2,3,4\} \cap N(4), \{2\} \cap N(4))$

Running time

- PivotClique(Ø, V, Ø) runs in $O(3^{n/3})$ time. (Analysis is not shown!)

- This is optimal as a function of n since a graph with n vertices can have up to $3^{n/3}$ maximal cliques.

Compatibility for characters with k possible states

- We can generalize the problem when the characters are not binary
- Definition:
  - A character c with k possible states is compatible to a leaf-labeled tree T if and only if there exist an assignment of states to the internal nodes of T such that the total number of state changes is exactly k-1
- Result:
  - Compatibility Problem
    - When the number of states is constant, polynomial time algorithm is still feasible
    - When the number of states is variable, NP-complete
  - Large Compatibility Problem
    - NP-complete

Maximum Likelihood

- Given a character-state matrix M, maximum likelihood tries to find a model T such that
  - Pr(M|𝒯) is maximized!

What is a model?

- A model consists of
  - A rooted tree which models the evolution relationship
  - Every edge is associated with a stochastic model of evolution
- Usually, it is assume that
  - the characters evolve identically and independently
  - Also, the tree has the markov property. That is, the evolution occurs at one subtree is independent to the other parts of the tree.
- Example of models:
  - Cavender-Felsenstein model (also called Cavender-Farris model)
  - Jukes-Cantor model

Cavender-Felsenstein Model

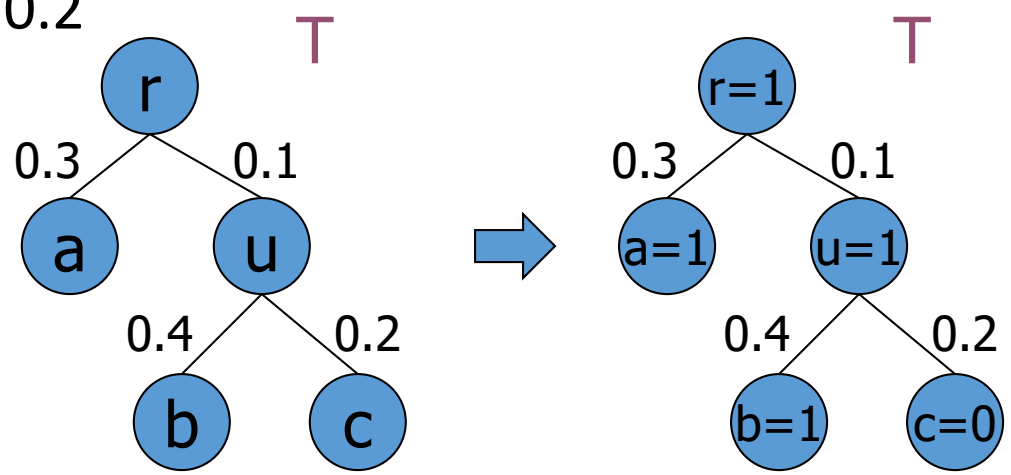
- Simplest possible markov model of evolution
- Assume each character has only two states
- The model $\mathcal{T}$ consist of
  - the topology T
  - a mutation probability p(e) for each edge e in T
- Assumption:
  - For every e=(u,v) in T, $0<p(e)<0.5$

	u=0	u=1
v=0	Pr(u=0|v=0)=1-p(e)	Pr(u=1|v=0)=p(e)
v=1	Pr(u=0|v=1)=p(e)	Pr(u=1|v=1)=1-p(e)

  - Pr(u|v) = Pr(v|u)
  - For the root r, Pr(r=0)=Pr(r=1)=0.5

Example of a CF model

- Consider 3 species a, b, and c.
- Below is a CF model $\mathcal{T}$=(T,p).
- a, b, c are observed data and r,u are unobserved data.
- For a particular character i, suppose we have: a_i=1, b_i=1, c_i=0, r_i=1, u_i=1.
- $Pr(a_i=1, b_i=1, c_i=0, r_i=1, u_i=1 | \mathcal{T})$
  $= Pr(r_i=1)\ Pr(a_i=1|r_i=1)\ Pr(u_i=1|r_i=1)\ Pr(b_i=1|u_i=1)\ Pr(c_i=0|u_i=1)$
  $=0.5 * (1-0.3) * (1-0.1) * (1-0.4) * 0.2$
  $=0.0378$

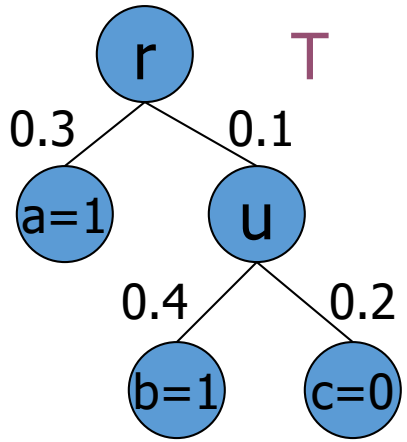


Example of a CF model

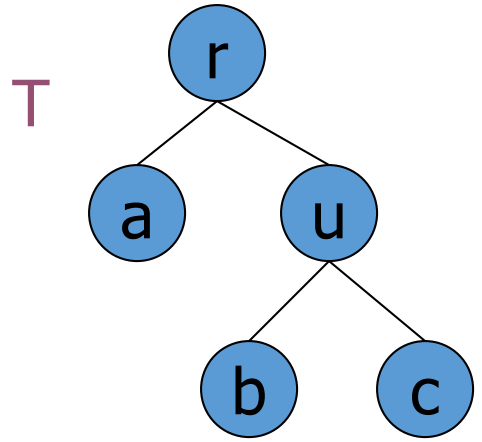
- We want to compute $Pr(a_i=1, b_i=1, c_i=0|\mathcal{G})$

- $Pr(a_i=1, b_i=1, c_i=0, r_i=1, u_i=1|\mathcal{G}) = 0.5 * (1-0.3) * (1-0.1) * (1-0.4) * 0.2 = 0.0378$
- $Pr(a_i=1, b_i=1, c_i=0, r_i=1, u_i=0|\mathcal{G}) = 0.5 * (1-0.3) * 0.1 * 0.4 * (1-0.2) = 0.0112$
- $Pr(a_i=1, b_i=1, c_i=0, r_i=0, u_i=1|\mathcal{G}) = 0.5 * 0.3 * 0.1 * (1-0.4) * 0.2 = 0.0018$
- $Pr(a_i=1, b_i=1, c_i=0, r_i=0, u_i=0|\mathcal{G}) = 0.5 * 0.3 * (1-0.1) * 0.4 * (1-0.2) = 0.0432$

- $Pr(a_i=1, b_i=1, c_i=0|\mathcal{G})$
  $= 0.0378 + 0.0112 + 0.0018 + 0.0432$
  $= 0.094$



Cavender-Felsenstein Model



- Consider 3 species a, b, and c.
- Assume the model says that the tree topology is T and the mutation probability for every edge e is p(e)
- Suppose the data M_i says: a_i=x, b_i=y, c_i=z
- Then, probability that the data is M_i given that the model is (T, p), $\Pr(M_i|T,p)$, equals

$$\sum_{\substack{k=0,1 \\ j=0,1}} \Pr(r_i = k) \Pr(a_i = x|r_i = k) \Pr(u_i = j|r_i = k) \Pr(b_i = y|u_i = j) \Pr(c_i = z|u_i = j)$$

Cavender-Felsenstein Model

- Consider n species each is characterized by m characters
- Let the data be $M=M_1 \cup \ldots \cup M_m$
- The model consists of the tree topology T and the mutation probability p.

$$\Pr(M|T,p) = \prod_{i=1}^{m} \Pr(M_i|T,p)$$

Computational Problems

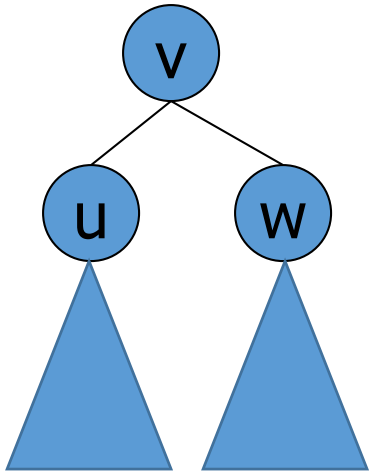
- Likelihood of a model
  - Given the model $\mathcal{G}$, for any data M, try to compute Pr(M|$\mathcal{G}$)
- Find model with maximum likelihood
  - Given data M, try to find a model $\mathcal{G}$ which maximizes Pr(M|$\mathcal{G}$)!

Likelihood of a model

- Input:
  - Data M: n species where each species is characterized by m character
  - Model $\mathcal{T}$=(T, p)
- Aim: Compute Pr(M|$\mathcal{T}$)

- Pr(M|$\mathcal{T}$) can be computed using the formula we stated before.
  - However, it takes exponential time.
- Can we do it better?
  - Yes! By defining the likelihood recursively and compute the value using dynamic programming.

Recursive Definition

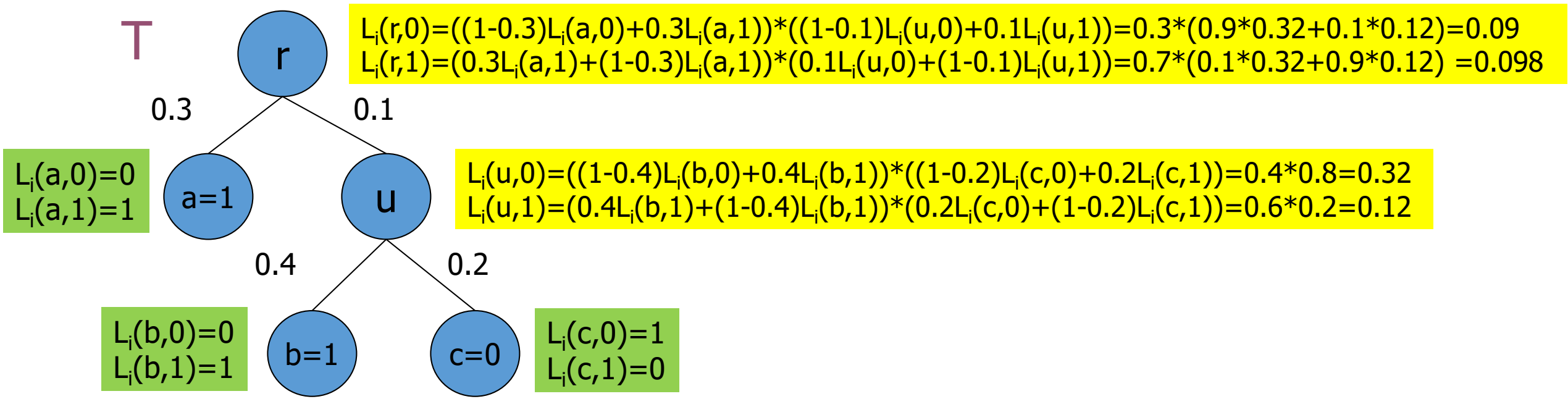


- For a particular character i, let $L_i(v, s)$ be the likelihood of the subtree rooted at v, given that character i has state s.
- Base case: For every leaf v and state s,
  - $L_i(v, s) = 1$ if v_i=s; 0,otherwise ---- (1)
- Recursive case: for every internal node v with children, says, u and w,

$$L_i(v,s) = \left[\sum_{y=0,1} L_i(u,s)\Pr(u_i = y | v_i = s)\right]\left[\sum_{y=0,1} L_i(w,y)\Pr(w_i = y | v_i = s)\right] \text{ --- (2)}$$

Example

$$\Pr(b_i = 1, b_i = 1, c_i = 0|\mathfrak{T}) = \frac{1}{2}\left(L_i(r,0) + L_i(r,1)\right) = \frac{1}{2}(0.09 + 0.098) = 0.094$$



Algorithm

Algorithm ComputeLikelihood(i, u)
  if u is a leaf **then**
    Compute and return $\{L_i(u,0), L_i(u,1)\}$ using Equation (1);
  else
    Let v and w be the two children of u;
    Call ComputeLikelihood(i, v) to compute $\{L_i(v,0), L_i(v,1)\}$;
    Call ComputeLikelihood(i, w) to compute $\{L_i(w,0), L_i(w,1)\}$;
    Compute and return $\{L_i(u,0), L_i(u,1)\}$ using Equation (2);
  end if

- Finally, for the root, we have

$$L = \prod_{i=1}^{m} \left[\frac{1}{2} L_i(root, 0) + \frac{1}{2} L_i(root, 1) \right]$$

Time complexity

- For every node v and every state s,
  - $L_i(v,s)$ can be computed in O(1) time according to the recurrence.
- Since there are n nodes and m characters, all $L_i(v,s)$ can be computed in O(mn) time.
- For L, it can be computed in O(m) time.

- In total, Likelihood of a tree can be computed in O(mn) time.

Find model using maximum likelihood

- Input:
  - Data M: n species where each species is characterized by m character
- Aim: Find $\mathcal{G}$=(T, p) which maximizes Pr(M|$\mathcal{G}$)

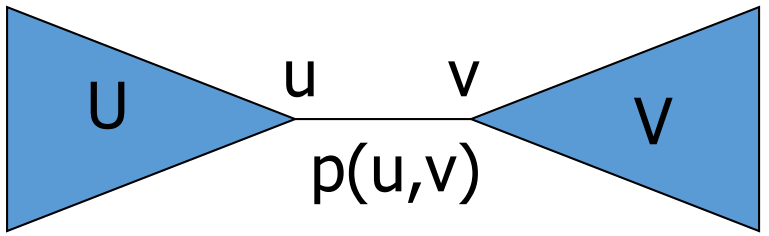
- This problem is NP-hard.
- Solution: uses heuristic to get close to optima (like DNAml)

Estimating the weight of an edge

- For h, h' $\in$ {0, 1}, let $L_i(U, u = h)$ and $L_i(V, v = h')$ be the maximum likelihood score of U and V with the i[th] state of u and v equals h and h' respectively.
- We would like to find p(u,v) of the edge (u,v) which maximizes the likelihood of the combined tree.
- Note that the likelihood of the combined tree is

$$L = \prod_i \sum_{h,h' \in \{0,1\}} L_i(U,h) L_i(V,h') \Pr(u_i = h | v_i = h')$$

- We would like to find p(u,v) which maximizes L.



Lemma: $\frac{d \ln L}{dp}$ is decreasing.

$$L = \prod_i \sum_{h,h' \in \{0,1\}} L_i(U,h) L_i(V,h') \Pr(u_i = h | v_i = h')$$

$$= \prod_i p(u,v) \left(\sum_{h, \in \{0,1\}} L_i(U,h) L_i(V, 1-h) \right) + \big(1 - p(u,v)\big) \left(\sum_{h, \in \{0,1\}} L_i(U,h) L_i(V,h) \right)$$

$$= \prod_i p(u,v) A_i + \big(1 - p(u,v)\big) B_i$$

$$\ln L = \sum_i pA_i + (1-p) B_i$$

$$\frac{d \ln L}{d p} = \sum_i \frac{A_i - B_i}{pA_i + (1-p) B_i} = 0$$

Since $\frac{d^2 \ln L}{dp^2} = -\sum_i \frac{(A_i - B_i)^2}{(pA_i + (1-p)B_i)^2} < 0$, $\frac{d \ln L}{dp}$ is decreasing.

Find p(u,v) that maximizes L

- To find p(u,v) that maximizes L, we need to find p(u, v) such that $\frac{d \ln L}{dp} = 0.$
- As 0<p<1, we can identify p such that $\frac{d \ln L}{dp} = 0$ by binary search.

DNAml

Algorithm DNAml

- Let S = {s_1, s_2, …, s_n} be the set of taxa.
- Build the tree T for species {s_1, s_2}
- For k = 3 to n
  - Among all (2k-5) ways to insert s_k into T,
    - we choose the way with the best likelihood.
  - If k>= 4,
    - While there exists NNI which can improve the likelihood of T,
      - We perform such NNI

Final remark for Maximum Likelihood

- For the Cavender-Felsenstein model, maximum likelihood is statistically consistent.