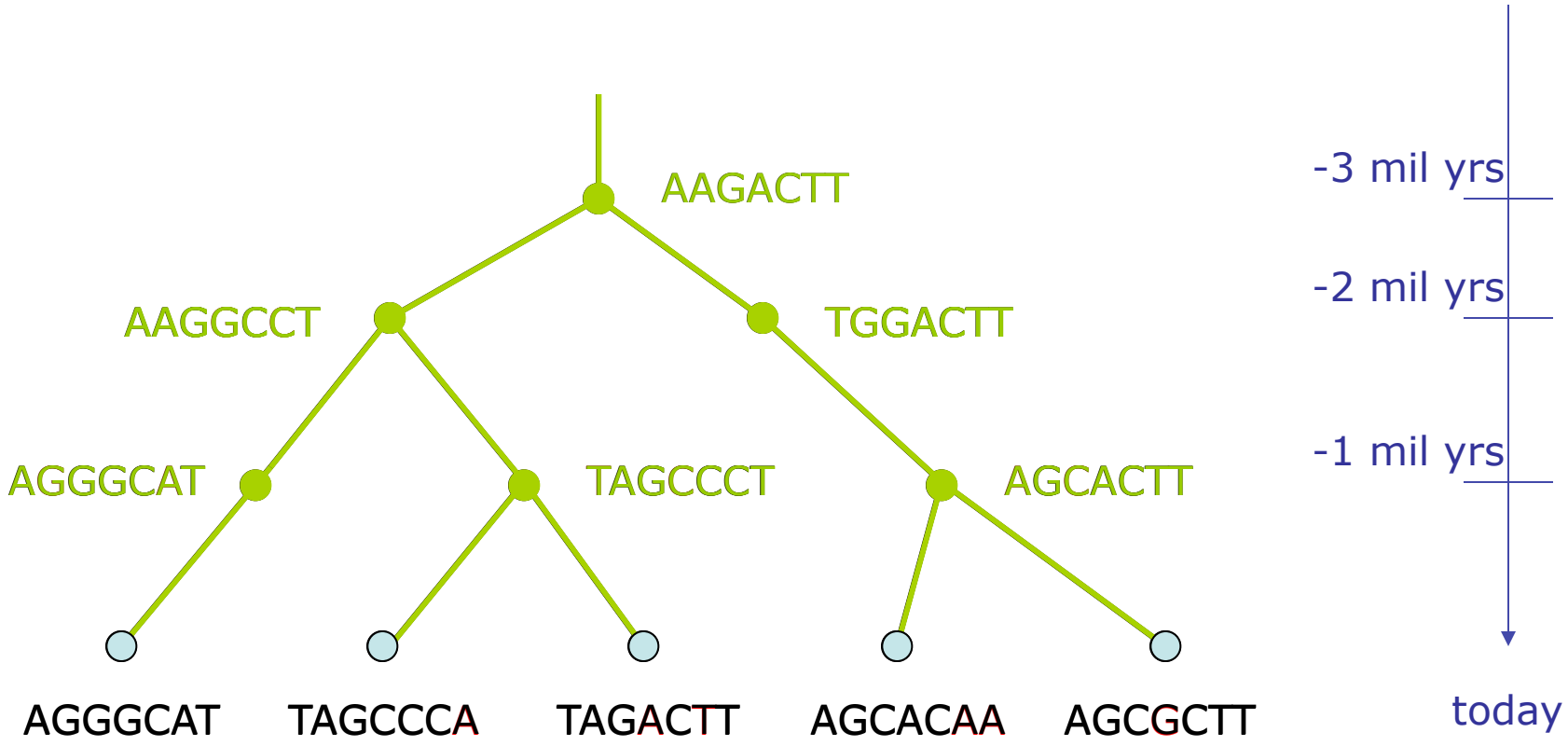
598AGB

Basics

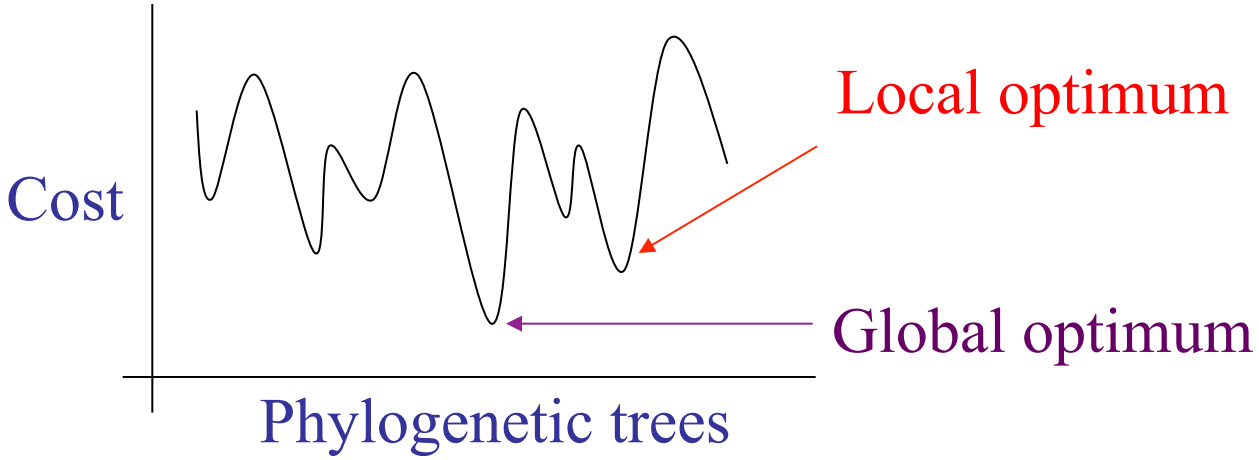
Tandy Warnow

DNA Sequence Evolution



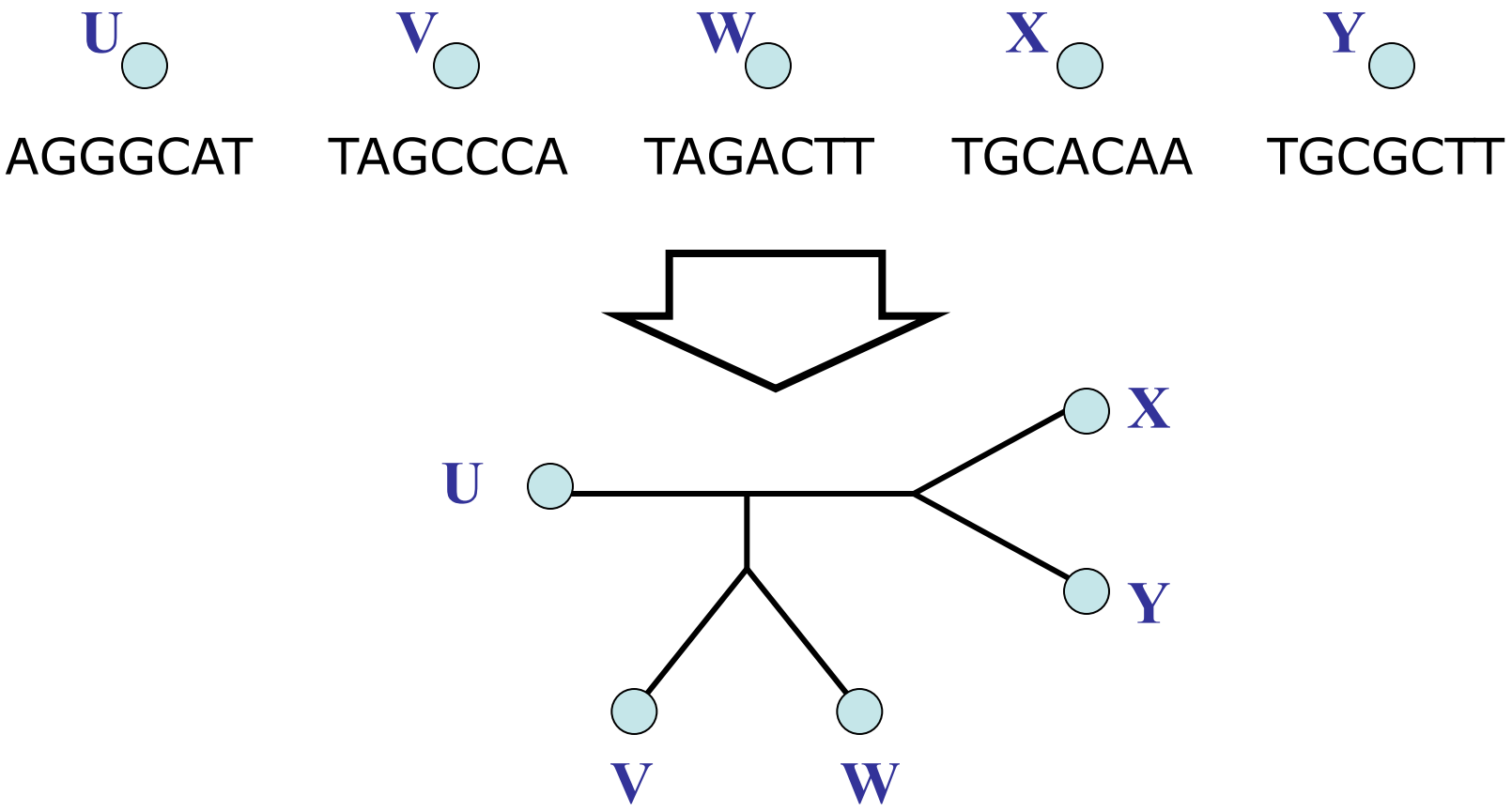
Phylogenetic reconstruction methods

1. Heuristics for NP-hard optimization criteria (Maximum Parsimony and Maximum Likelihood)



2. Polynomial time distance-based methods: Neighbor Joining, FastME, etc.
3. Bayesian MCMC methods.

Phylogeny Problem



Today's material

- We will see some basic techniques for representing trees (whether rooted or unrooted), and for constructing trees from different types of input data (induced subtrees or pairwise distances).
- This material will allow us to understand proofs of statistical identifiability of trees under stochastic models of evolution, and statistical consistency of methods under these models, which are the main subjects of this course.
- These techniques and theorems will also provide insights into computational complexity issues for different approaches to tree estimation.

Today's Material (Details)

- Newick notation for rooted and unrooted trees
- Constructing rooted trees from their clades and triplet trees (Aho, Sagiv, Szymanski, and Ullman)
- Constructing unrooted trees from their bipartitions and quartet trees (All Quartets Method)
- Constructing trees from additive matrices (Naïve Quartet Method)
- Comparing two trees (and quantifying error)
- Testing compatibility of trees

Newick representations

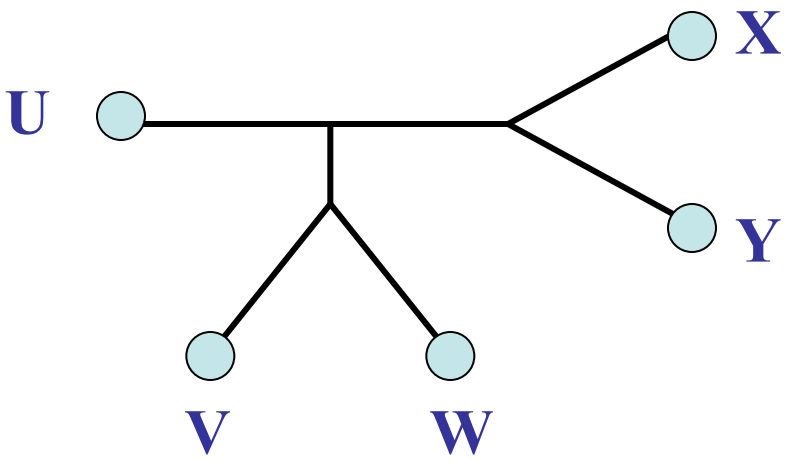
- For a rooted tree, we represent a graph with a string with the taxa, commas, and nested parentheses.
- For example, what tree is represented by (a,(b,(c,((d,e),(f,g))))))?
- How do we represent an unrooted tree? (Easy - root it somewhere, and write down the Newick representation of the rooted version.)

(U,((V,W),(X,Y)))

or

((X,Y),(U,(V,W))

or …



Rooted vs. unrooted

- Task: be able to move between rooted and unrooted representations of trees
- Task: be able to compare two trees and see if they are different or the same

Clades

- Definition: Let T be a *rooted* tree leaf-labelled by S, let v an internal node in T, and let $\mathbf{X_v}$ be the set of leaves in T below v. Let

  Clades(T) = {X_v: v in V(T)}.

  Note: X_v is also called the "cluster" at node v, so this is sometimes called Clusters(T).

- Question: Given Clades(T), can we compute T?

Bipartitions

- Given an edge e in a leaf-labelled unrooted tree T, the removal of the edge e (but not its endpoints) defines a bipartition on the leaves of the tree T. We denote by $\mathbf{c_e}$ the bipartition defined by the edge e. We let

  $\mathbf{C(T)=\{c_e: e \text{ in } E(T)\}}$.

- Questions: Given C(T), can we compute T?

Quartet subtrees

- Given tree T leaf-labelled by S, and quartet a,b,c,d of leaves, we let T|{a,b,c,d} denote the minimal homeomorphic subtree of T restricted to {a,b,c,d}. We let **Q(T)** denote {T|X: X is a four taxon subset of S}.

- Question: Given Q(T), can we compute T?

Computing trees

- Given Q(T) (the quartet subtrees of T), can we determine T?
- Given C(T) (the bipartitions of S defined by the edges of T), can we determine T?
- Given Clades(T) (the sets of leaves defined by internal nodes in the rooted tree T), can we determine T?

Quartet-based reconstruction

- Definition: Let T be a tree leaf-labelled by a set S, and let Q(T) be the set of quartet subtrees of T (derived from each of the four-taxon subsets of S).

Question: can we reconstruct T from Q(T)?

Computing T from Q(T): All Quartet Method

- Given Q(T):
  - Find a sibling pair A, B (a pair of leaves which are always together in every quartet in which they both appear)
  - Compute the tree T' for S-{A} by recursing on the subset of Q(T) that doesn't include taxon A
  - Insert A into T' by making A sibling to B, and return the tree obtained

Analysis of the algorithm

Questions:

- Accuracy?
- Running time?
- But: *how are we to compute quartet subtrees?*

Clade compatibility

- Definition: Let T be a rooted tree leaf-labelled by S, v an internal node in T, and X_v the leaves in T below v. Let Clades(T)=$\{X_v: v \text{ in } V(T)\}$.
- Theorem: Let X be a set of subsets of S. Then there exists a tree T leaf-labelled by S such that X = Clades(T) if and only if for all A, B in X, either A and B are disjoint, or one contains the other.

Proof of the theorem

- One direction is easy
- The other direction is a proof by construction!

Computing rooted trees from clades

- Partially order the set of clades by containment, add in the full set S, and compute the Hasse Diagram of the resultant poset

Tree construction from clades

Questions:

- Accuracy?
- Running time?
- But, *how are we to compute clades?*

Bipartition compatibility

- Definition: Let X be a set of bipartitions on a set S. Then X is said to be **compatible** if there exists a tree T leaf-labelled by S such that X = C(T), where C(T) = $\{c_e: e \text{ in } E(T)\}$.

Question: Can we construct the tree T from C(T)?

Computing trees from bipartitions

Given the set of bipartitions on the leaf-set induced by the edges of a tree T, how can we compute the tree T?

Hint: "root" the tree T by picking it up at a leaf, and then consider the set of bipartitions as a set of "clades", and apply the previous algorithm. (Note: the choice of leaf does not matter!)

Computing trees from bipartitions

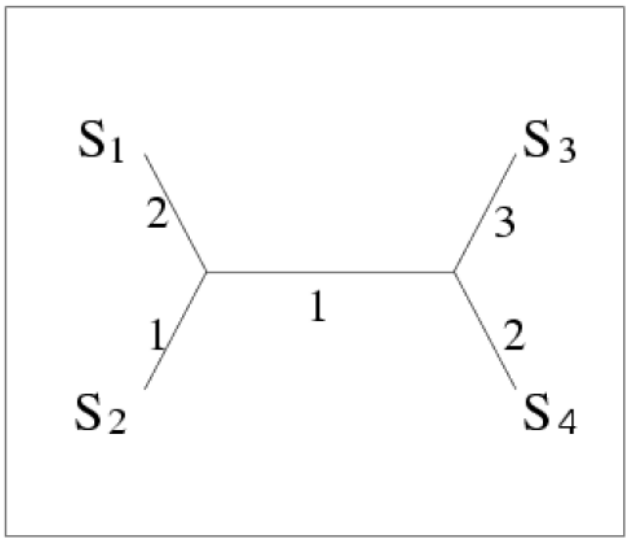
- Question:

  How are we to obtain bipartitions?

Additive Distance Matrices

	S_1	S_2	S_3	S_4
S_1	0	3	6	5
S_2		0	5	4
S_3			0	5
S_4				0

POLYTIME
INVERTIBLE



Four-point condition

Theorem (Buneman and others): A matrix D is additive if and only if for every four indices i,j,k,l, the maximum and median of the three pairwise sums are identical

$$D_{ij}+D_{kl} < D_{ik}+D_{jl} = D_{il}+D_{jk}$$

Proof: one direction is easy. The other direction requires some work!

Four-point method

- The Four-Point Method computes trees on quartets using the ideas in the Four-point condition
- Given a "dissimilarity" matrix D (may not satisfy the triangle inequality, but will be symmetric and zero on the diagonal), we compute a tree on four leaves s_i, s_j, s_k, s_l as follows:
- If

$D_{ij}+D_{kl}$ is less than both $D_{ik}+D_{jl}$, and $D_{il}+D_{jk}$

then set the tree to be $((s_i,s_j),(s_k,s_l))$.

Naïve Quartet Method

- Given additive distance matrix D, we compute a tree T by:
  - Using the Four Point Method to compute a quartet tree on every set of four leaves
  - Combine the quartet trees into a tree on the entire set of leaves using the All Quartets Method

Naïve Quartet Method

- Easy to see it is correct and polynomial time.
- But what if the distance matrix isn't additive? We try the algorithm anyway. If the All Quartets Method returns a tree, then we're happy. Otherwise we return "Fail".

Comparing two trees using bipartition sets

- To see if two trees T and T' are the same, write down C(T) and C(T') and see if they are the same set.

- When computing the error in an estimated tree T with respect to a true tree T*, we set
  - C(T)-C(T*) = false positives, and
  - C(T*)-C(T) = false negatives (missing branches)

Consensus Trees

- Given a set of trees, we compute consensus trees to represent what they have in common. For example:
  - The strict consensus
  - The majority consensus
  - The maximum agreement subtree
  - The maximum compatible subset tree
- Not all of these problems are solvable in polynomial time.

Compatibility trees

- The compatibility tree of a set of trees (all on the same set of leaves) is the minimal common refinement, if it exists.
- Determining if a set of trees (all on the same set of leaves) have a common refinement is solvable in polynomial time.

Testing compatibility

- Suppose we have a set X of unrooted trees, and they are not all on the same leaf set. For example, X could be a set unrooted quartet trees, each on a different set of four leaves.
- We want to know if there is a tree T that agrees with all the trees in X. This is the "tree compatibility" problem.

Unrooted Tree Compatibility

- Input: Set X of unrooted trees, not all on the same set of leaves.
- Output: Tree T (if it exists) that agrees with all the trees in X, and otherwise “Fail”

This problem is NP-hard. However, some special cases can be solved in polynomial time.

Rooted Tree Compatibility

- Input: Set X of rooted trees, not all on the same set of leaves.
- Output: Tree T (if it exists) that agrees with all the trees in X, and otherwise “Fail”

This problem is solvable in polynomial time.

Proof: recall the Aho, Sagiv, Szymanski, and Ullman algorithm!

Summary so far

- We have seen some basic techniques for representing trees (whether rooted or unrooted), and for constructing trees from different types of input data (induced subtrees or pairwise distances).
- This material will allow us to understand proofs of statistical identifiability of trees under stochastic models of evolution, and statistical consistency of methods under these models, which are the main subjects of this course.
- These techniques and theorems also provide insights into computational complexity issues for different approaches to tree estimation.