Lecture notes on phylogenetics

Usman Roshan
Department of Computer Science
New Jersey Institute of Technology

February 15, 2005

1 Basics

1.1 Phylogeny

A *phylogenetic tree* is a tree whose leaves are labeled by current species. Phylogenies are usually binary and unrooted.

1.2 Bipartitions

Every edge e in a phylogeny T defines a bipartition π_e on $L(T)$ (leaves of T) induced by removing e. A tree T is uniquely identified by its set of bipartitions. Bipartitions defined by external edges are trivial and included in all trees. Therefore, we will only refer to bipartitions induced by internal edges.

Bipartitions can also be used to identify topological differences between two trees. Given two trees T and T', the *false positive* of T' with respect to

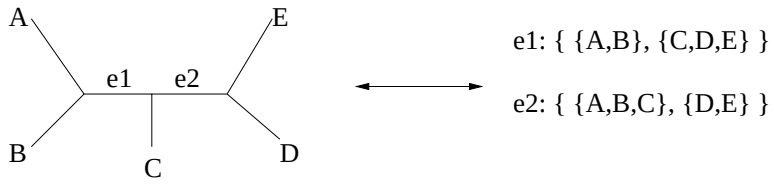


Figure 1: Tree and its set of bipartitions

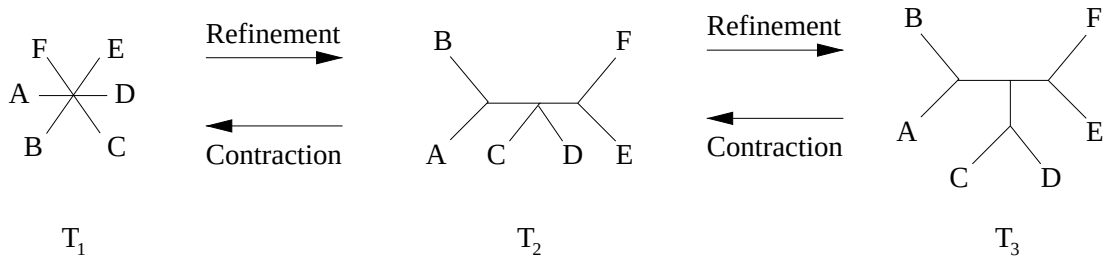


Figure 2: T_1 is a star and can be refined to obtain T_2, a partially resolved tree; T_2 can be further refined to obtain T_3, a fully resolved tree

T is the number of bipartitions in T' that are not in T. The *false negative* of T' is the number of bipartitions in T that are not in T'.

1.3 Compatibility

A set of bipartitions C is said to be compatible if there exists a tree T defined by C. For example $\{\{A, B\}, \{C, D\}\}$ and $\{\{A, C\}, \{B, D\}\}$ are incompatible because there is no tree on the leaf-set $\{A, B, C, D\}$ containing these two bipartitions.

1.4 Strict consensus

The strict consensus of a set of trees $\mathcal{T}$ is the set of bipartitions $\cap_{forall T \in \mathcal{T}} C(T)$, i.e., the intersection of the set of bipartitions of all trees in $\mathcal{T}$. This strict consensus is clearly compatible and therefore a valid tree.

1.5 Tree resolution

Let $C(T)$ denote the set of bipartitions of T. The *degree of resolution* of a tree T is $\frac{|E_I(T)|}{|L(T)-3|}$, i.e., the number of internal edges in T divided by the number of internal edges in an unrooted binary tree with n leaves. Given two trees T and T' we say that T is a *contraction* of T' if $C(T) \subset C(T')$. We also say that T' is a *refinement* of T. A tree with no internal edges is called a *star*, and a tree with all the internal edges (a binary tree) is called a *fully resolved* tree; it has $n - 3$ internal edges where n is the number of leaves. A tree with the number of edges between a star and a binary tree is called a *partially resolved* tree.

1.6 Tree restriction

We say that $T_{|X}$, where $X \subseteq L(T)$, is the tree restricted to the set of taxa X. It is obtained by restricting the leaf set of T to X and suppressing nodes of degree 2 (see Figure 3).

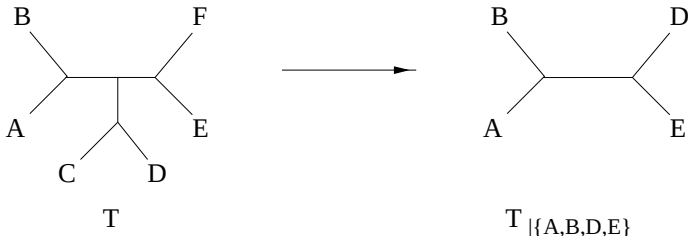


Figure 3: Tree T restricted to leaf set $\{A, B, D, E\}$.

2 Models

DNA nucleotides evolve on a tree according to a stochastic process. We will focus only on independent and identical site evolution (i.i.d) and therefore it suffices to discuss the evolution of a single site of a DNA sequence.

2.1 Poisson process—evolution on a single edge

The number of changes of a nucleotide on any given edge is modeled as a Poisson process. This means that

1. the number of changes in a fixed time interval Δt is independent of the number of changes in any other non-overlapping time interval Δu,
2. the number of changes is time interval Δt is proportional to the Δt, the length of the time interval, and
3. there are no changes in a time interval of length 0.

For a moment assume that we are working with an alphabet of $\{0, 1\}$ instead of DNA which is $\{A, C, G, T\}$. This means we are modeling the evolution of zero one sequences instead of DNA.

If X is the number of changes and follows a Poisson process, then probability dictates that $P(X = k) = \frac{\lambda^k}{k!}e^{-\lambda}$, where $P(X = k)$ is the probability that the number of nucleotide changes X is k and λ is the expected value of X, i.e., the expected number of changes. Since only observed changes can be accounted for the probability of any change on edge e is $p_e = \sum_{oddk} P(X = k) = \frac{1-e^{-2\lambda}}{2}$. This means that the expected number of changes on any edge $\lambda_e = -\frac{1}{2}\ln(1 - 2p_e)$.

2.2 Markov Model—evolution of DNA

In general a probability substitution matrix is used for each edge when we have more than two states. Under two states the matrix is

	0	1
0	$1 - p_e$	p_e
1	p_e	$1 - p_e$

The matrix is determined basically by one parameter p_e. The substitution matrix M under the *General Markov Model* for DNA is

	A	C	G	T
A	p_{AA}	p_{AC}	p_{AG}	p_{AT}
C	p_{CA}	p_{CC}	p_{CG}	p_{CT}
G	p_{GA}	p_{GC}	p_{GG}	p_{GT}
T	p_{TA}	p_{TC}	p_{TG}	p_{TT}

Each row must sum to one since this is a probability matrix. We usually work with reversible models where we assume that $p_{ij} = p_{ji}$. This gives us the *General Time Reversible* model. We can get even simpler models by further restricting the values in M. A popular model is the *Jukes-Cantor* (JC) model whose probability matrix is

	A	C	G	T
A	$1 - p_e$	$\frac{p_e}{3}$	$\frac{p_e}{3}$	$\frac{p_e}{3}$
C	$\frac{p_e}{3}$	$1 - p_e$	$\frac{p_e}{3}$	$\frac{p_e}{3}$
G	$\frac{p_e}{3}$	$\frac{p_e}{3}$	$1 - p_e$	$\frac{p_e}{3}$
T	$\frac{p_3}{3}$	$\frac{p_e}{3}$	$\frac{p_e}{3}$	$1 - p_e$

Note that the JC model can also be captured by just one parameter p_e. Under JC the expected number of changes is $\lambda_e = -\frac{3}{4}\ln(1 - \frac{4}{3}p_e)$.

2.3 Evolution on a path—set of edges

We have just defined the probability DNA and zero-one states on a single edge of a tree. What can we say about the probability of change on a given path? Assume we have a path of k edges and that $p_1, p_2, ..., p_k$ are the probabilities of change on each edge of the path. Using induction we can show that the probability of a change on the path is $p = frac12\left(1 - \prod_{i=1}^{k}(1-2p_i)\right)$. However, the expected number of changes on the path p is $\lambda = -\frac{1}{2}\ln(1-2p) = \sum_{i=1}^{k}\lambda_e$, a sum of the expected number of changes on each edge. This brings us to additive matrices.

3 Distance-based reconstruction

3.1 Additivity

A matrix D is additive if there exists a tree T with positive edge lengths such that $D_{ij} = \sum_{e\in}$ path from $_i$ to $_j\, w(e)$, where $w(e)$ is the edge length of edge e. The four-point condition allows us to determine if a matrix is additive: a matrix D is additive if and only if $\forall i, j, k, l$ the maximum of $D_{ij}+D_{kl}, D_{ik}+D_{jl}, D_{il}+D_{jk}$ is not unique. Thus, if a matrix is additive, we can compute all the *quartets*, four-leaf trees, and construct the full tree using a naive quartet algorithm. We call this the naive quartet approach. Note that we can only construct trees for additive matrices with this approach. If not additive we can get conflicting quartets.

3.2 NJ and UPGMA

UPGMA and neighbor joining are two distance-based methods which will always return a tree. UPGMA is not additive while NJ is.

3.3 Estimation

In order to construct trees using a distance-based method we need to estimate a distance matrix from a set of sequences. We first must select a model and then estimate distances under that model. We will work with the JC model. In our distance matrix d_{ij} will represent the expected number of changes from sequence i to j, and not the probability of change. This is so that we can then apply concepts of additivity and other distance methods to our matrix.

Recall that under the JC model $\lambda_e = -\frac{3}{4}\ln(1 - \frac{4}{3}p_e)$. Thus to estimate λ_e we need to estimate p_e. For two given sequences S, T we estimate p_e as the normalized Hamming distance between S and T. This is actually a maximum likelihood estimate of p_e since we are assuming *i.i.d.* evolution.

3.4 Error tolerance

It's possible that our distance estimate is poor, especially when sequences are short and evolutionary rates are high. It's the same as trying to estimate the probability of heads of a coin toss from very few coin tosses and the probability of heads is high. Our estimated matrix d may still be additive but not exactly the same as the matrix for the "true tree".

However, we can still reconstruct the true tree if the error in our matrix is not too high. Let T be the true tree (which we cannot see) and D be the additive matrix for T. Let x be the smallest edge length in T. Let d be our estimated matrix from a set of sequences S evolved on T under the Jukes-Cantor model, and let T' be the tree one gets from d. If $L_\infty(D, d) = \max_{ij} |D_{ij} - d_{ij}| < \frac{x}{2}$ then the trees T and T' are identical, i.e., their set of bipartitions is exactly the same.

4 Simulation studies

In practice the true evolutionary tree is never known; therefore, we cannot determine how well our algorithm performs in terms of recovering the true tree. However, in simulation studies the true tree is known which allows us to compare the reconstructed tree to the true one. Simulation studies, thus, allow us to study the performance of methods in practice.

4.0.1 Steps of a simulation study

1. We first decide on a stochastic model M which will govern the evolution of DNA sequences on a model tree T. For example, we may select the Jukes-Cantor model.

2. We now decide on a class of model tree topologies $\mathcal{T}$, such as uniform trees or birth-death trees. We randomly select trees from this class to produce m model trees.

3. For each of the m model trees, we evolve sequences of length k under the model M; this produces a set of aligned sequences. Note that we are only working with substitution models; therefore there are no insertions and deletions. Using a phylogenetic reconstruction algorithm A, we then construct a tree on each set of sequences to obtain a set of inferred trees.

4. For each set of sequences, we compare the topology of the inferred tree to that of the model tree. We use the false positive and the false negative for comparing trees. We average the false positives of the inferred trees and do the same for false negatives; from this, we can determine the average performance of algorithm A over various tree topologies from $\mathcal{T}$ under the model M.

5 Maximum Likelihood (ML)

The input to the ML problem is a set of aligned sequences S and a model M. The output is the tree T_{M_e} (with substitution matrices for each edge) that maximizes $P(S|T_{M_e})$, the probability of the data given the tree, under the model M. This problem has two parts:

1. finding the likelihood score of a fixed tree T (with no probability substitution matrices for all edges)

2. finding the tree with the maximum likelihood score.

We call the first problem the *fixed point estimation* problem. There is no known exact algorithm for it and therefore it has unknown complexity. However, the ML score of a tree with probability substitution matrices edge lengths can be computed in polynomial time using a dynamic programming approach.

6 Maximum Parsimony (MP)

The input to the MP problem is a set of aligned sequences S. The output is a tree T and an assignment of sequences to its internal nodes such that the *parsimony length* of T is minimized. Given an assignment of sequences to the internal nodes of tree T, the parsimony length of T is $\sum_{(a,b)\in E(T)} H(a,b)$

where $E(T)$ is the edge-set of T and $H(a, b)$ is the Hamming distance between the sequences a and b. This problem is NP-hard.

However, the problem of finding the MP score of a given tree can be solved in polynomial time using a dynamic programming approach. In this problem one just has to assign sequences to the internal nodes of T so as to minimize the parsimony length of T.

7 Heuristics for MP and ML—Iterative improvement

Since both MP and ML are very hard to solve exactly in practice, we use heuristics. A popular heuristic is hill-climbing or iterative improvement. Assume we are trying to find the minima of the function $g(s)$.

- Determine a candidate solution s
- While s is not a local minima
  - Find a *neighbor* s' of s such that $g(s') < g(s)$
  - If found then set s=s'

We can apply this kind of search for solving MP or ML. A candidate solution is usually a randomized greedy tree. This is constructed by first randomly ordering the sequences. A tree (trivial star) is constructed on the first three sequences and the remaining are then inserted, one by one, so as to minimize the MP (or ML) scores.

There are three different ways to select a neighbor of a tree T.

7.1 Nearest Neighborhood Interchange (NNI)

We illustrate the NNI move using Figure 4. In the first move subtrees A and D interchange, and in the second move subtrees A and C interchange. For each internal edge there are two possible moves, and since there are $n - 3$ internal edges, the neighborhood size of NNI is $2n - 6$.

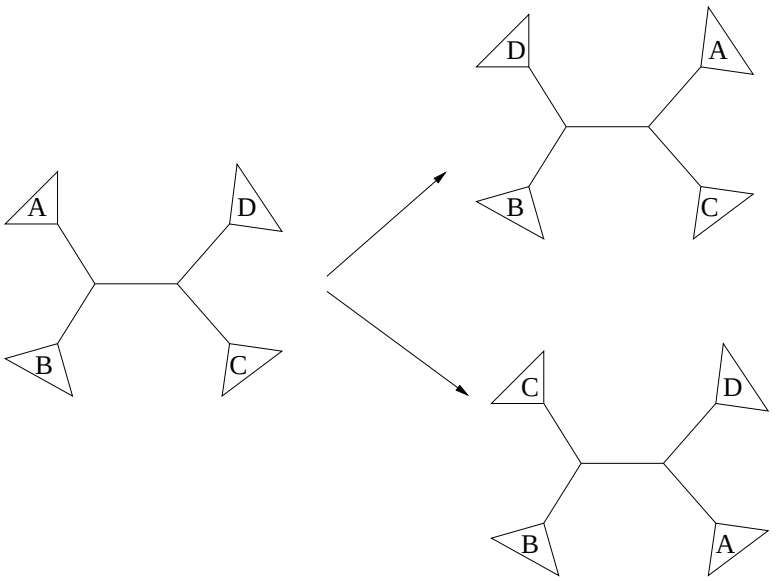


Figure 4: The two possible NNI moves from edge e

7.2 Subtree Prune and Regraft (SPR) and Tree Bisection and Reconnection (TBR)

We illustrate the SPR and TBR moves by example using Figures 5 and 6. The edge e is bisected which gives rise to two subtrees, $t1$ containing leaves A, B, E, F, and $t2$ containing leaves C, F, D. Note that $t2$ is rooted, and so when we reconnect $t1$ and $t2$ we select an arbitrary edge in $t1$ and connect it to the root of $t2$.

TBR is similar to SPR except that we can connect any two edges of $t1$ and $t2$.

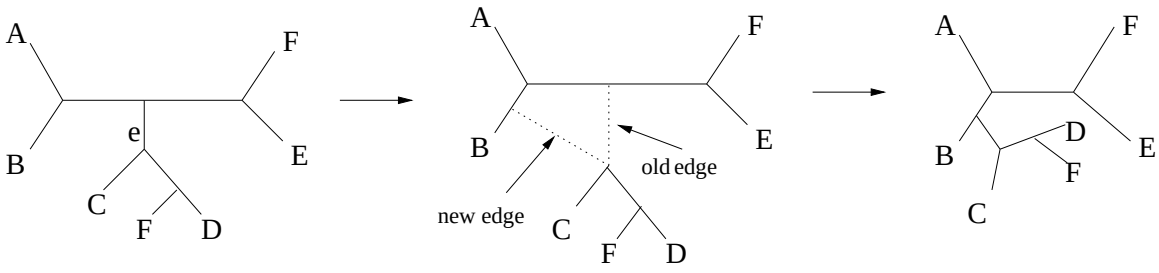


Figure 5: An SPR move on edge e

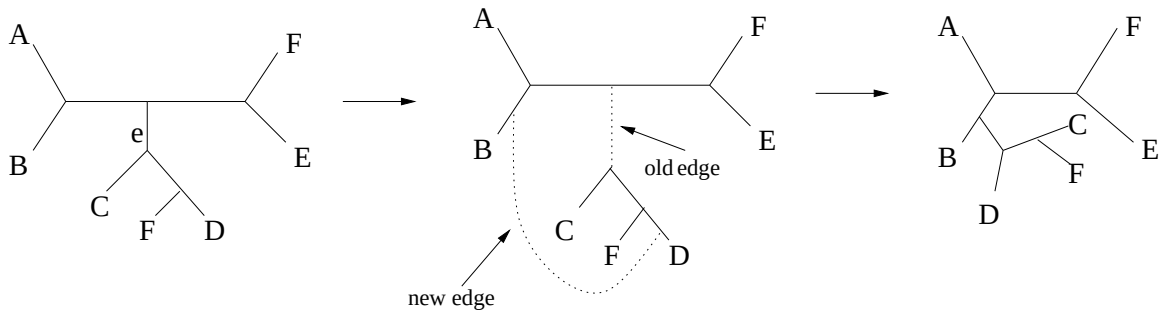


Figure 6: A TBR move on edge e

8 Supertrees

8.1 Subtree compatibility

The input to the subtree compatibility problem is a set of trees $\mathcal{T}$ and the output is a supertree T such that $\forall t \in \mathcal{T}, T_{|L(t)} = t$, if such a tree exists. This is an NP-hard problem. This problem also defines the notion of a compatible supertree. In the case that all the input trees are rooted, this problem can then be solved in polynomial time.

8.2 Supertree methods

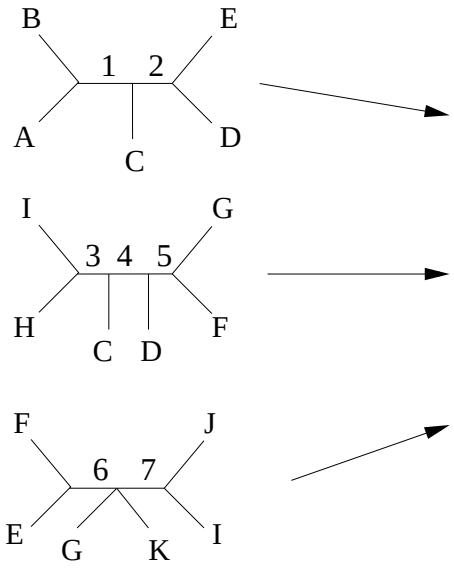
Since the compatibility problem is NP-hard in general, all methods developed are heuristics. We will look at Matrix Representation using Parsimony (MRP) and the Strict Consensus Merger (SCM) as tools for constructing parsimony trees.

8.2.1 MRP

We first describe the *MRP encoding* of a set of subtrees $\mathcal{T}$. An example is illustrated in Figure 7.

MRP encoding

- **Input:** Set of trees $\mathcal{T}$.
- **Output:** A set of sequences S, over the alphabet $\Sigma = \{0, 1, ?\}$, for the species $L(\mathcal{T}) = \cup_{T \in \mathcal{T}} L(T)$.



	1	2	3	4	5	6	7
A	0	0	?	?	?	?	?
B	0	0	?	?	?	?	?
C	1	0	1	0	0	?	?
D	1	1	1	1	0	?	?
E	1	1	?	?	?	0	0
F	?	?	1	1	1	0	0
G	?	?	1	1	1	1	0
H	?	?	0	0	0	?	?
I	?	?	0	0	0	1	1
J	?	?	?	?	?	1	1
K	?	?	?	?	?	1	0

Input trees

MRP encoding
data matrix

Figure 7: Each of the internal edges in each tree (edges numbered from 1 through 7) give rise to a character in the MRP encoding matrix shown on the right.

- **Algorithm:** The algorithm proceeds by defining sites to build a new data matrix S. Initialize S to the $n \times 0$ data matrix where $n = |L(\mathcal{T})|$ (and each row corresponds to a species in $L(\mathcal{T})$), and set $i = 0$. Let S_{si} denote the i^{th} site of species s ($s \in L(\mathcal{T})$).
  For each tree $T \in \mathcal{T}$ do the following:
  - For each internal edge $e \in E(T)$
    * Let $\pi(e)$ be the bipartition $\{A, B\}$ induced by edge e where A and B are the sets of taxa in the bipartition.
    * For each species in $s \in L(\mathcal{T})$ define the i^{th} site by setting $S_{si} = 0$ if $s \in A$, $S_{si} = 1$ if $s \in B$, and $S_{si} =?$ if $s \notin A \cup B$.
    * Increment i by one.

The "?" state is used to denote *missing data* and, for an MP analysis in practice, it is either treated as an additional state or, the best possible state from Σ which minimizes the MP score is assigned to it.

Once we have constructed the MRP encoding of a set of trees, we then try to find the MP tree on this matrix. Since MP is NP-hard, clearly this problem is also NP-hard.

8.2.2 SCM

SCM takes two trees T_1, T_2 and returns a tree T_{12} on the leaf set $L(T_1) \cup L(T_2)$.

- **Input:** Trees T_1, T_2
- **Output:** A tree T_{12} whose leaf set is $L(T_1) \cup L(T_2)$
- **Algorithm:**
  - Set $X = L(T_1) \cap L(T_2)$ where $L(T)$ is the leaf set of tree T. We call X the *backbone* and it must satisfy $|X| \geq 3$, otherwise the merger is not defined.
  - Compute the strict consensus, T_X, of T_1 and T_2, each restricted to the leaf set X i.e., $T_X = StrictConsensus(T_{1|X}, T_{2|X})$.
  - Add the remaining taxa from T_1 and T_2 into T_X to form T, so as to preserve as much structure as possible. See Figures 10 and 11 for examples of how this merger is accomplished. Note that it is possible that some piece of each tree T_1 and T_2 may attach onto the same edge of T_X; such an event is called a *collision*.

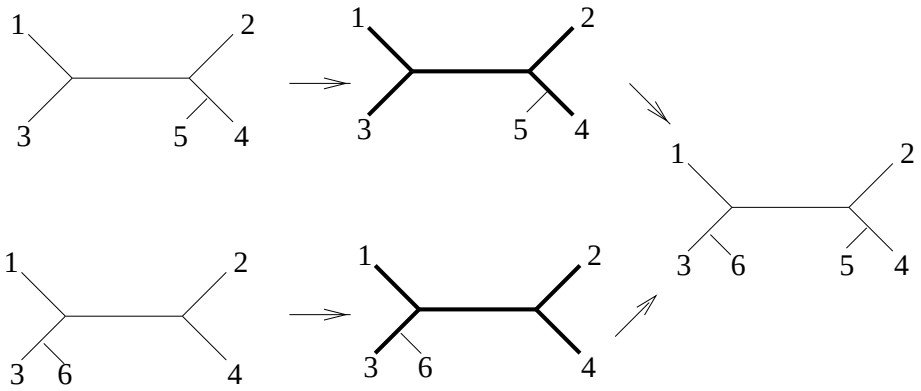


Figure 8: SCM on compatible trees without any collisions. We merge the two trees by first computing the strict consensus of the two trees restricted to the backbone (highlighted by thick edges). Note that the strict consensus does not contract edges in this case, since the trees are compatible.

Trees T_1 and T_2 are said to be *compatible* if there exists a tree T such that $T_{|L(T_1)} = T_1$ and $T_{|L(T_2)} = T_2$. If there is no such tree T then T_1 and T_2 are said to be *incompatible*. Figures 8 and 9 illustrate the SCM algorithm on compatible trees and Figures 10, and 11 demonstrate the SCM algorithm on incompatible trees, both with and without collisions.

9 Disk-Covering Methods (DCMs)

DCMs are divide-and-conquer techniques used for improving the performance of a given *base method*. This base method may be an MP heuristic, NJ, UPGMA, or any method for reconstructing phylogenies. DCMs have four steps:

1. Decompose the input set of aligned sequences Sinto smaller subsets $s_1, s_2, ..., s_k$. We call the subset *subproblems*. Different ways of decomposing the dataset have been studied and they yield different DCMs. (Later we will look at DCM3.)

2. For each subproblem compute a *subtree* using the given base method M. This gives us k subtrees $t_1, t_2, ..., t_k$, one for each subproblem.

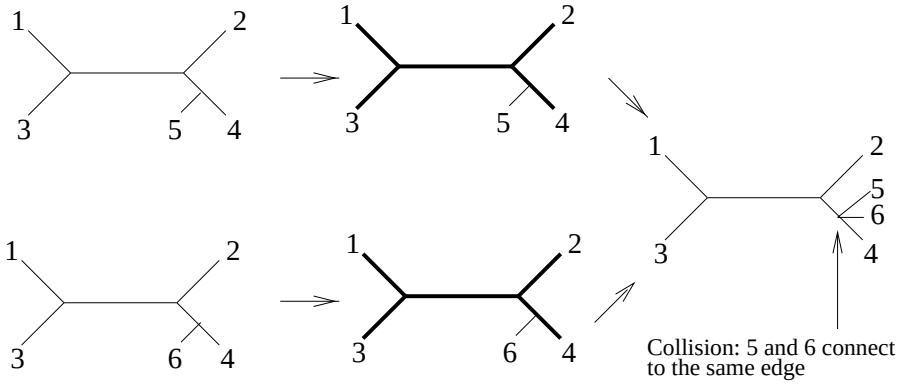


Figure 9: SCM on compatible trees with a collision. We merge the two trees by first computing the strict consensus of the two trees restricted to the backbone (backbone highlighted by thick edges).

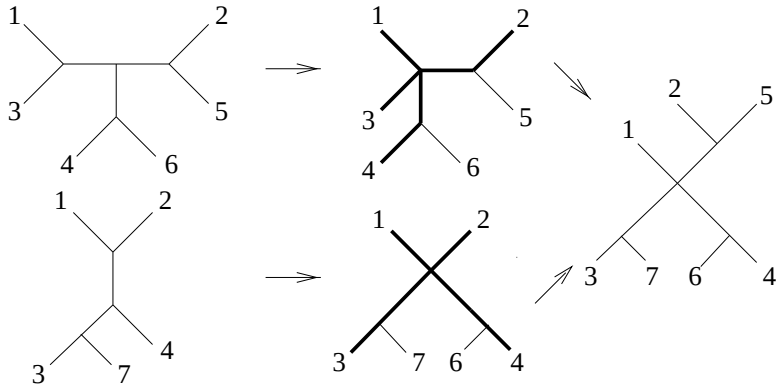


Figure 10: SCM on incompatible trees without any collisions. We merge the two trees by first computing the strict consensus of the two trees restricted to the backbone. Since the trees are incompatible, the strict consensus contracts edges in this example.

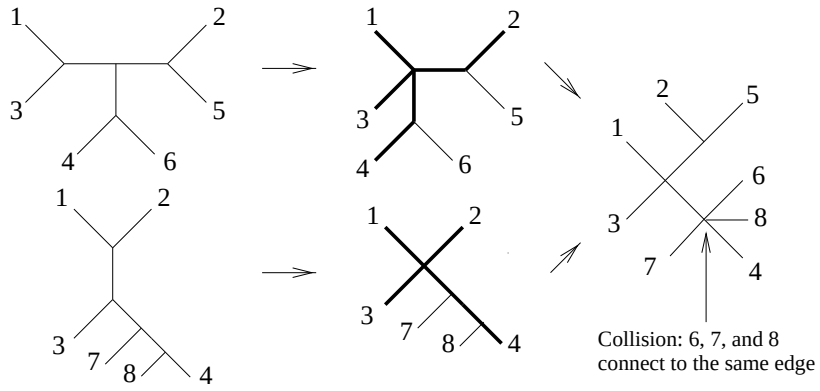


Figure 11: SCM on incompatible trees with a collision. We merge the two trees by computing the strict consensus of the two trees restricted to the backbone.

3. Merge the resulting subtrees using SCM to get a tree T'. Note that T' may not necessarily be binary

4. Refine the tree T' using a refinement procedure to get the binary tree T

The main idea behind DCMs is to get better accuracy and better running times by working on subsets of the original dataset with smaller *evolutionary diameters* and smaller sizes. The evolutionary diameter of a dataset is the max evolutionary distance over all pairs of sequences in the dataset.

10 Heuristics for MP and ML—Iterated local search

The basic hill-climbing search, which we saw earlier, suffers from the local minima problem. There is no way to tell if the local minima reached is our optimal solution. Therefore we have to keep searching using a technique called *random-restarts*. In this approach we use different random starting points and hope that one of our starting points will lead us to the global minimum. Iterated local search uses the local minimum to move to a "better" part of the search space instead of doing random restarts. Assume for generality that we are trying to minimize the function $g(s)$. Then an iterated local search approach is:

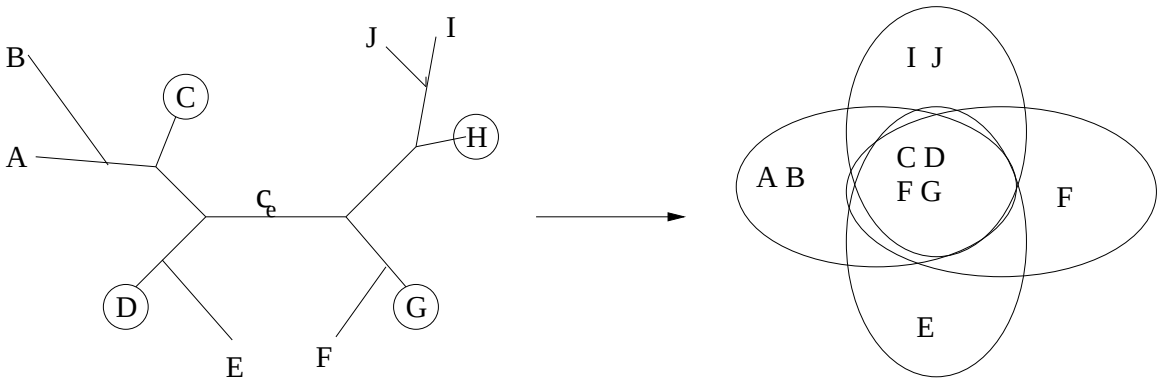


Figure 12: A DCM3 decomposition around the centroid edge c_e

- Determine an initial candidate solution s
- Perform local search starting from s. (This can be an iterative improvement procedure.)
- While termination condition not satisfied
  - Set $r = s$
  - Perform a *perturbation* on s to get s'
  - Perform local search starting from s' and let s'' be local minimum.
  - Based on acceptance criterion, set $s = s''$ or $s = r$.

We look at two such approaches for MP searches: iterative-DCM3 and the ratchet.

10.1 Iterative-DCM3 (I-DCM3)

First we define a DCM3 decomposition. This is illustrated in Figure 12. The input is a phylogeny and output is a set of sequences. We first find the *centroid edge* c_e and then compute the set of closest leaves X to c_e in the subtrees around c_e. X is the intersection common to all subsets and is necessary for the supertree phase. The remaining leaves will form the rest of the subsets.

Once we have a DCM3 decomposition, we (1) compute the subtrees, (2) merge them, (3) and then *randomly* refine the supertree to make it binary. We call this a DCM3 tree.

In I-DCM3 we begin with a hill-climbing search and stop at a local minima T. To escape this minima we use T as a *guide-tree* to construct a DCM3 tree. Various methods can be used as the base method, but we won't concern ourselves with that for now; however, note that performance depends upon what base method is selected.

Once we have the DCM3 tree we then use it as a starting point for a new hill-climbing search. Once we reach a local minima we again use it as a guide-tree to construct a new DCM3 tree, and this process iterates in this manner. Note that I-DCM3 can be used for computing MP or ML trees by selecting an appropriate MP or ML heuristic as the base method.

Smaller subsets can be obtained by recursively applying DCM3 to its subsets. This is called recursive-iterative-DCM3.

10.2 Ratchet

The ratchet is a heuristic for solving MP. The ratchet approach is to perturb the input set of sequences to yield a different landscape. The perturbation is done by randomly selecting $p\%$ of the sites and setting the weight of each site to w. The weight of a site is the number of times it appears in the alignment. Normally p is 25 and w is 2; these settings appear to work well in practice.

The search then proceeds from the local minimum using the perturbed data for computing MP scores. Once the local minima is reached the data is "reset" to its original form, and the search then proceeds from the current minimum. This process iterates in this manner.