Maximum Likelihood Methods in Phylogenetics

ML trees and likelihood-based tree topology testing

Heiko A. Schmidt

Center for Integrative Bioinformatics Vienna (CIBIV)
Max F. Perutz Laboratories (MFPL)
Vienna, Austria
heiko.schmidt@univie.ac.at

September 2007

Main Types of Phylogenetic Methods

Data	Method	Evaluation Criterion
Characters (Alignment)	**Maximum Parsimony**	Parsimony
Characters (Alignment)	**Statistical Approaches: Likelihood, Bayesian**	Evolutionary Models
Distances	**Distance Methods**	Evolutionary Models

What is the Maximum Likelihood (ML) Approach?

Having the probabilistic process of evolution and its parameters, we could compute the probability of any outcoming sequence data.

Probability

$p(\text{Data} \mid \text{Parameter set } \theta)$

What is the Maximum Likelihood (ML) Approach?

Having the probabilistic process of evolution and its parameters, we could compute the probability of any outcoming sequence data.

Probability

$$p(\text{Data} \mid \text{Parameter set } \theta)$$

But here we are interested in the process and parameters themselves.

What is the Maximum Likelihood (ML) Approach?

Having the probabilistic process of evolution and its parameters, we could compute the probability of any outcoming sequence data.

$$\begin{array}{ccc} \textbf{Probability} & & \textbf{Likelihood} \\ p(\text{Data} \mid \text{Parameter set } \theta) & \equiv & L(\text{Parameter set } \theta \mid \text{Data}) \end{array}$$

But here we are interested in the process and parameters themselves.

Hence, *"likelihood flips the probability around."*

What is the Maximum Likelihood (ML) Approach?

Having the probabilistic process of evolution and its parameters, we could compute the probability of any outcoming sequence data.

Probability		**Likelihood**
$p(\text{Data} \mid \text{Parameter set } \theta)$	$\equiv$	$L(\text{Parameter set } \theta \mid \text{Data})$

But here we are interested in the process and parameters themselves.

Hence, *"likelihood flips the probability around."*

Aim: The ML approach seaches for that parameter set θ for the process (i.e., evolution) which maximizes the probability of our given dataset.

Problem: parameter sets

Problem: In phylogenetic analysis, the parameter set θ comprises:

- evolutionary model
- its parameters
- tree topology
- its branch lengths

Problem: parameter sets

Problem: In phylogenetic analysis, the parameter set θ comprises:

- evolutionary model
- its parameters
- tree topology
- its branch lengths

This makes ML a high dimensional optimization problem that usually cannot be solved in one go.

Problem: parameter sets

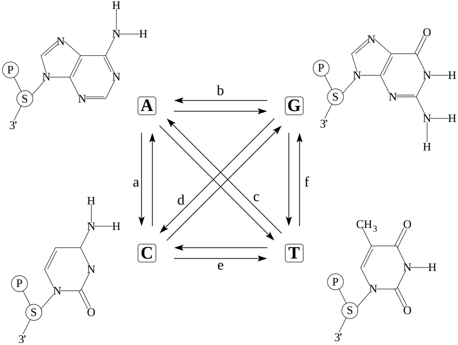
Problem: In phylogenetic analysis, the parameter set θ comprises:

- evolutionary model
- its parameters
- tree topology
- its branch lengths

This makes ML a high dimensional optimization problem that usually cannot be solved in one go.

Hence, some parameters like the substitution model and the model parameters are often determined/set separately from the tree.

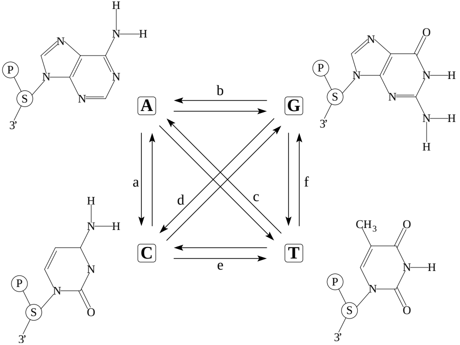
Evolutionary models are often described using a substitution rate matrix R and character frequencies Π. Here, 4×4 matrix for DNA models:



$$R = \begin{array}{c} \begin{array}{cccc} A & C & G & T \end{array} \\ \begin{pmatrix} - & a & b & c \\ a & - & d & e \\ b & d & - & f \\ c & e & f & - \end{pmatrix} \end{array}$$

$$\Pi = (\pi_A, \pi_C, \pi_G, \pi_T)$$

Evolutionary models are often described using a substitution rate matrix R and character frequencies Π. Here, 4×4 matrix for DNA models:

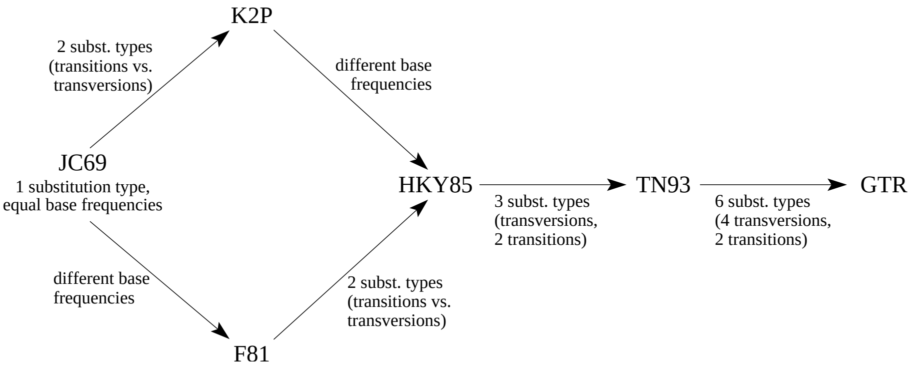


$$R = \begin{array}{c} \begin{array}{cccc} A & C & G & T \end{array} \\ \begin{pmatrix} - & a & b & c \\ a & - & d & e \\ b & d & - & f \\ c & e & f & - \end{pmatrix} \end{array}$$

$$\Pi = (\pi_A, \pi_C, \pi_G, \pi_T)$$

R and Π are then combined to a substitution probability matrix $P(t)$ wich allows us to compute the probability $P_{ij}(t)$ of a change $i \rightarrow j$ over a time t.

DNA substitution models



There are further submodels (see Modeltest) and extensions like models assuming rate heterogeneity and codon models.

Protein Models

Generally this is the same for protein sequences, but with 20×20 matrices. Some protein models are:

- Poisson model ("JC69" for proteins)
- Dayhoff (Dayhoff *et al.*, 1978)
- JTT (Jones *et al.*, 1992)
- mtREV (Adachi & Hasegawa, 1996)
- cpREV (Adachi *et al.*, 2000)
- VT (Müller & Vingron, 2000)
- WAG (Whelan & Goldman, 2000)
- ~~BLOSUM 62 (Henikoff & Henikoff, 1992)~~

Computing ML Distances Using $\mathbf{P_{ij}(t)}$

The Likelihood of sequence s evolving to s' in time t:

$$L(t|s \longrightarrow s') = \prod_{i=1}^{m} \left(\Pi(s_i) \cdot P_{s_i s_i'}(t) \right)$$

Likelihood surface for two sequences under JC69:

```
GATCCTGAGAGAAATAAAC
GGTCCTGACAGAAATAAAC
```

Computing ML Distances Using $\mathbf{P_{ij}(t)}$

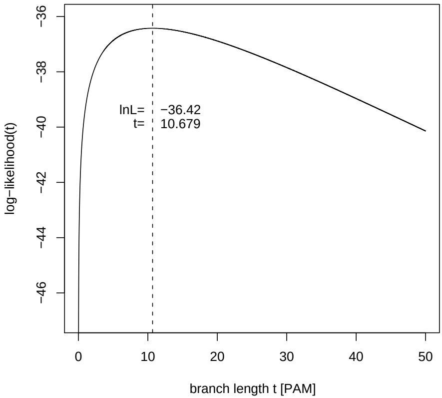
The Likelihood of sequence s evolving to s' in time t:

$$L(t|s \longrightarrow s') = \prod_{i=1}^{m} \Big(\Pi(s_i) \cdot P_{s_i s_i'}(t) \Big)$$

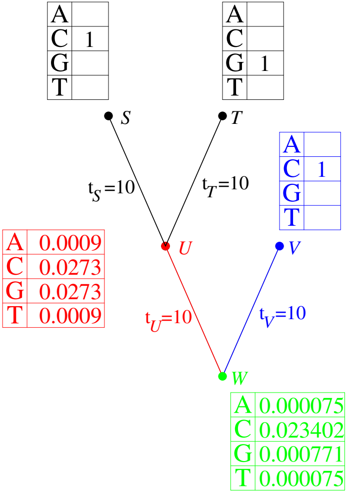
Likelihood surface for two sequences under JC69:

GATCCTGAGAGAAATAAAC
GGTCCTGACAGAAATAAAC

Note: we do not compute the probability of the distance t but that of the data $D = \{s, s'\}$.



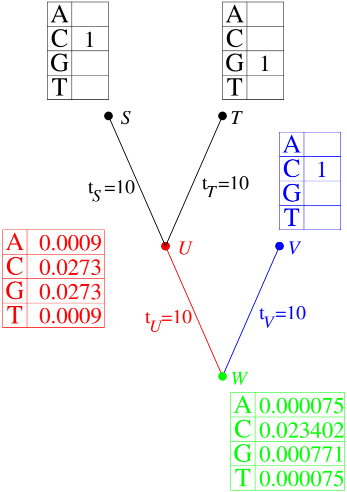
Likelihoods of Trees (Single column $\begin{smallmatrix}C\\G\\C\end{smallmatrix}$, given tree)



Likelihoods of nucleotides at inner nodes:

$$L_U(i) = [P_{iC}(10) \cdot L(C)] \cdot [P_{iG}(10) \cdot L(G)]$$

Likelihoods of Trees (Single column $\begin{smallmatrix} C \\ G \\ C \end{smallmatrix}$, given tree)



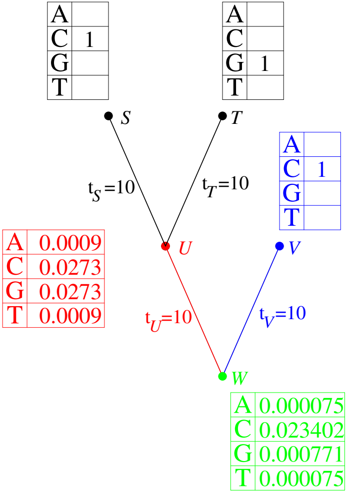
Likelihoods of nucleotides at inner nodes:

$$L_U(i) = \quad [P_{iC}(10) \cdot L(C)] \cdot [P_{iG}(10) \cdot L(G)]$$

$$L_W(i) = \quad \left[\sum_{\substack{u= \\ ACGT}} P_{iu}(t_U) \cdot L_U(u)\right] \cdot$$

$$\left[\sum_{\substack{v= \\ ACGT}} P_{iv}(t_V) \cdot L_V(v)\right]$$

Likelihoods of Trees (Single column $\begin{smallmatrix} C \\ G \\ C \end{smallmatrix}$, given tree)



Likelihoods of nucleotides at inner nodes:

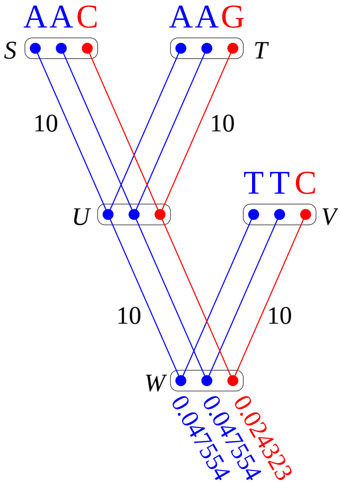
$$L_U(i) = \quad [P_{iC}(10) \cdot L(C)] \cdot [P_{iG}(10) \cdot L(G)]$$

$$L_W(i) = \quad \left[\sum_{\substack{u= \\ ACGT}} P_{iu}(t_U) \cdot L_U(u) \right] \cdot$$

$$\left[\sum_{\substack{v= \\ ACGT}} P_{iv}(t_V) \cdot L_V(v) \right]$$

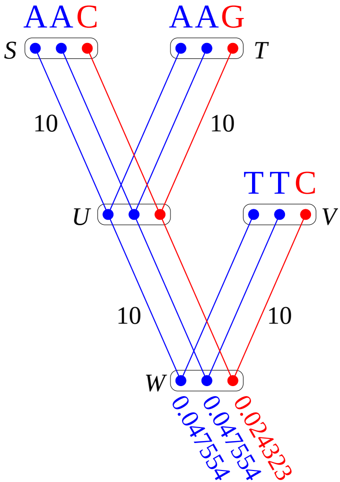
Site-Likelihood of an alignment column k:

$$L^{(k)} = \sum_{\substack{i= \\ ACGT}} \pi_i \cdot L_W(i) = 0.024323$$



Considering this tree with $n = 3$ sequences of length $m = 3$ the tree likelihood of this tree is

$$\mathcal{L}(T) = \prod_{k=1}^{m} L^{(k)}$$

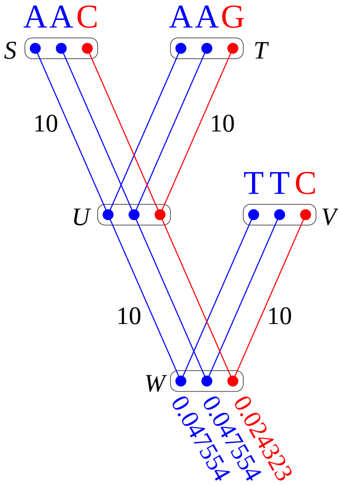


Considering this tree with $n = 3$ sequences of length $m = 3$ the tree likelihood of this tree is

$$\mathcal{L}(T) = \prod_{k=1}^{m} L^{(k)} = 0.047554^2 \cdot 0.024323$$

$$= 0.000055$$

Likelihoods of Trees (multiple columns)



Considering this tree with $n = 3$ sequences of length $m = 3$ the tree likelihood of this tree is

$$\mathcal{L}(T) = \prod_{k=1}^{m} L^{(k)} = 0.047554^2 \cdot 0.024323 = 0.000055$$

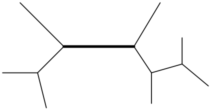
or the log-likelihood

$$\ln \mathcal{L}(T) = \sum_{k=1}^{m} \ln L^{(k)} = -9.80811$$

Adjusting Branch Lengths Step-By-Step

To compute optimal branch lengths do the following. Initialize the branch lengths.
Choose a branch (A.). Move the virtual root to an adjacent node (B.).
Compute all partial likelihoods recursively (C.). Adjust the branch length to maximize the likelihood value (D.).

A.

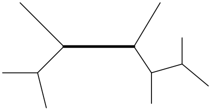


Adjusting Branch Lengths Step-By-Step

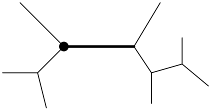
To compute optimal branch lengths do the following. Initialize the branch lengths.

Choose a branch (A.). Move the virtual root to an adjacent node (B.). Compute all partial likelihoods recursively (C.). Adjust the branch length to maximize the likelihood value (D.).

A.



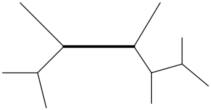
B.



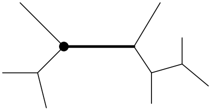
Adjusting Branch Lengths Step-By-Step

To compute optimal branch lengths do the following. Initialize the branch lengths.
Choose a branch (A.). Move the virtual root to an adjacent node (B.). Compute all partial likelihoods recursively (C.). Adjust the branch length to maximize the likelihood value (D.).

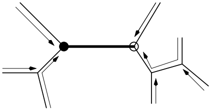
A.



B.



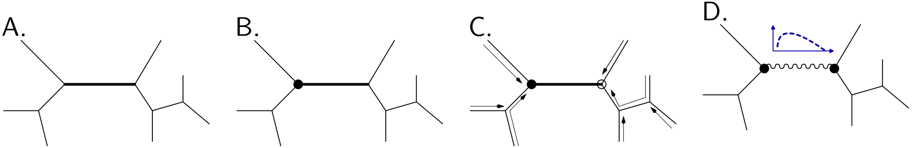
C.



Adjusting Branch Lengths Step-By-Step

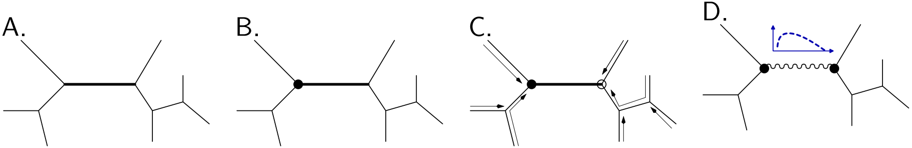
To compute optimal branch lengths do the following. Initialize the branch lengths.

Choose a branch (A.). Move the virtual root to an adjacent node (B.). Compute all partial likelihoods recursively (C.). Adjust the branch length to maximize the likelihood value (D.).



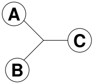
Adjusting Branch Lengths Step-By-Step

To compute optimal branch lengths do the following. Initialize the branch lengths.
Choose a branch (A.). Move the virtual root to an adjacent node (B.). Compute all partial likelihoods recursively (C.). Adjust the branch length to maximize the likelihood value (D.).

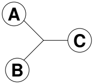


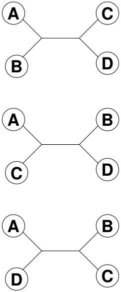
Repeat this for every branch until no better likelihood is gained.

Number of Trees to Examine...

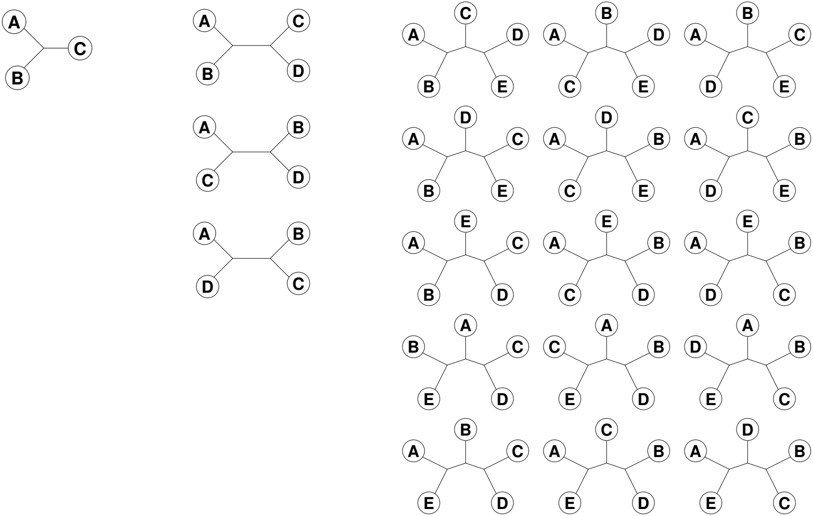


Number of Trees to Examine...

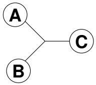


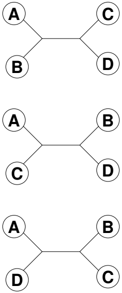


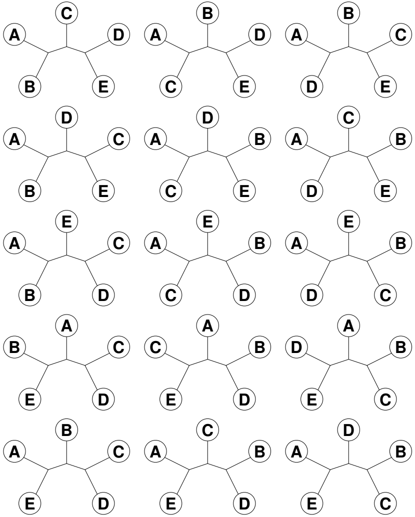
Number of Trees to Examine…



Number of Trees to Examine...

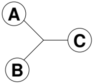


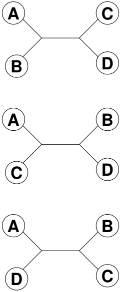




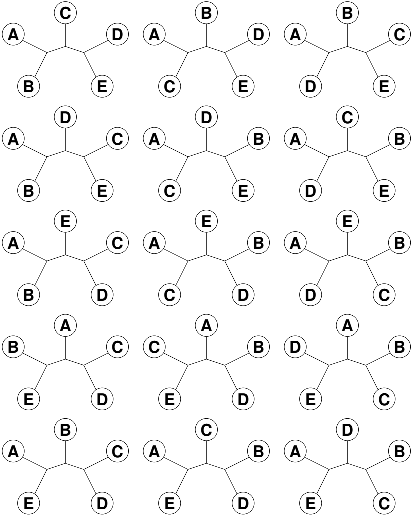
$$B(n) \quad = \frac{(2n-5)!}{2^{n-3}(n-3)!}$$

Number of Trees to Examine...





$$
\begin{aligned}
B(n) &= \frac{(2n-5)!}{2^{n-3}(n-3)!} \\
B(10) &= 2027025 \\
B(55) &= 2.98 \cdot 10^{84} \\
B(100) &= 1.70 \cdot 10^{182}
\end{aligned}
$$



Finding the ML Tree

Exhaustive Search: guarantees to find the optimal tree, because all trees are evaluated, but not feasible for more than 10-12 taxa.

Finding the ML Tree

Exhaustive Search: guarantees to find the optimal tree, because all trees are evaluated, but not feasible for more than 10-12 taxa.

Branch and Bound: guarantees to find the optimal tree, without searching certain parts of the tree space – can run on more sequences, but often not for current-day datasets.

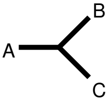
Finding the ML Tree

Exhaustive Search: guarantees to find the optimal tree, because all trees are evaluated, but not feasible for more than 10-12 taxa.

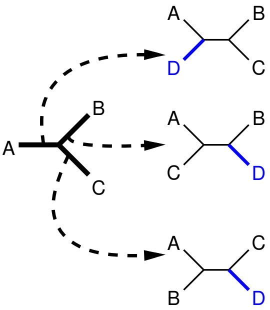
Branch and Bound: guarantees to find the optimal tree, without searching certain parts of the tree space – can run on more sequences, but often not for current-day datasets.

Heuristics: cannot guarantee to find the optimal tree, but are at least able to analyze large datasets.

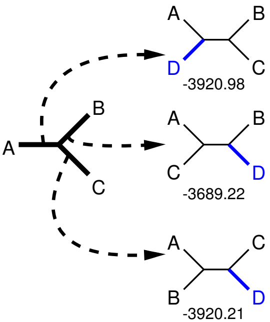
Build up a tree: Stepwise Insertion



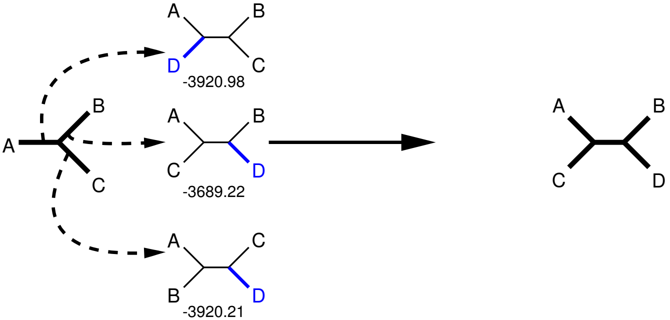
Build up a tree: Stepwise Insertion



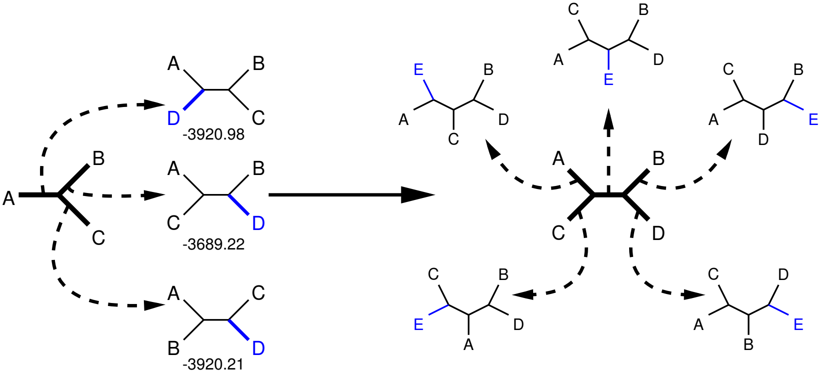
Build up a tree: Stepwise Insertion



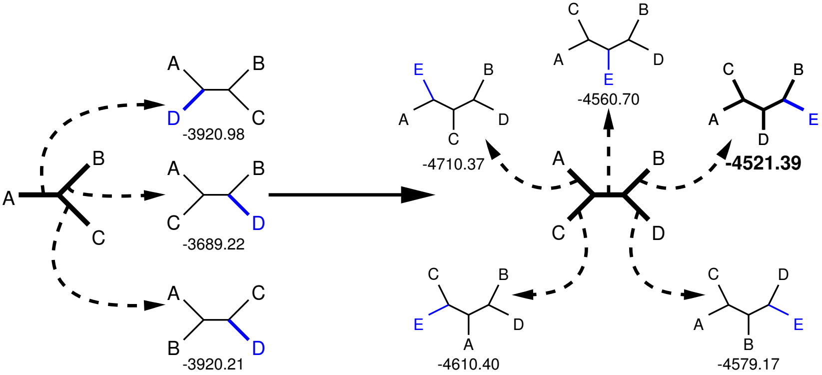
Build up a tree: Stepwise Insertion



Build up a tree: Stepwise Insertion

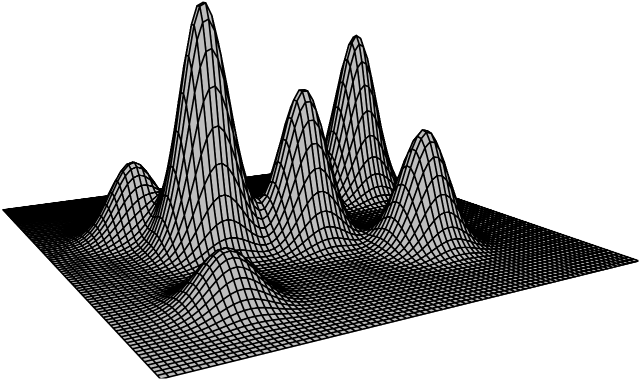


Build up a tree: Stepwise Insertion



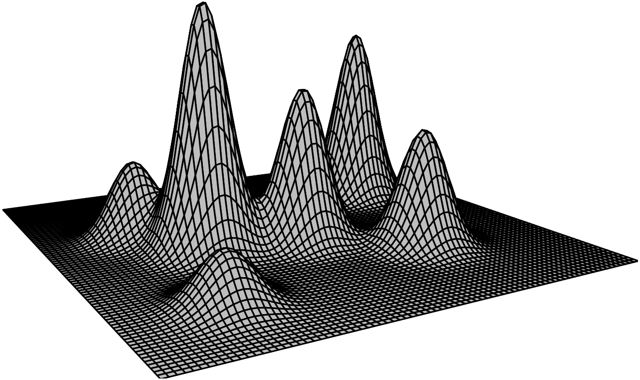
Local Maxima

What if we have multiple maxima in the likelihood surface?



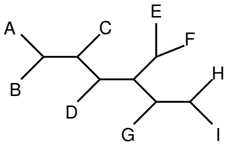
Local Maxima

What if we have multiple maxima in the likelihood surface?

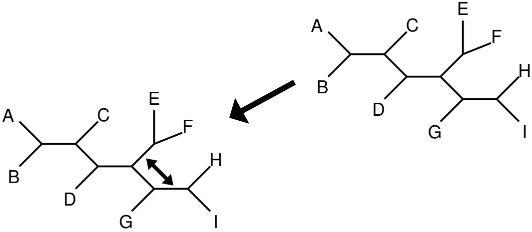


Tree rearrangements to escape local maxima.

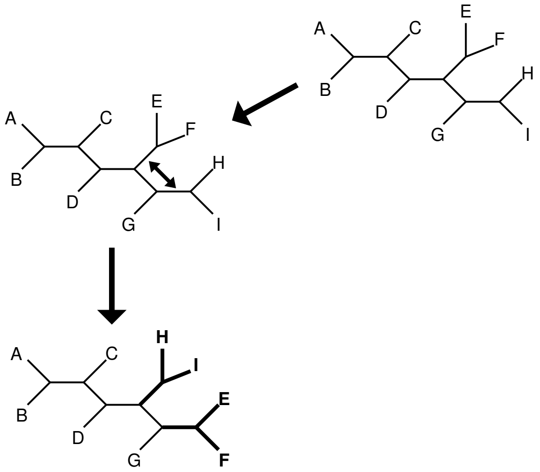
Tree Rearrangements: Scanning a Tree's Neighborhood



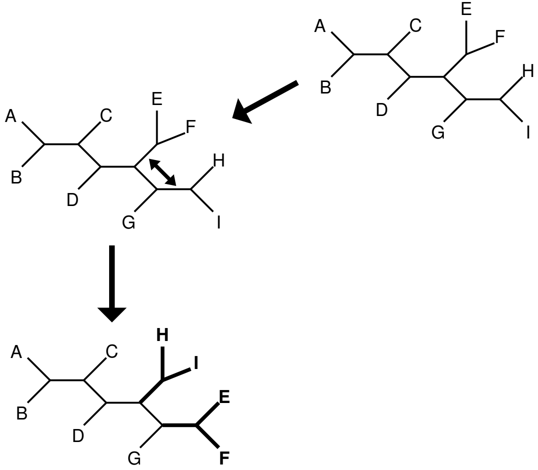
Tree Rearrangements: Scanning a Tree's Neighborhood



Tree Rearrangements: Scanning a Tree's Neighborhood



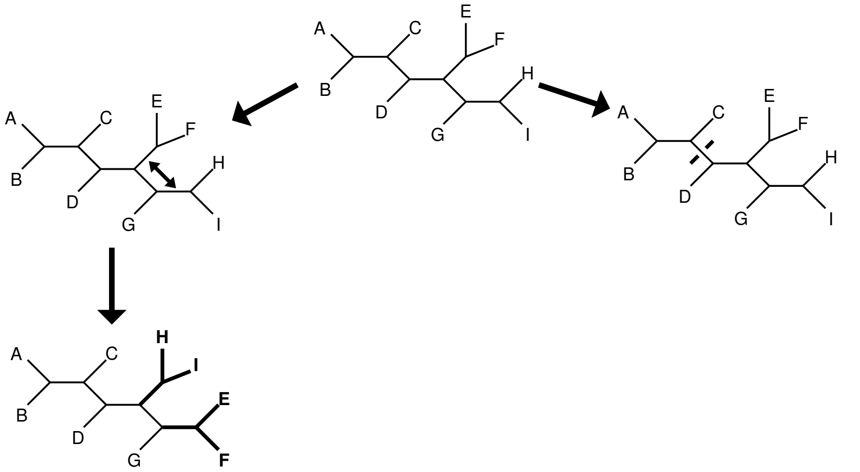
Tree Rearrangements: Scanning a Tree's Neighborhood



Nearest Neighbor Interchange

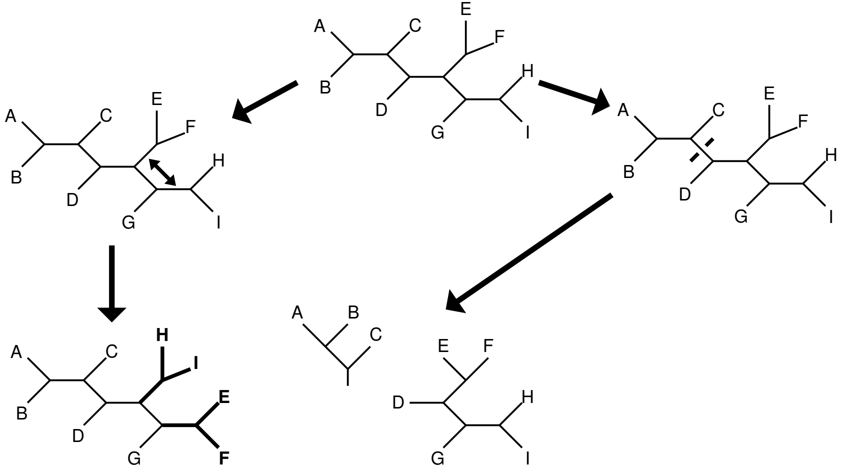
Possible NNI trees = O(n)

Tree Rearrangements: Scanning a Tree's Neighborhood



Nearest Neighbor Interchange

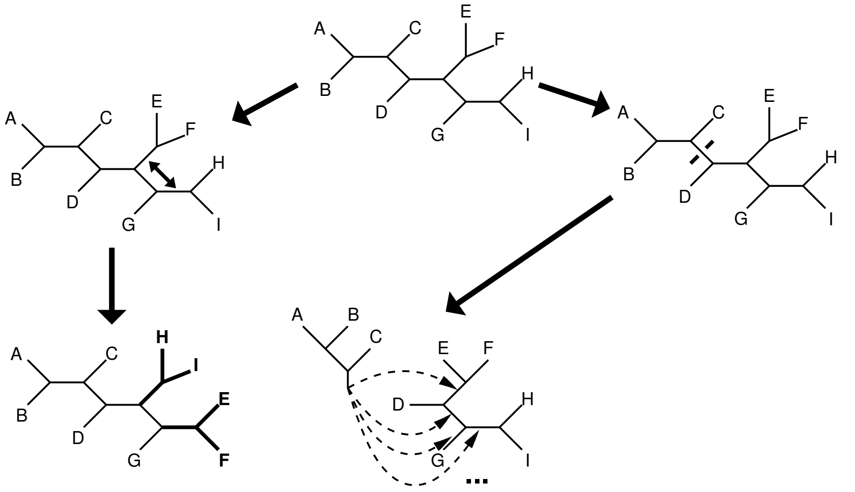
Possible NNI trees = O(n)



Nearest Neighbor Interchange

Possible NNI trees = O(n)

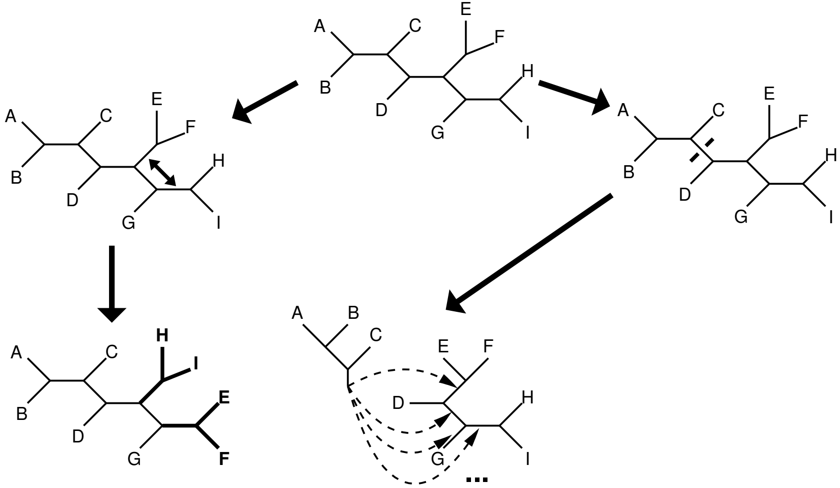
Tree Rearrangements: Scanning a Tree's Neighborhood



Nearest Neighbor Interchange

Possible NNI trees = O(n)

Tree Rearrangements: Scanning a Tree's Neighborhood



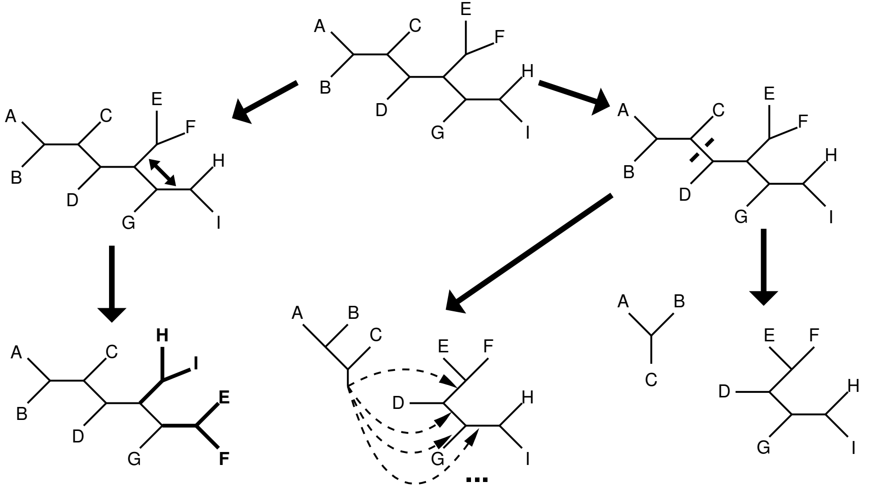
Nearest Neighbor Interchange

Possible NNI trees = O(n)

subtree pruning + regrafting

Possible SPR trees = O(n*n)

Tree Rearrangements: Scanning a Tree's Neighborhood



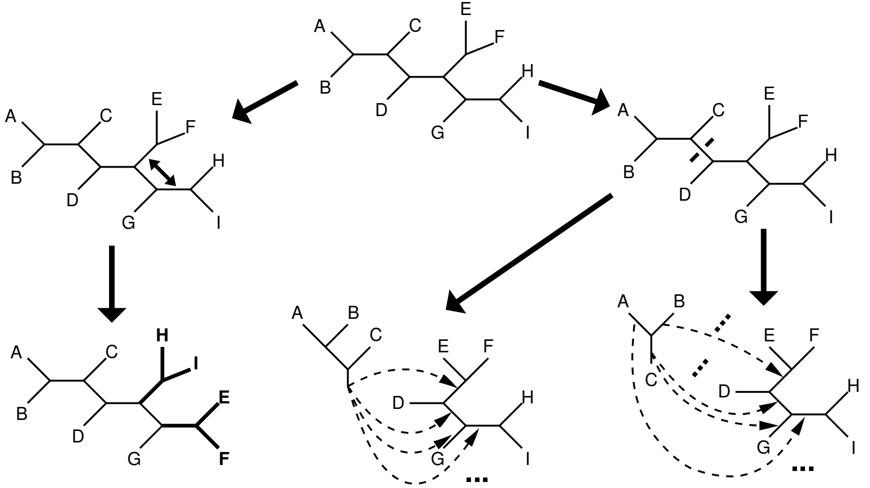
Nearest Neighbor Interchange

Possible NNI trees = O(n)

subtree pruning + regrafting

Possible SPR trees = O(n*n)

Tree Rearrangements: Scanning a Tree's Neighborhood



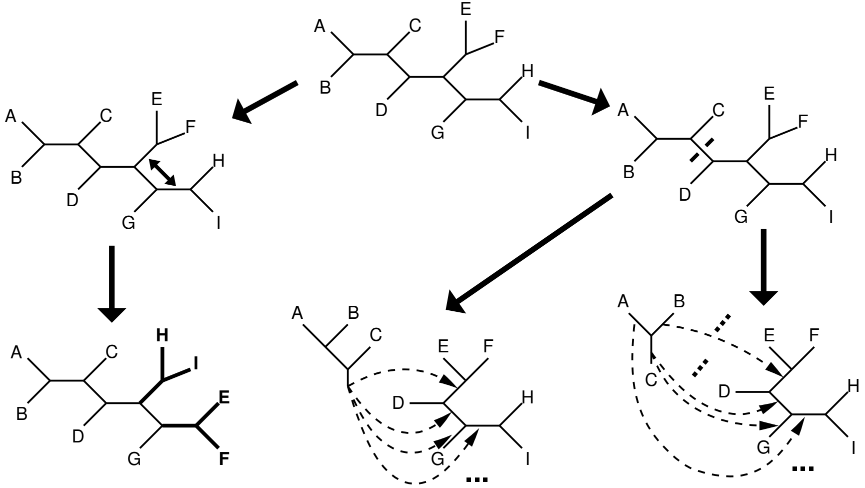
Nearest Neighbor Interchange

Possible NNI trees = O(n)

subtree pruning + regrafting

Possible SPR trees = O(n*n)

Tree Rearrangements: Scanning a Tree's Neighborhood



Nearest Neighbor Interchange

Possible NNI trees = O(n)

subtree pruning + regrafting

Possible SPR trees = O(n*n)

tree-bisection + reconnection

Possible TBR trees = O(n*n*n)

ML programs: DNAML (PHYLIP), fastDNAml

Concept: Stepwise insertion + NNI/SPR

1. Build tree with stepwise insertion
   (a) after each insertion optimize using NNI/local rearrangement (default, but user-adjustable gradually up to SPR; only fastDNAml)
   (b) repeat (a) rearrangements until no better tree found.

ML programs: DNAML (PHYLIP), fastDNAml

Concept: Stepwise insertion + NNI/SPR

1. Build tree with stepwise insertion
   (a) after each insertion optimize using NNI/local rearrangement (default, but user-adjustable gradually up to SPR; only fastDNAml)
   (b) repeat (a) rearrangements until no better tree found.
2. after the last insertion optimize using SPR/global rearrangement (in DNAML; in fastDNAml user-adjustable gradually down to NNI)
3. repeat (2) rearrangements until no better tree found.

ML programs: DNAML (PHYLIP), fastDNAml

Concept: Stepwise insertion + NNI/SPR

1. Build tree with stepwise insertion
   (a) after each insertion optimize using NNI/local rearrangement (default, but user-adjustable gradually up to SPR; only fastDNAml)
   (b) repeat (a) rearrangements until no better tree found.
2. after the last insertion optimize using SPR/global rearrangement (in DNAML; in fastDNAml user-adjustable gradually down to NNI)
3. repeat (2) rearrangements until no better tree found.

Pro: Evaluating large neighborhood with SPR.

Con: Slow.

ML programs: DNAML (PHYLIP), fastDNAml

Concept: Stepwise insertion + NNI/SPR

1. Build tree with stepwise insertion
   (a) after each insertion optimize using NNI/local rearrangement (default, but user-adjustable gradually up to SPR; only fastDNAml)
   (b) repeat (a) rearrangements until no better tree found.
2. after the last insertion optimize using SPR/global rearrangement (in DNAML; in fastDNAml user-adjustable gradually down to NNI)
3. repeat (2) rearrangements until no better tree found.

Pro: Evaluating large neighborhood with SPR.

Con: Slow.

Note: To save time, in other methods steps (1) and (2) are usually substituted by swiftly computed trees (e.g., BioNJ).

ML programs: RAxML

Concept: MP tree + LSR
Descendant on fastDNAml, but ...

1. Starting with MP tree.
2. Uses *lazy subtree rearrangements* (only the 3 insertion branches are optimized), collecting candidates.
3. Candidates are evaluated.
4. Iterating (2)-(4).

ML programs: RAxML

Concept: MP tree + LSR
Descendant on fastDNAml, but . . .

1. Starting with MP tree.
2. Uses *lazy subtree rearrangements* (only the 3 insertion branches are optimized), collecting candidates.
3. Candidates are evaluated.
4. Iterating (2)-(4).

Pro: Fast,
smart algorithmic and numerical optimized ML computation.

Con: Only few trees fully evaluated trees.

ML programs: PHYML

Concept: BioNJ tree + fastNNI

1. Start with BioNJ tree.
2. Do fastNNIs to optimize trees, i.e., evaluate all NNIs simultaneously and then merge all best ones which are non-conflicting.
3. Repeat (1) until no better tree found anymore.

ML programs: PHYML

Concept: BioNJ tree + fastNNI

1. Start with BioNJ tree.
2. Do fastNNIs to optimize trees, i.e., evaluate all NNIs simultaneously and then merge all best ones which are non-conflicting.
3. Repeat (1) until no better tree found anymore.

Pro: Fast

Con: Prone to get stuck on local optima due to NNI-only.
(SPR-based version PhyML-SPR has not been released yet.)

ML programs: IQPNNI

Concept: BioNJ tree + randomization + fastNNI

1. Start with BioNJ tree.
2. Do fastNNIs to optimize trees, i.e., evaluate all NNIs simultaneously and then accept all best ones which are non-conflicting. (after first round, identical to PHYML).
3. Remove randomly a certain amount of taxa and re-insert them by a fast and rough quartet-based method. (some randomization)
4. Repeat (2)-(3) until stop criterion is met.

ML programs: IQPNNI

Concept: BioNJ tree + randomization + fastNNI

1. Start with BioNJ tree.
2. Do fastNNIs to optimize trees, i.e., evaluate all NNIs simultaneously and then accept all best ones which are non-conflicting. (after first round, identical to PHYML).
3. Remove randomly a certain amount of taxa and re-insert them by a fast and rough quartet-based method. (some randomization)
4. Repeat (2)-(3) until stop criterion is met.

Pro: Can evade local optima,
offers automatic stopping criterion,
hints when search didn't run enough,
numerically optimized ML computation,
offers codon models

Con: slower than PhyML/RAxML

ML programs: Other strategies

- Genetic Algorithms (GARLI, GAML, MetaPIGA)
- Simulated Annealing (SSA, RAxML-SA)
- Quartet-based trees (TREE-PUZZLE, Qstar)
- . . .

Note: The first two are also based on NNI/SPR/TBR.

How reliable is the reconstructed tree:

- Usually programs deliver a single tree, but without confidence values for the subtrees.
- How can we assess reliability for the subtree?

- We can now reconstruct ML trees, but how comparable are the likelihoods, how reliable the groupings?
- Branch reliability can be checked, support values computed using:
  - Randomizing input orders in stepwise insertions (e.g., TREE-PUZZLE).
  - Jackknifing alignment columns + consensus.
  - Bootstrapping alignment columns + consensus.
  - Trees from Bayesian MCMC sampling + consensus.

Are two evolutionary trees/models different?

Given sequence alignments and substitution models, we can reconstruct tree and compute their likelihoods.

Are two evolutionary trees/models different?

Given sequence alignments and substitution models, we can reconstruct tree and compute their likelihoods.

But can we decide from the likelihood

- which is the best substitution model to use?
- which tree is better?
- whether two tree likelihoods are significantly different?
- whether one tree likelihood is significantly better/worse?

Are two evolutionary trees/models different?

Given sequence alignments and substitution models, we can reconstruct tree and compute their likelihoods.

But can we decide from the likelihood

- which is the best substitution model to use?

$\longrightarrow$ yes, using (hierarchical) LRTs

- which tree is better?
- whether two tree likelihoods are significantly different?
- whether one tree likelihood is significantly better/worse?

Are two evolutionary trees/models different?

Given sequence alignments and substitution models, we can reconstruct tree and compute their likelihoods.

But can we decide from the likelihood

- which is the best substitution model to use?

$\longrightarrow$ yes, using (hierarchical) LRTs

- which tree is better?

$\longrightarrow$ only if likelihoods are computed from identical datasets/taxa.

- whether two tree likelihoods are significantly different?

- whether one tree likelihood is significantly better/worse?

Are two evolutionary trees/models different?

Given sequence alignments and substitution models, we can reconstruct tree and compute their likelihoods.

But can we decide from the likelihood

- which is the best substitution model to use?

$\longrightarrow$ yes, using (hierarchical) LRTs

- which tree is better?

$\longrightarrow$ only if likelihoods are computed from identical datasets/taxa.

- whether two tree likelihoods are significantly different?

$\longrightarrow$ yes, for identical datasets/taxa

- whether one tree likelihood is significantly better/worse?

Are two evolutionary trees/models different?

Given sequence alignments and substitution models, we can reconstruct tree and compute their likelihoods.

But can we decide from the likelihood

- which is the best substitution model to use?

$\longrightarrow$ yes, using (hierarchical) LRTs

- which tree is better?

$\longrightarrow$ only if likelihoods are computed from identical datasets/taxa.

- whether two tree likelihoods are significantly different?

$\longrightarrow$ yes, for identical datasets/taxa

- whether one tree likelihood is significantly better/worse?

$\longrightarrow$ yes, for identical datasets/taxa

Are two evolutionary trees/models different?

Given sequence alignments and substitution models, we can reconstruct tree and compute their likelihoods.

But can we decide from the likelihood

- which is the best substitution model to use?

$\longrightarrow$ yes, using (hierarchical) LRTs

- which tree is better?

$\longrightarrow$ only if likelihoods are computed from identical datasets/taxa.

- whether two tree likelihoods are significantly different?

$\longrightarrow$ yes, for identical datasets/taxa

- whether one tree likelihood is significantly better/worse?

$\longrightarrow$ yes, for identical datasets/taxa

These questions can be assesed by hypothesis testing.

Hypothesis testing: prerequisistes

- What question do I want to answer?

Hypothesis testing: prerequisistes

- What question do I want to answer?
  - Say should I use the JC model or the GTR model?

Hypothesis testing: prerequisistes

- What question do I want to answer?
  - Say should I use the JC model or the GTR model?
  - Or perhaps, Is tree T_a statistically different from tree T_b?

Hypothesis testing: prerequisistes

- What question do I want to answer?
  - Say should I use the JC model or the GTR model?
  - Or perhaps, Is tree T_a statistically different from tree T_b?
- It is important to note that you should know the null hypothesis/hypotheses **before** you “collect” the data.

Testing Tree Topologies with LRT?

- **Only nested models** can be tested by ordinary LRT:
  One model (H_0, Null-model, constraint model) is nested in another model (H_A, alternative, unconstraint model) if the model H_0 can be produced by restricting parameters in model H_A.

Testing Tree Topologies with LRT?

- **Only nested models** can be tested by ordinary LRT:
  One model (H_0, Null-model, constraint model) is nested in another model (H_A, alternative, unconstraint model) if the model H_0 can be produced by restricting parameters in model H_A.
- two differrent topologies are not nested.

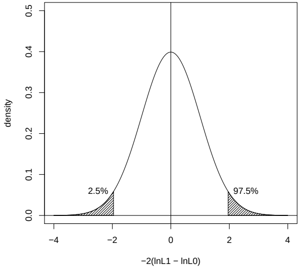
Testing Tree Topologies with LRT?

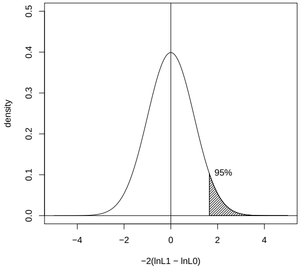
- **Only nested models** can be tested by ordinary LRT:
  One model (H_0, Null-model, constraint model) is nested in another model (H_A, alternative, unconstraint model) if the model H_0 can be produced by restricting parameters in model H_A.
- two differrent topologies are not nested.
- Thus, LRT cannot be used on different topologies, because the assumption of the χ^2 distribution does not fit.

Testing Tree Topologies with LRT?

- **Only nested models** can be tested by ordinary LRT:
  One model (H_0, Null-model, constraint model) is nested in another model (H_A, alternative, unconstraint model) if the model H_0 can be produced by restricting parameters in model H_A.
- two differrent topologies are not nested.
- Thus, LRT cannot be used on different topologies, because the assumption of the χ^2 distribution does not fit.
- Hence, other (bootstrap-bases) methods have been devised to determine the distribution of log-likelihood differences for testing (e.g., KH or SH test).

Usual Null-Hypotheses:





First the Null hypothesis has to be stated, for example:

- **top:** The two likelihood are not significantly different – i.e. their expected difference $E(\ln L_1 - \ln L_0) = 0$.
- **bottom:** The 2nd likelihood is not significantly worse – i.e. their expected difference $E(\ln L_1 - \ln L_0) \leq 0$.

If the observed value falls into the white area, the Null hypothesis cannot be rejected. If it falls into the grey area, this is interpreted as support for the alternative by rejecting the Null hypothesis.

Basic Idea:

- Compute log-likelihood values $L_1, \ldots, L_N$ for your trees $T_1, \ldots, T_N$.

Basic Idea:

- Compute log-likelihood values $L_1, \ldots, L_N$ for your trees $T_1, \ldots, T_N$.
- Draw bootstrap samples i from the alignment, re-estimate the log-likelihood values $L_x^{(i)}$ for each tree T_x and for each sample i.

Basic Idea:

- Compute log-likelihood values $L_1, \ldots, L_N$ for your trees $T_1, \ldots, T_N$.
- Draw bootstrap samples i from the alignment, re-estimate the log-likelihood values $L_x^{(i)}$ for each tree T_x and for each sample i.
- Adjust the log-likelihoods with the mean by setting $\tilde{L}_x^{(i)} = L_x^{(i)} - \bar{L}_x^{(i)}$ (Centering) Centering is needed to correct make the bootstrap samples conforming to the Null model.

Basic Idea:

- Compute log-likelihood values $L_1, \ldots, L_N$ for your trees $T_1, \ldots, T_N$.
- Draw bootstrap samples i from the alignment, re-estimate the log-likelihood values $L_x^{(i)}$ for each tree T_x and for each sample i.
- Adjust the log-likelihoods with the mean by setting $\tilde{L}_x^{(i)} = L_x^{(i)} - \bar{L}_x^{(i)}$ (Centering) Centering is needed to correct make the bootstrap samples conforming to the Null model.
- Use the differences between the $\tilde{L}_x^{(i)}$ to determine the distribution of differences $\delta^{(i)} = \tilde{L}_y^{(i)} - \tilde{L}_z^{(i)}$.

Basic Idea:

- Compute log-likelihood values $L_1, \ldots, L_N$ for your trees $T_1, \ldots, T_N$.
- Draw bootstrap samples i from the alignment, re-estimate the log-likelihood values $L_x^{(i)}$ for each tree T_x and for each sample i.
- Adjust the log-likelihoods with the mean by setting $\tilde{L}_x^{(i)} = L_x^{(i)} - \bar{L}_x^{(i)}$ (Centering) Centering is needed to correct make the bootstrap samples conforming to the Null model.
- Use the differences between the $\tilde{L}_x^{(i)}$ to determine the distribution of differences $\delta^{(i)} = \tilde{L}_y^{(i)} - \tilde{L}_z^{(i)}$.
- Use the distribution of $\delta^{(i)}$ to test your trees.

Time Saving: RELL

- The re-optimization to get the log-likelihood values $L_x^{(i)}$ is very time consuming.

Time Saving: RELL

- The re-optimization to get the log-likelihood values $L_x^{(i)}$ is very time consuming.
- Hence, often the site-likelihoods are used fixed.

Time Saving: RELL

- The re-optimization to get the log-likelihood values $L_x^{(i)}$ is very time consuming.
- Hence, often the site-likelihoods are used fixed.
- During the bootstrap the already estimated site-log-likelihoods are sampled and added to produce $L_x^{(i)}$.

Time Saving: RELL

- The re-optimization to get the log-likelihood values $L_x^{(i)}$ is very time consuming.
- Hence, often the site-likelihoods are used fixed.
- During the bootstrap the already estimated site-log-likelihoods are sampled and added to produce $L_x^{(i)}$.
- The resampling of estimated log-likelihoods (RELL) has been shown to be often sufficient to produce the distribution of log-likelihood differences.

Original Kishino and Hasegawa test (KH test)

- This test was devised to test whether two *a priory* chosen trees (e.g., from a Markov Chain) are equally well supported by the dataset.

Original Kishino and Hasegawa test (KH test)

- This test was devised to test whether two *a priory* chosen trees (e.g., from a Markov Chain) are equally well supported by the dataset.
- H_0: the expected $\delta = L_1 - L_2 = 0$.
  H_A: the expected $\delta = L_1 - L_2 \neq 0$.
  KH assumes that the ML tree is not among the trees.

Kishino-Hasegawa test:

	$D^{(1)}$	$D^{(2)}$	$\cdots$	$D^{(B)}$
T_1	$l_1^{(1)}$	$l_1^{(2)}$	$\cdots$	$l_1^{(B)}$
T_2	$l_2^{(1)}$	$l_2^{(2)}$	$\cdots$	$l_2^{(B)}$

Kishino-Hasegawa test:

$$
\begin{array}{c|cccc}
 & D^{(1)} & D^{(2)} & \cdots & D^{(B)} \\
\hline
T_1 & l_1^{(1)} & l_1^{(2)} & \cdots & l_1^{(B)} \xrightarrow{\text{mean}} \bar{l}_1 \\
T_2 & l_2^{(1)} & l_2^{(2)} & \cdots & l_2^{(B)} \xrightarrow{\text{mean}} \bar{l}_2
\end{array}
$$

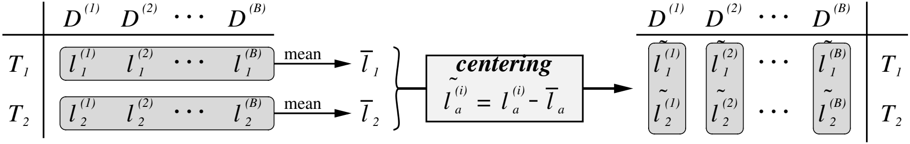
Kishino-Hasegawa test:

	$D^{(1)}$	$D^{(2)}$	$\cdots$	$D^{(B)}$		
T_1	$l_1^{(1)}$	$l_1^{(2)}$	$\cdots$	$l_1^{(B)}$	$\xrightarrow{\text{mean}}$	$\bar{l}_1$
T_2	$l_2^{(1)}$	$l_2^{(2)}$	$\cdots$	$l_2^{(B)}$	$\xrightarrow{\text{mean}}$	$\bar{l}_2$

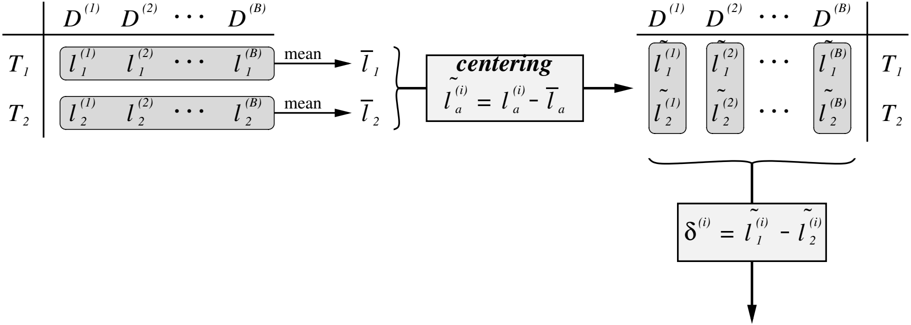
centering

$$\tilde{l}_a^{(i)} = l_a^{(i)} - \bar{l}_a$$

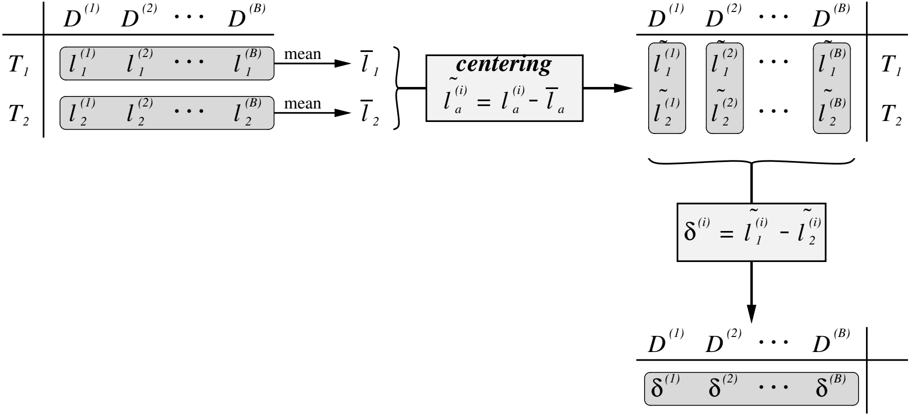
Kishino-Hasegawa test:



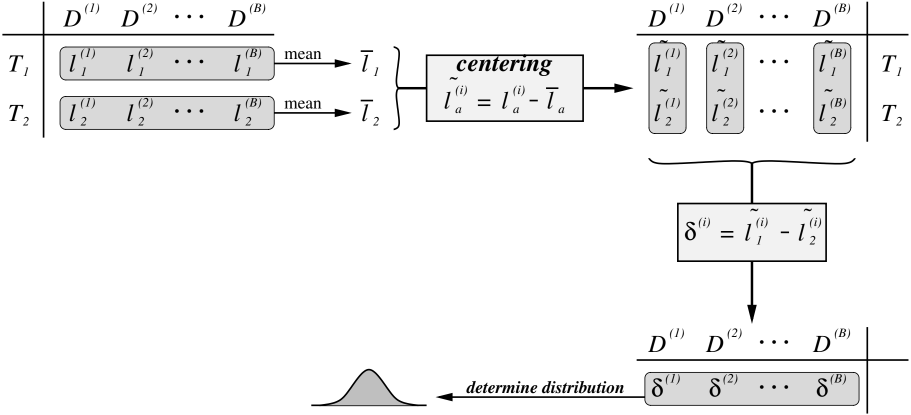
Kishino-Hasegawa test:



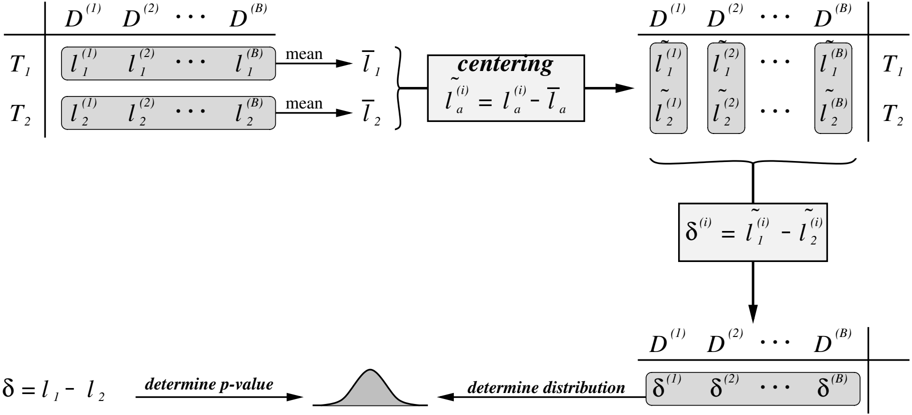
Kishino-Hasegawa test:



Kishino-Hasegawa test:



Kishino-Hasegawa test:



Mis-use of the Kishino and Hasegawa test (KH test)

- Often, instead two *a priory* chosen trees, one tree is tested against the ML tree T_{ML}.

Mis-use of the Kishino and Hasegawa test (KH test)

- Often, instead two *a priory* chosen trees, one tree is tested against the ML tree T_{ML}.
- That means, $\delta = L_{ML} - L_1$ is rarely negative.

Mis-use of the Kishino and Hasegawa test (KH test)

- Often, instead two *a priory* chosen trees, one tree is tested against the ML tree T_{ML}.
- That means, $\delta = L_{ML} - L_1$ is rarely negative.
- Hence, δ has to be tested in a single-sided regime.

Mis-use of the Kishino and Hasegawa test (KH test)

- Often, instead two *a priory* chosen trees, one tree is tested against the ML tree T_{ML}.
- That means, $\delta = L_{ML} - L_1$ is rarely negative.
- Hence, δ has to be tested in a single-sided regime.
- H_0: the expected $\delta = L_1 - L_2 = 0$. H_A: the expected $\delta = L_1 - L_2 > 0$.

Multiple trees (Shimodaira and Hasegawa test - SH test)

- The SH test offers a correct way to test a set of trees, which may be chosen *a posteriory* after ML analysis.

Multiple trees (Shimodaira and Hasegawa test - SH test)

- The SH test offers a correct way to test a set of trees, which may be chosen *a posteriory* after ML analysis.
- H_0: All trees including T_{ML} are equally supported. H_A: Some or all trees T_x are not equally well supported.

Multiple trees (Shimodaira and Hasegawa test - SH test)

- The SH test offers a correct way to test a set of trees, which may be chosen *a posteriory* after ML analysis.
- H_0: All trees including T_{ML} are equally supported. H_A: Some or all trees T_x are not equally well supported.
- The SH test assumes, that the ML tree T_{ML} is among the trees.

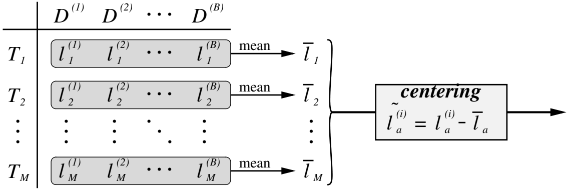
Shimodaira-Hasegawa test:

$$
\begin{array}{c|cccc}
 & D^{(1)} & D^{(2)} & \cdots & D^{(B)} \\
\hline
T_1 & l_1^{(1)} & l_1^{(2)} & \cdots & l_1^{(B)} \\
T_2 & l_2^{(1)} & l_2^{(2)} & \cdots & l_2^{(B)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
T_M & l_M^{(1)} & l_M^{(2)} & \cdots & l_M^{(B)}
\end{array}
$$

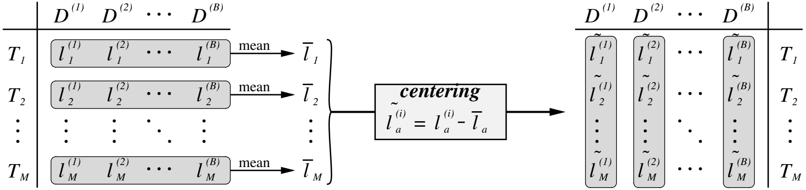
Shimodaira-Hasegawa test:

$$\begin{array}{c|cccccc}
 & D^{(1)} & D^{(2)} & \cdots & D^{(B)} & & \\ \hline
T_1 & l_1^{(1)} & l_1^{(2)} & \cdots & l_1^{(B)} & \xrightarrow{\text{mean}} & \bar{l}_1 \\
T_2 & l_2^{(1)} & l_2^{(2)} & \cdots & l_2^{(B)} & \xrightarrow{\text{mean}} & \bar{l}_2 \\
\vdots & \vdots & \vdots & \ddots & \vdots & & \vdots \\
T_M & l_M^{(1)} & l_M^{(2)} & \cdots & l_M^{(B)} & \xrightarrow{\text{mean}} & \bar{l}_M
\end{array}$$

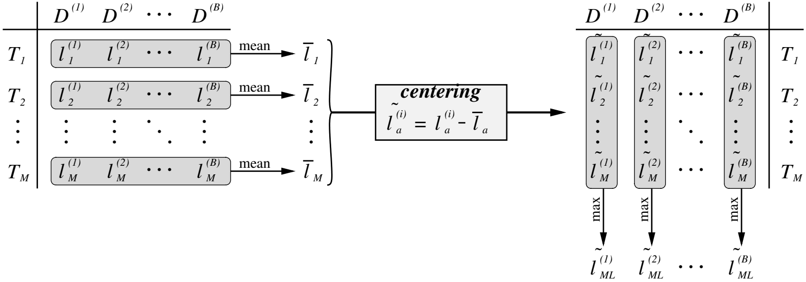
Shimodaira-Hasegawa test:



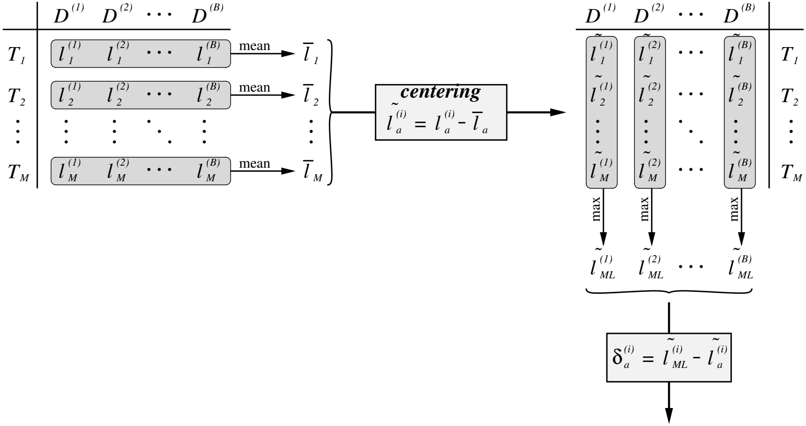
Shimodaira-Hasegawa test:



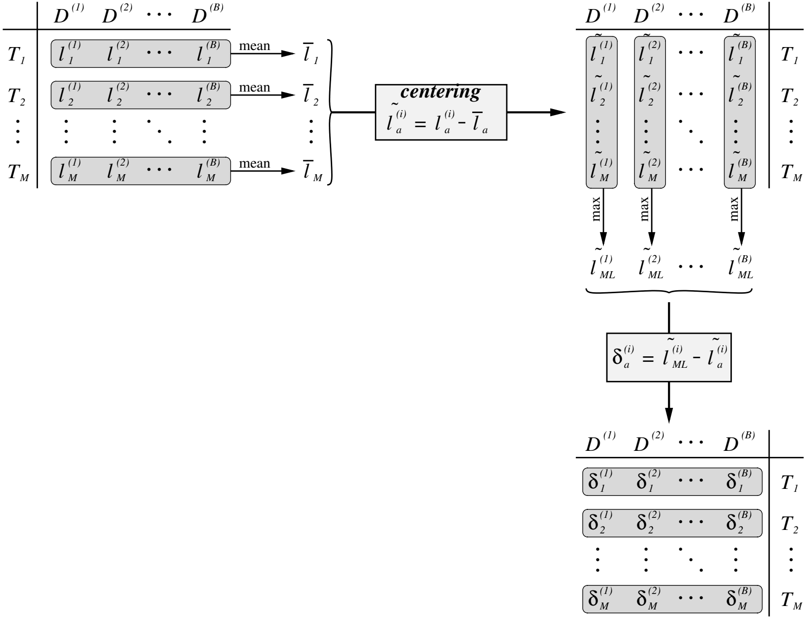
Shimodaira-Hasegawa test:



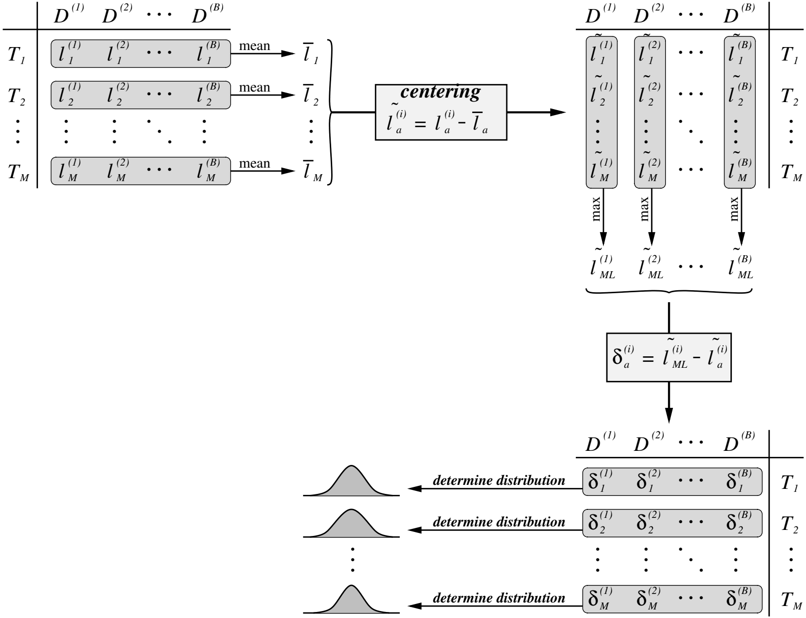
Shimodaira-Hasegawa test:



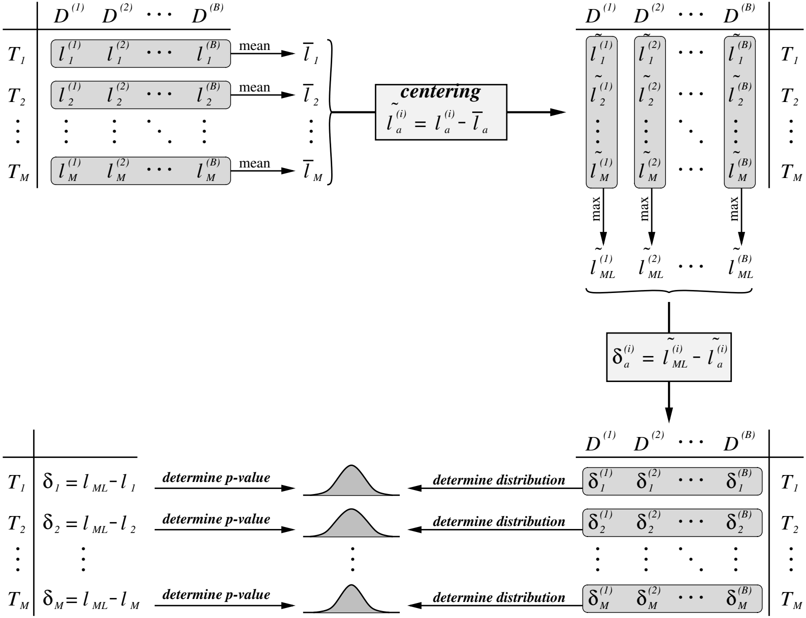
Shimodaira-Hasegawa test:



Shimodaira-Hasegawa test:



Shimodaira-Hasegawa test:



The SH procedure:

- Compute log-likelihood values and the differences $\delta_x = L_{ML} - L_x$ for your trees.

The SH procedure:

- Compute log-likelihood values and the differences $\delta_x = L_{ML} - L_x$ for your trees.
- Draw bootstrap samples i from the alignment (with RELL) to gain log-likelihood values $L_x^{(i)}$ for each tree T_x.

The SH procedure:

- Compute log-likelihood values and the differences $\delta_x = L_{ML} - L_x$ for your trees.
- Draw bootstrap samples i from the alignment (with RELL) to gain log-likelihood values $L_x^{(i)}$ for each tree T_x.
- Adjust the log-likelihoods with the mean over the samples i by setting $\tilde{L}_x^{(i)} = L_x^{(i)} - \bar{L}_x^{(i)}$ (Centering)

The SH procedure:

- Compute log-likelihood values and the differences $\delta_x = L_{ML} - L_x$ for your trees.
- Draw bootstrap samples i from the alignment (with RELL) to gain log-likelihood values $L_x^{(i)}$ for each tree T_x.
- Adjust the log-likelihoods with the mean over the samples i by setting $\tilde{L}_x^{(i)} = L_x^{(i)} - \bar{L}_x^{(i)}$ (Centering)
- For each sample i, find $\tilde{L}_{ML}^{(i)}$ over all topologies T_x.

The SH procedure:

- Compute log-likelihood values and the differences $\delta_x = L_{ML} - L_x$ for your trees.
- Draw bootstrap samples i from the alignment (with RELL) to gain log-likelihood values $L_x^{(i)}$ for each tree T_x.
- Adjust the log-likelihoods with the mean over the samples i by setting $\tilde{L}_x^{(i)} = L_x^{(i)} - \bar{L}_x^{(i)}$ (Centering)
- For each sample i, find $\tilde{L}_{ML}^{(i)}$ over all topologies T_x.
- and compute $\delta_x^{(i)} = \tilde{L}_{ML}^{(i)} - \tilde{L}_x^{(i)}$.

The SH procedure:

- Compute log-likelihood values and the differences $\delta_x = L_{ML} - L_x$ for your trees.
- Draw bootstrap samples i from the alignment (with RELL) to gain log-likelihood values $L_x^{(i)}$ for each tree T_x.
- Adjust the log-likelihoods with the mean over the samples i by setting $\tilde{L}_x^{(i)} = L_x^{(i)} - \bar{L}_x^{(i)}$ (Centering)
- For each sample i, find $\tilde{L}_{ML}^{(i)}$ over all topologies T_x.
- and compute $\delta_x^{(i)} = \tilde{L}_{ML}^{(i)} - \tilde{L}_x^{(i)}$.
- For each tree T_x, test whether δ_x is a plausible sample from the distribution of $\delta_x^{(i)}$ (over all replicates i).

The SH procedure:

- Compute log-likelihood values and the differences $\delta_x = L_{ML} - L_x$ for your trees.
- Draw bootstrap samples i from the alignment (with RELL) to gain log-likelihood values $L_x^{(i)}$ for each tree T_x.
- Adjust the log-likelihoods with the mean over the samples i by setting $\tilde{L}_x^{(i)} = L_x^{(i)} - \bar{L}_x^{(i)}$ (Centering)
- For each sample i, find $\tilde{L}_{ML}^{(i)}$ over all topologies T_x.
- and compute $\delta_x^{(i)} = \tilde{L}_{ML}^{(i)} - \tilde{L}_x^{(i)}$.
- For each tree T_x, test whether δ_x is a plausible sample from the distribution of $\delta_x^{(i)}$ (over all replicates i).
- We use a single sided test, since $\tilde{L}_{ML}^{(i)} \geq \tilde{L}_x^{(i)}$.

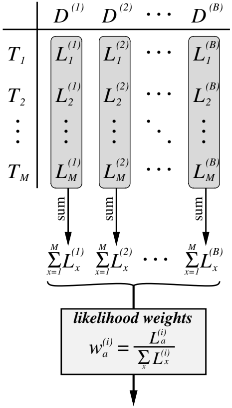
Expected Likelihood weights:

	$D^{(1)}$	$D^{(2)}$	$\cdots$	$D^{(B)}$
T_1	$L_1^{(1)}$	$L_1^{(2)}$	$\cdots$	$L_1^{(B)}$
T_2	$L_2^{(1)}$	$L_2^{(2)}$	$\cdots$	$L_2^{(B)}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
T_M	$L_M^{(1)}$	$L_M^{(2)}$	$\cdots$	$L_M^{(B)}$

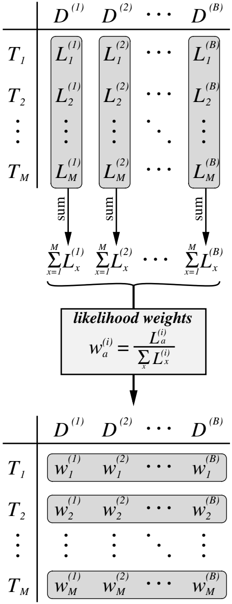
Expected Likelihood weights:

	$D^{(1)}$	$D^{(2)}$	$\cdots$	$D^{(B)}$
T_1	$L_1^{(1)}$	$L_1^{(2)}$	$\cdots$	$L_1^{(B)}$
T_2	$L_2^{(1)}$	$L_2^{(2)}$	$\cdots$	$L_2^{(B)}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
T_M	$L_M^{(1)}$	$L_M^{(2)}$	$\cdots$	$L_M^{(B)}$
	sum $\downarrow$	sum $\downarrow$		sum $\downarrow$
	$\sum_{x=1}^{M} L_x^{(1)}$	$\sum_{x=1}^{M} L_x^{(2)}$	$\cdots$	$\sum_{x=1}^{M} L_x^{(B)}$

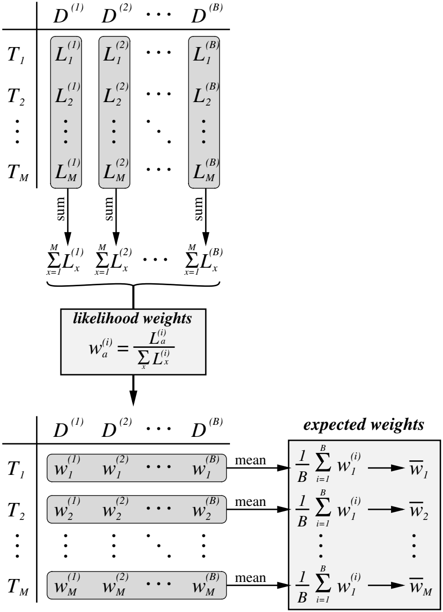
Expected Likelihood weights:



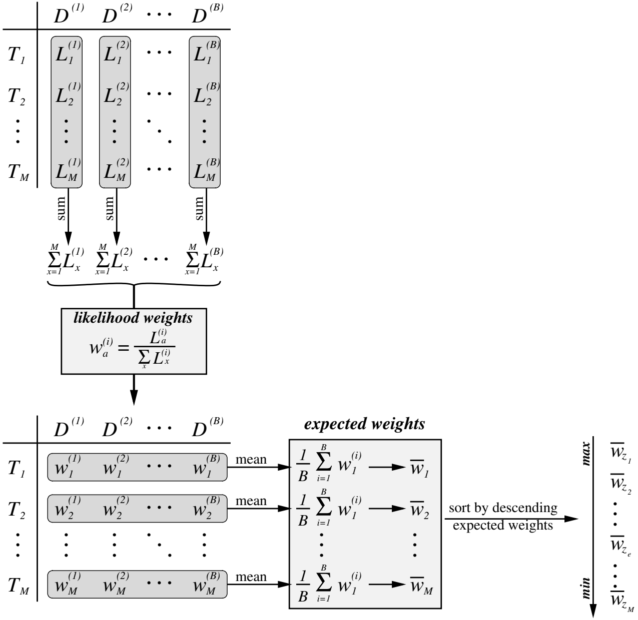
Expected Likelihood weights:



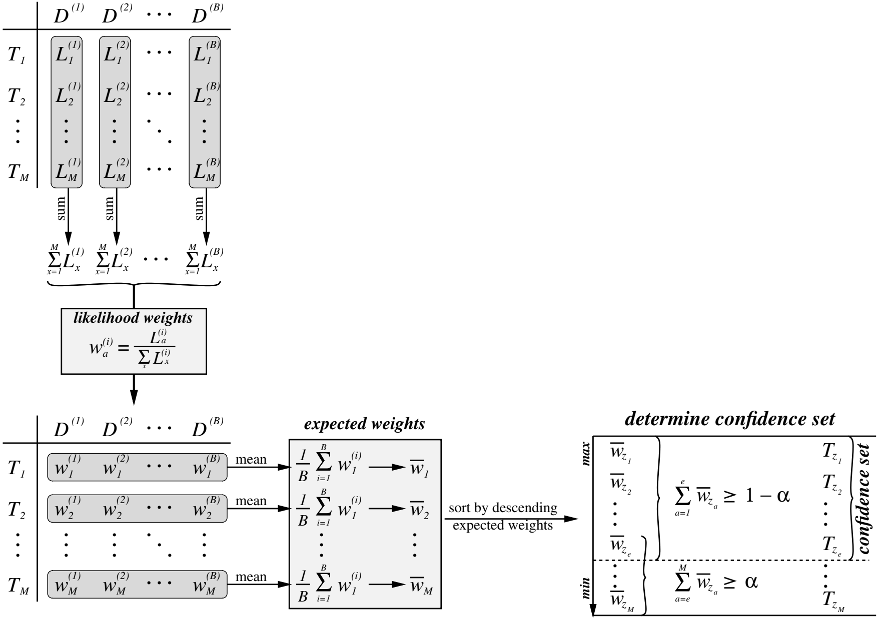
Expected Likelihood weights:



Expected Likelihood weights:



Expected Likelihood weights:



Pros and Cons of various tests:

- **Kishino-Hasegawa test (KH)** – usually mis-used if the trees are not chosen *a priori*.

Pros and Cons of various tests:

- **Kishino-Hasegawa test (KH)** – usually mis-used if the trees are not chosen *a priori*.
- **Shimodaira-Hasegawa (SH)** – test for multiple trees, affected by selection error, i.e., gets more conservative with the number of trees.

Pros and Cons of various tests:

- **Kishino-Hasegawa test (KH)** – usually mis-used if the trees are not chosen *a priori*.
- **Shimodaira-Hasegawa (SH)** – test for multiple trees, affected by selection error, i.e., gets more conservative with the number of trees.
- **Weighted Shimodaira-Hasegawa (wSH)** – SH test weighted with the variance of the likelihood difference (less conservative than SH).

Pros and Cons of various tests:

- **Kishino-Hasegawa test (KH)** – usually mis-used if the trees are not chosen *a priori*.
- **Shimodaira-Hasegawa (SH)** – test for multiple trees, affected by selection error, i.e., gets more conservative with the number of trees.
- **Weighted Shimodaira-Hasegawa (wSH)** – SH test weighted with the variance of the likelihood difference (less conservative than SH).
- **Expected likelihood weights (ELW)** – less conservative than SH, but the impact of model mis-specification unclear.

Pros and Cons of various tests:

- **Kishino-Hasegawa test (KH)** – usually mis-used if the trees are not chosen *a priori*.
- **Shimodaira-Hasegawa (SH)** – test for multiple trees, affected by selection error, i.e., gets more conservative with the number of trees.
- **Weighted Shimodaira-Hasegawa (wSH)** – SH test weighted with the variance of the likelihood difference (less conservative than SH).
- **Expected likelihood weights (ELW)** – less conservative than SH, but the impact of model mis-specification unclear.
- **Approximately unbiased test (AU)** – fixes the conservativeness issue of SH, but many similarly good trees can lead to artificial over-confidence.

Summary

- Likelihoods gives a strong statistical framework for hypothesis testing.

Summary

- Likelihoods gives a strong statistical framework for hypothesis testing.
- Proper experimental design and proper use of tests is required.

Summary

- Likelihoods gives a strong statistical framework for hypothesis testing.
- Proper experimental design and proper use of tests is required.
- One should alway be aware of the hypothesis a test assesses and should make sure that this answers the question asked.

Summary

- Likelihoods gives a strong statistical framework for hypothesis testing.
- Proper experimental design and proper use of tests is required.
- One should alway be aware of the hypothesis a test assesses and should make sure that this answers the question asked.
- Testing tree topologies can be used to assess whether two competing hypotheses are really substantially different. If they are not, one cannot be prefered over the other.

Exercises:

the exercises can be found at

`http://www.cibiv.at/~hschmidt/VEME/ML-test`