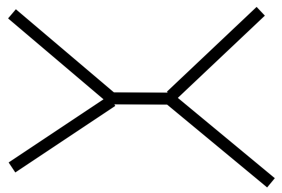
Distance Methods

Overview

```
acca
gcca
gcct
tgca
```

	1	2	3	4
1				
2				
3				
4				



Step 1:
Construct distance matrix

Step 2:
Build tree

1: Sequences to Distances

Can use a model (e.g., PAM) to compute evolutionary distances

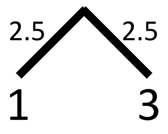
Distances to Trees

- Many different approaches:
  - Iterative/greedy (UPGMA, neighbour-joining)
  - Optimization (Fitch, minimum evolution)

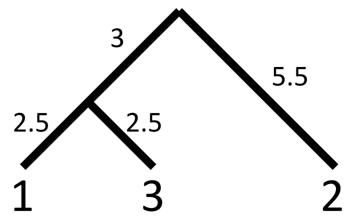
UPGMA again

Unweighted Pair Grouping with Arithmetic Mean

	1	2	3
1			
2	10		
3	5	12	



	1+3	2
1+3		
2	11	



Assumes a molecular clock
(distances from the root to all leaves will be EQUAL)

Neighbor-joining (Saitou and Nei 1987)

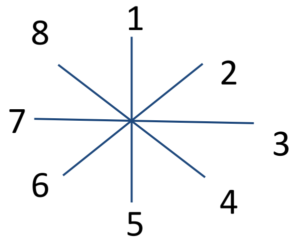
Start with a 'star' tree

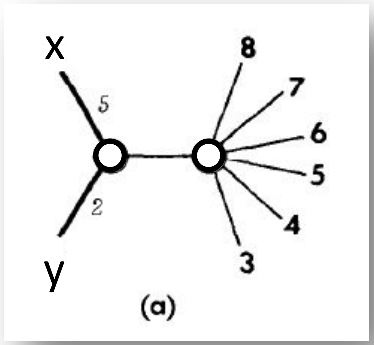
At each iteration, split off the pair of taxa that minimizes the total sum of branch lengths in the tree

Choose groups x and y to minimize the **Q-criterion**:

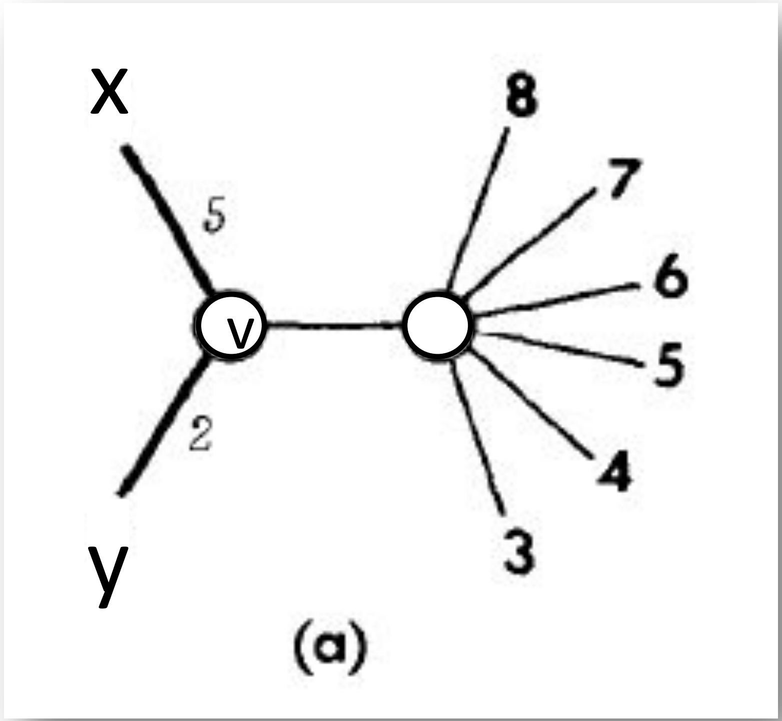
$$\boxed{\delta(x,y)} - \underbrace{\frac{1}{(n-2)}\sum_z \delta(x,z) - \frac{1}{(n-2)}\sum_z \delta(y,z)}_{\text{Weighted distance to all leaves}}$$

Distance matrix entry for (x,y)





This splitting creates a new internal node, v, and assigns x and y as sisters in the growing tree



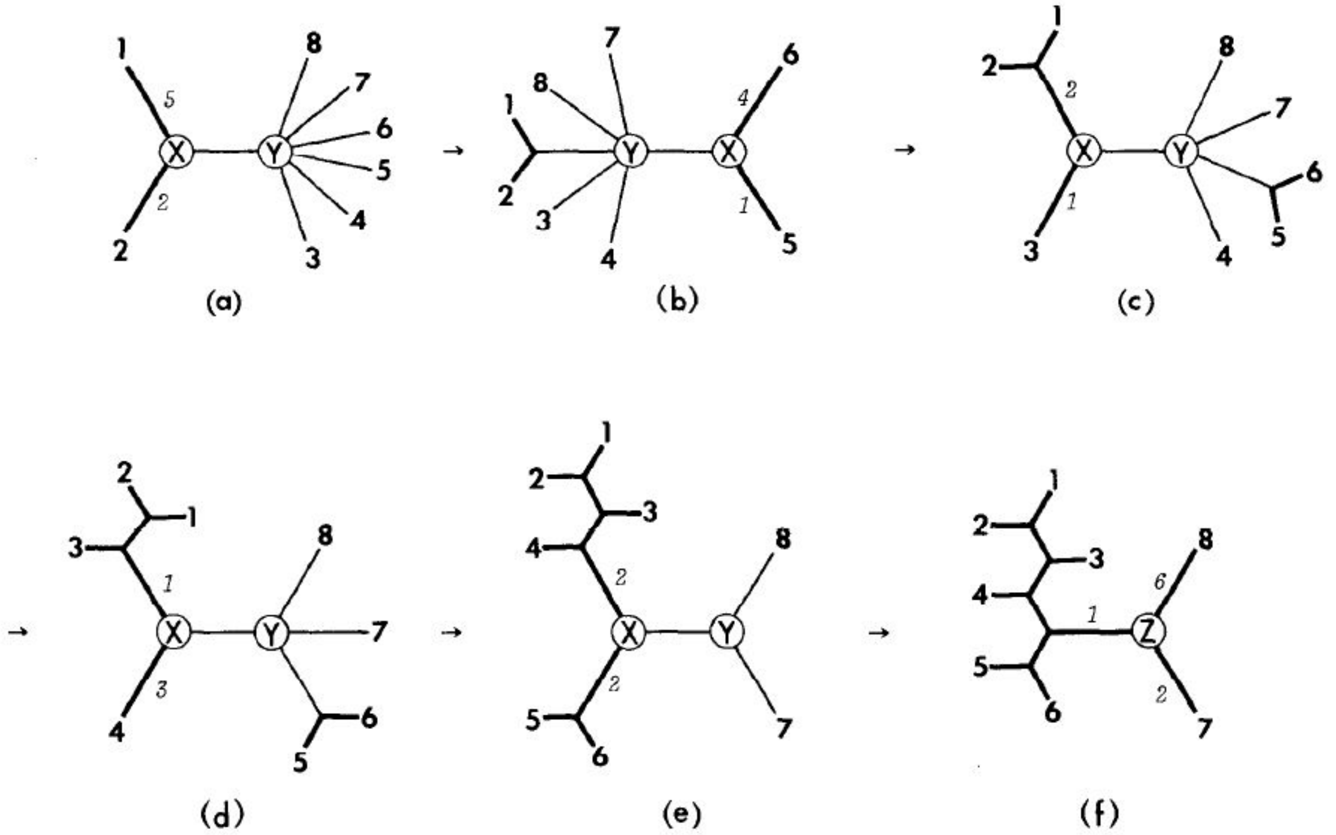
REDUCTION STEP: Recompute distances from all leaves to node v to allow subsequent computations of the Q criterion

$$\delta'(u, v_{xy}) = \frac{1}{2}(\delta(u,x) + \delta(u,y) - \delta(x,y))$$

And assign branch lengths x-v and y-v

$$b_x = \frac{1}{n-2} \sum_{z \neq x,y} (\delta(x,z) + \delta(x,y) - \delta(y,z))$$

Continue until binary tree is obtained



Figures from Saitou and Nei (1987)
Formulas from Bryant, *J Classific* (2005)

Neighbor-joining vs. UPGMA

- Neighbor-joining uses a somewhat less intuitive optimality criterion **Q**

- However, it is still iterative and still fast

- Another advantage is that it does not assume a molecular clock – branch lengths are assigned based on **all** distances in the matrix

Advantages of Distance Methods

- Explicit modelling of residue changes
- Can be very FAST – neighbour-joining can build trees with thousands of leaves

Disadvantages of Distance Methods

- A considerable amount of information is lost when sequence pairs are replaced with a single distance
- Greedy methods may perform poorly for some problems

Conclusion

- **Parsimony**: Character-based, model-free
  - tree search required
- **Distance**: Pairwise distances, can use a model
  - Greedy approaches or iterative searches
- Is there a way to use models without collapsing each pair of sequences to a single distance value? yes

Maximum Likelihood

The story so far

Parsimony: nice and simple

- Too simple!
- “Model free” / ignores data

Distance: nice and fast

- Can be applied to any distance matrix (not necessarily genetic distances)
- Model-based, fast
- Uses every alignment column to generate distances

Parsimony is **inconsistent**

- As we add data, a method should *converge* on the correct answer
- With parsimony, more data can often reinforce an **incorrect** conclusion
- The long-branch attraction problem is an example of this

Likelihood

- If we can specify a model ϒ of evolution, then we can calculate the probability that the data were generated under ϒ

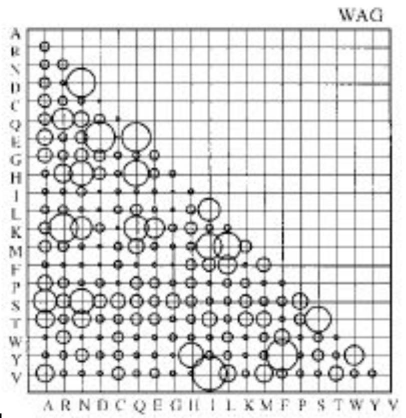
- The probability of the **data**, given the **model**, is the <u>likelihood</u>

What Data?

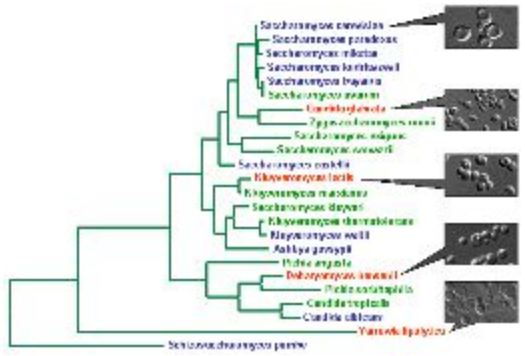
The sequence alignment (our genes or proteins of interest)

What model?

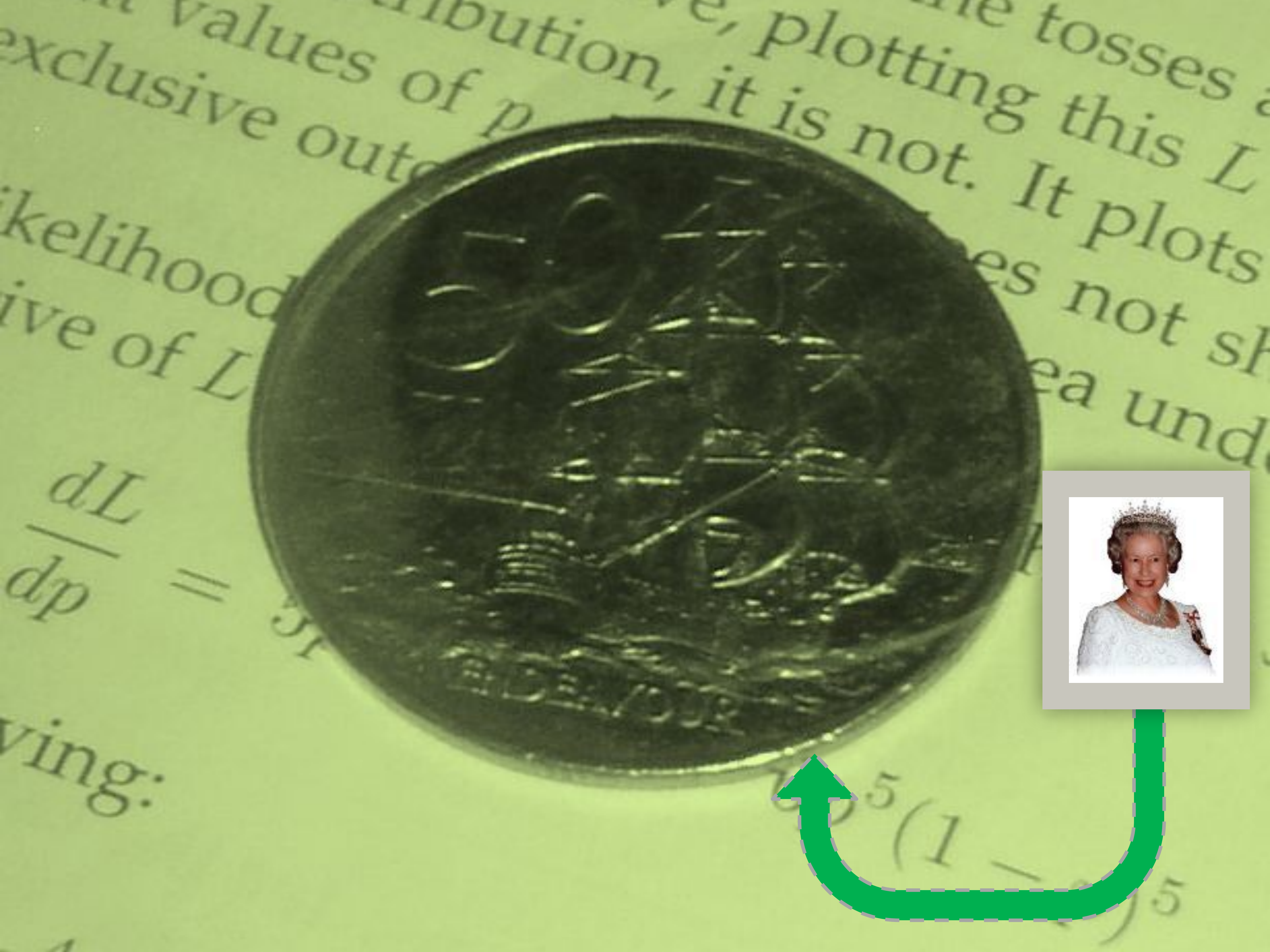
- At least three components:
  - A substitution model



  - Topology <u>and</u> branch lengths in a tree



From *Genolevures* project


values of distribution, plotting this tosses
exclusive of p it is not. It plots
likelihood does not sh
ive of L area unde
$\frac{dL}{dp} =$
ving:
$p^5(1-p)^5$
ENDEAVOUR

Coin-toss likelihoods

One model parameter (probability of ship)
= 1 – (probability of Queen Elizabeth)

We need **data** (proportion of throws that came up ship)

What is the p(ship)?

Formula

$$L = p(D \mid p(ship) = x) = \binom{\#trials}{\#ships} \times p(ship)^{\#ships} \times p(queen)^{\#queens}$$

Concrete example: 10 throws, 6 ships, 4 Queens
what is L(p(ship) = 0.4)?

$$L = p(D \mid p(ship) = 0.4) = \binom{10}{4} \times 0.4^6 \times 0.6^4 = 0.1115$$

what is L(p(ship) = 0.6)?

$$L = p(D \mid p(ship) = 0.6) = \binom{10}{6} \times 0.6^6 \times 0.4^4 = 0.2508$$

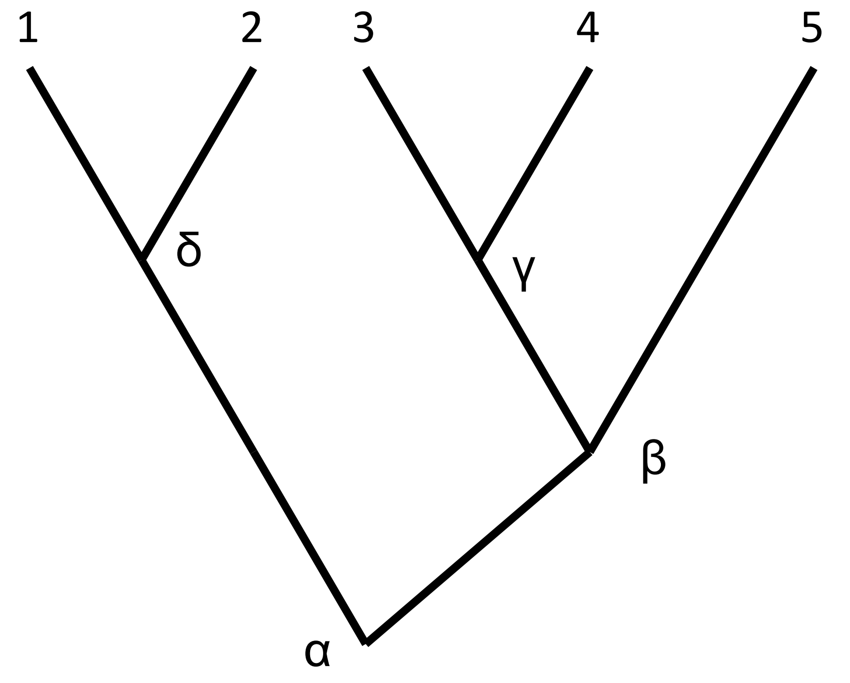
0.6 is the maximum likelihood estimate of p(ship), given these data

Likelihood of an alignment, given ϒ

If we assume independence of each character (alignment column), then we can compute the likelihood separately for each column and multiply the results together

So column order doesn't really matter (kinda like in the language example)

Computing the likelihood for a given column



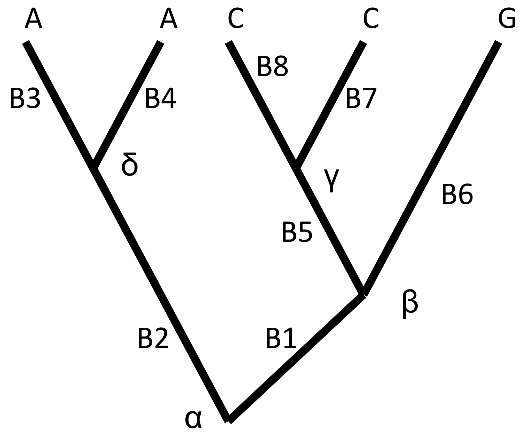
1	A
2	A
3	C
4	C
5	G

$$P(Data \mid T) = \sum_{\alpha} \sum_{\beta} \sum_{\gamma} \sum_{\delta} P(A, A, C, C, G, \alpha, \beta, \gamma, \delta \mid T)$$

Huh?

$$P(Data \mid T) = \sum_{\alpha} \sum_{\beta} \sum_{\gamma} \sum_{\delta} P(A, A, C, C, G, \alpha, \beta, \gamma, \delta \mid T)$$

Means we need to sum over all probabilities (4 nucleotides or 20 amino acids) at every internal node



$= P(\alpha = A) \times P(\beta = A \mid \alpha = A, B_1) \times \ldots$
$+ P(\alpha = C) \times P(\beta = A \mid \alpha = C, B_1) \times \ldots$
…

4^4 terms!

What is $P(\beta = C \mid \alpha = A, B_1)$???

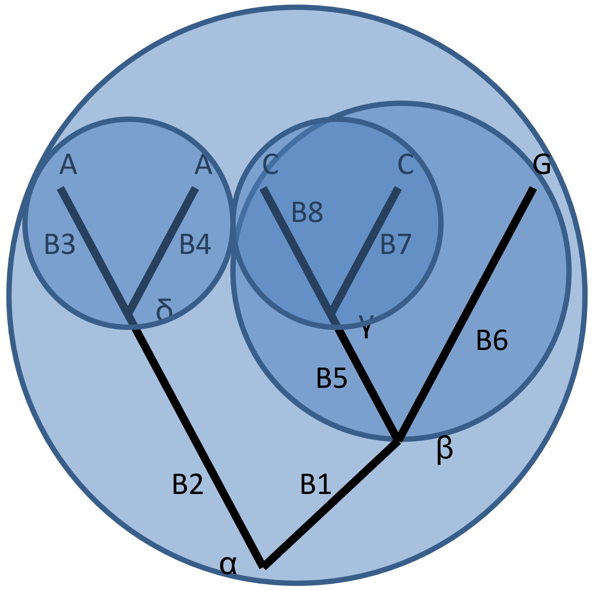
- B_1 is the branch length (in substitutions per site)

- Our substitution model defines the probability of observing a substitution from A to C over a branch of a given length

- A matrix like PAM needs to be converted into an *instantaneous rate matrix* **Q**, which accounts for residue frequencies

- Then $P(C,A|B_1) = {e^{\mathbf{Q}B_1}}_{C,A}$

Felsenstein's likelihood algorithm



Dynamic Programming yet again
Start at the tips, and work backward through the tree

Previous method was b^{n-1} operations
    b = # of bases (alphabet size)
    n = # of taxa
DP method requires $(n - 1)b^2$ operations

Reuse computed likelihoods on each branch, rather than recomputing them every time

Substitution matrices

4x4 nucleotide matrices are typically inferred with the data, along with the tree

Different degrees of freedom:

- Jukes-Cantor (all rates equal)
- Kimura two-parameter (transitions vs. transversions)
- Felsenstein 84 model (different nuc frequencies)
- General time reversible

	A	C	G	T
A				
C	🟥			
G	🟧	🟨		
T	🟩	🟦	🟪	

Substitution matrices

20 x 20 amino acid matrices are usually predefined (*empirical* substitution matrices)

Examples: PAM, JTT, BLOSUM, VT, WAG, LG – different source datasets and counting techniques

Why don’t we do amino acid GTR?

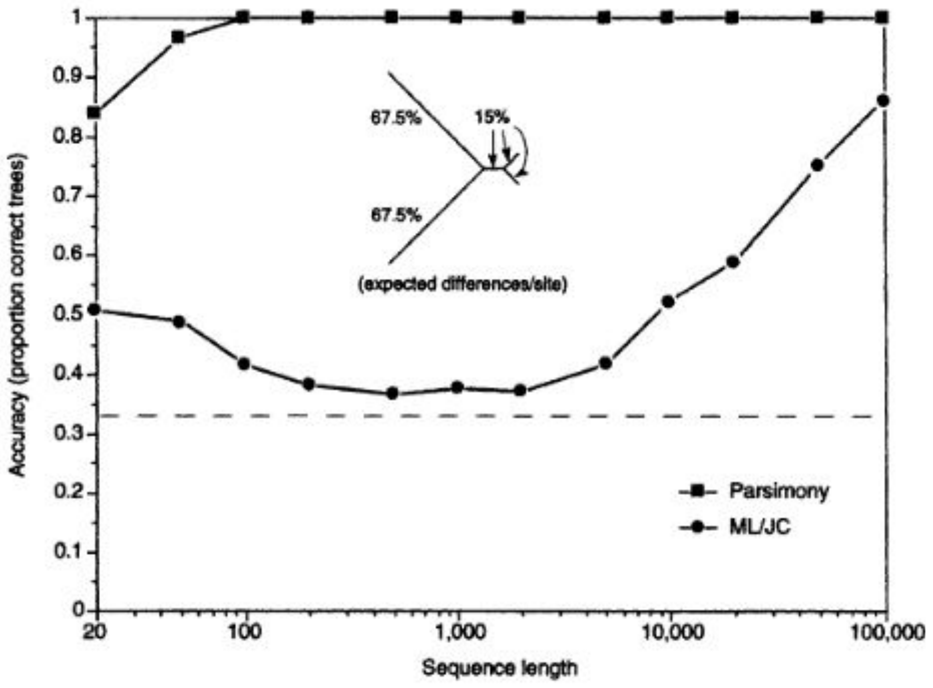
Maximum Likelihood

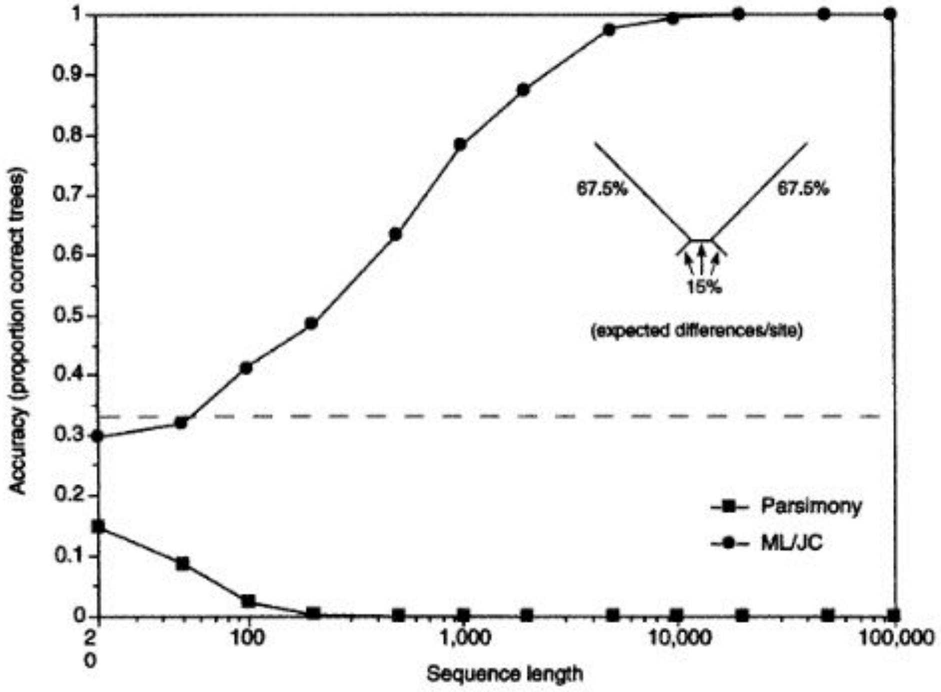
- Given an alignment, find the set of parameter values that maximize L
- As with parsimony, we need to perform a search through tree space
- But now, in addition to considering the tree <u>shape</u>, we must add <u>branch lengths</u> and <u>substitution probabilities</u> to the model

How do those distance methods work again?

Likelihood vs. Parsimony

Accuracy under two different tree shapes (simulated data)





Parsimony does *really well* when long branches are together in the tree

Parsimony is *awful* when long branches are separate in the tree

Swofford et al,. *Systematic Biology*, 2001

What’s going on?

- **Convergent substitutions:**
  - Long branches will have many changes
  - Some of these changes will converge by chance!
  - Parsimony consequently sees these sequences as being more similar than they really are

= **Long-branch attraction**

The key difference…

- In parsimony we consider only the best internal states of the tree

- Whereas in likelihood calculations, all possible internal states are modeled

$= P(\alpha = A) \times P(\beta = A \mid \alpha = A, B_1) \times$

…

$+ P(\alpha = C) \times P(\beta = A \mid \alpha = C, B_1) \times$

…

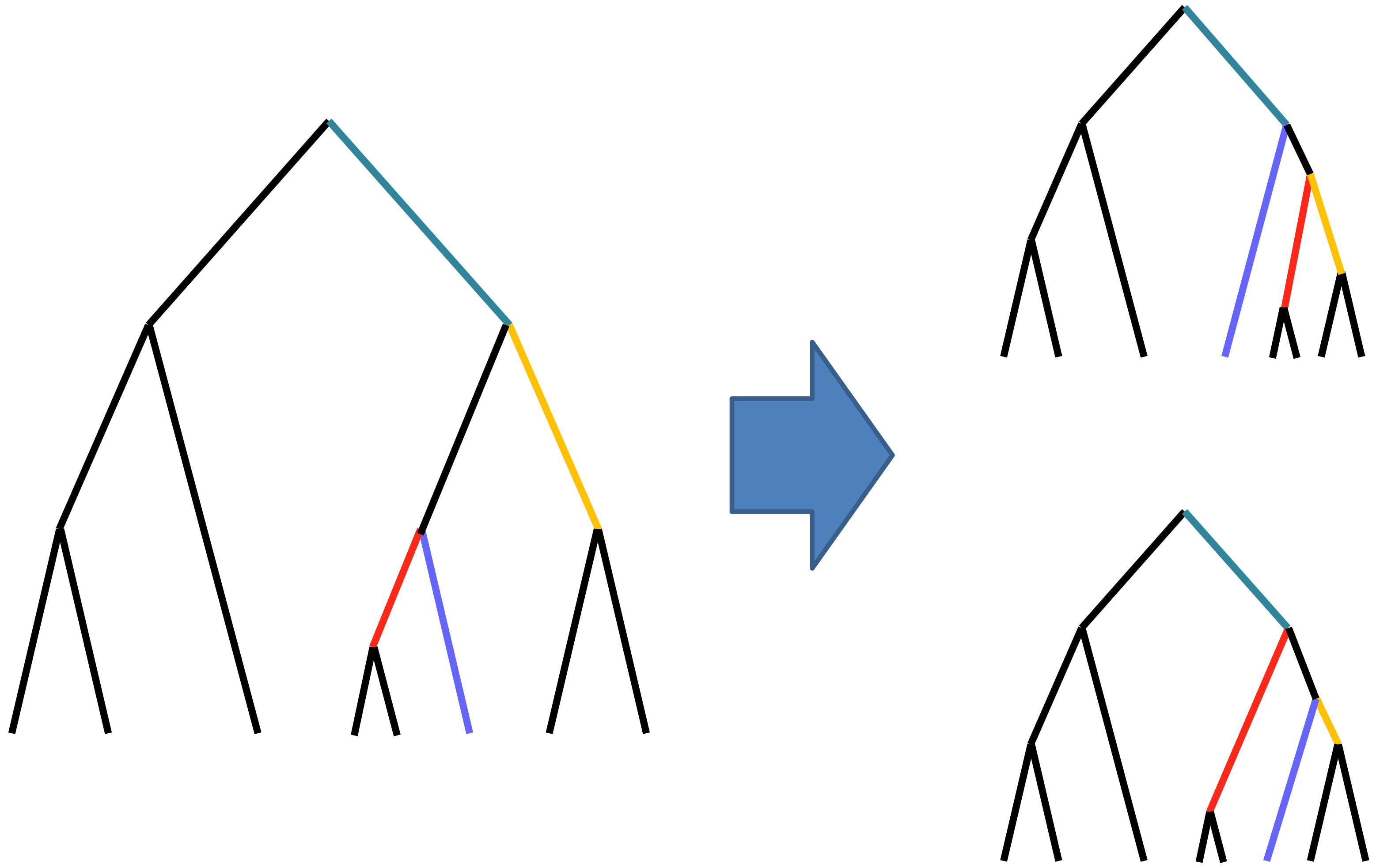
Maximum Likelihood in practice

- Not only do we need to find the best tree shape, we must also optimize the branch lengths
- Heuristics are desperately needed!

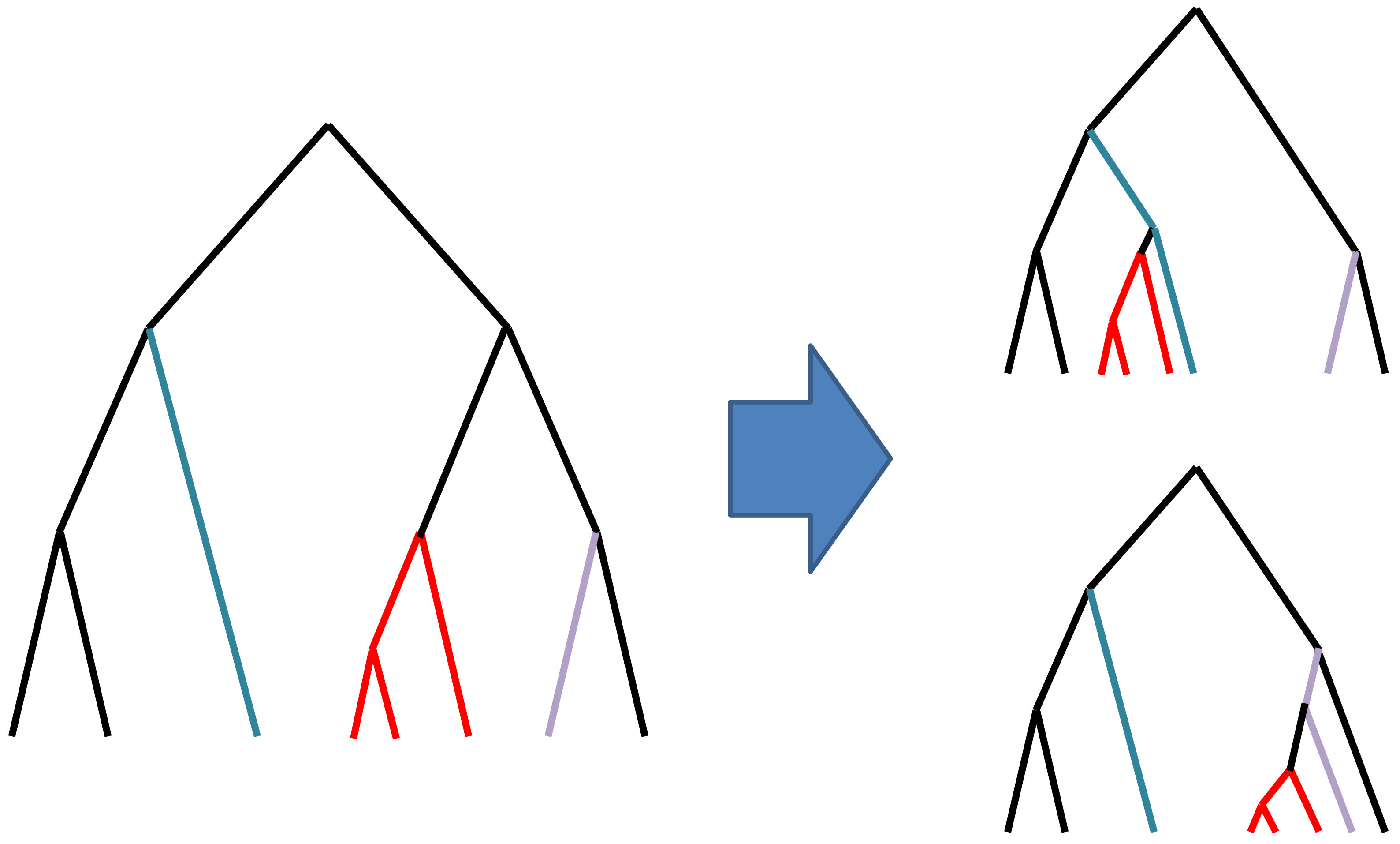
Searching through tree space

- We need techniques to *permute* the tree at every step
- Different permutations induce smaller or larger changes in the tree topology

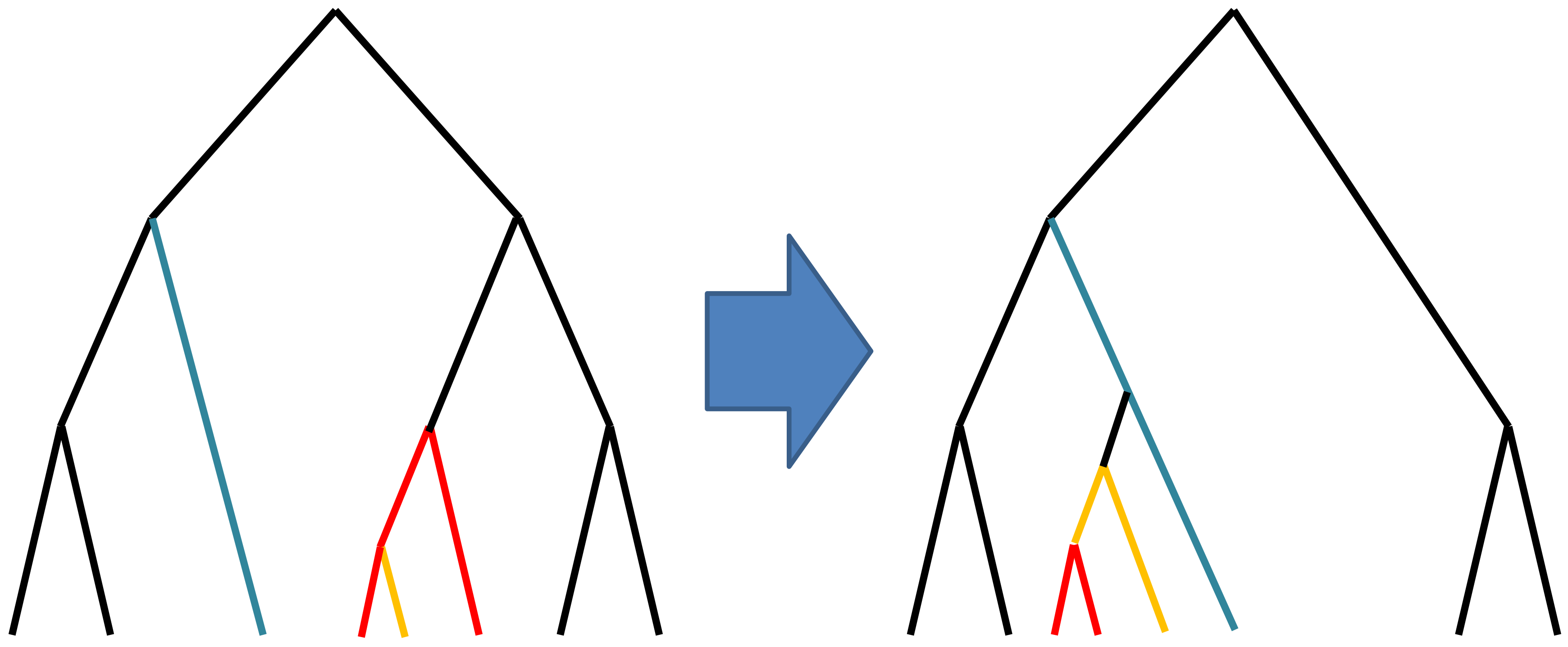
Nearest-neighbour interchange (NNI)



Subtree Prune and Regraft (SPR)



Tree bisection and reconnection



Thoughts on which is best for searching tree space?

Key questions in ML tree finding

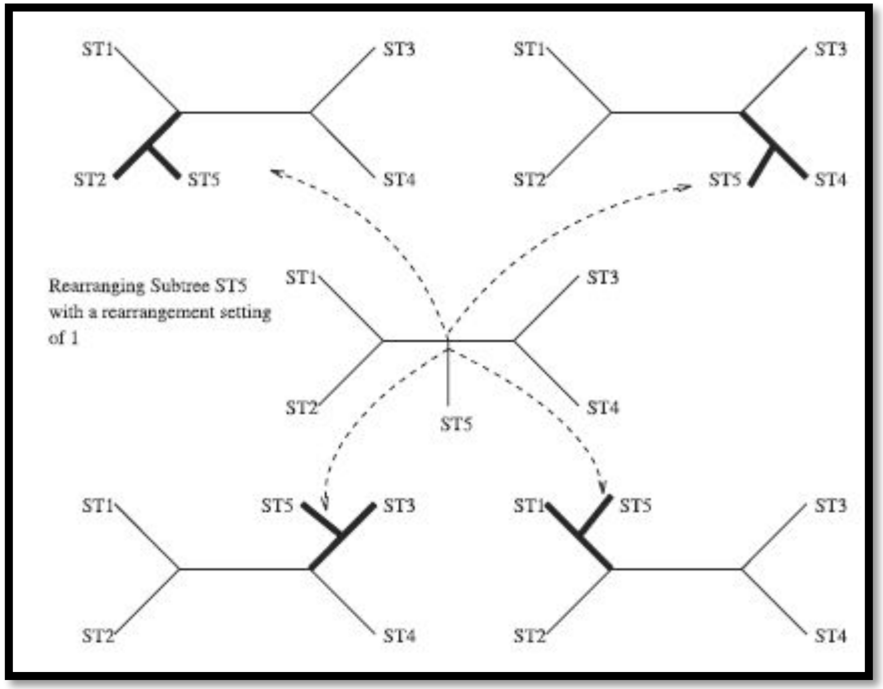
- Where do we start?
- What search strategy do we use?
- When do we optimize branch lengths?
- When do we stop?

RAxML: Fancy Tree Searching

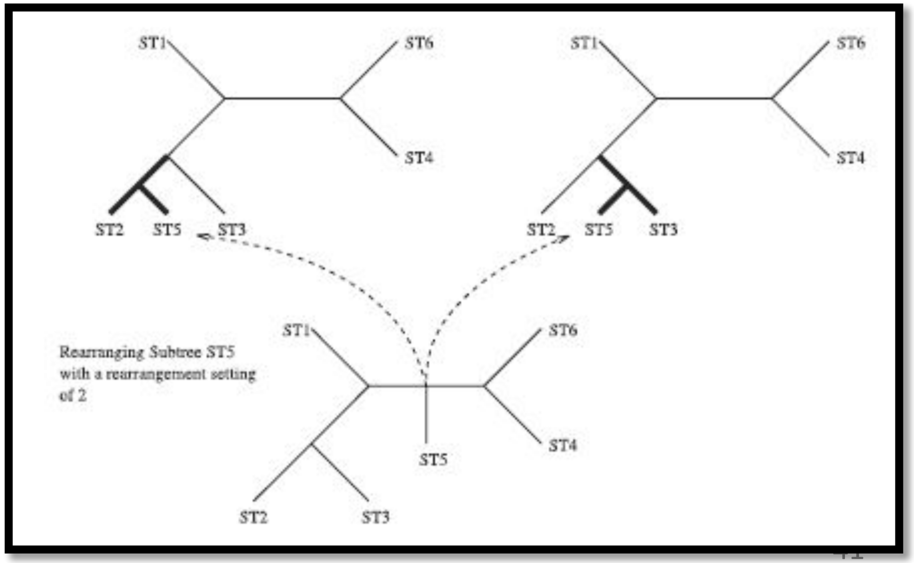
- Starting tree: stepwise addition, maximum parsimony (fast!)

- Tree search procedure:
  - Starting tree
  - Constrained SPR, where each subtree is moved between *Rmin* and *Rmax* steps along the tree

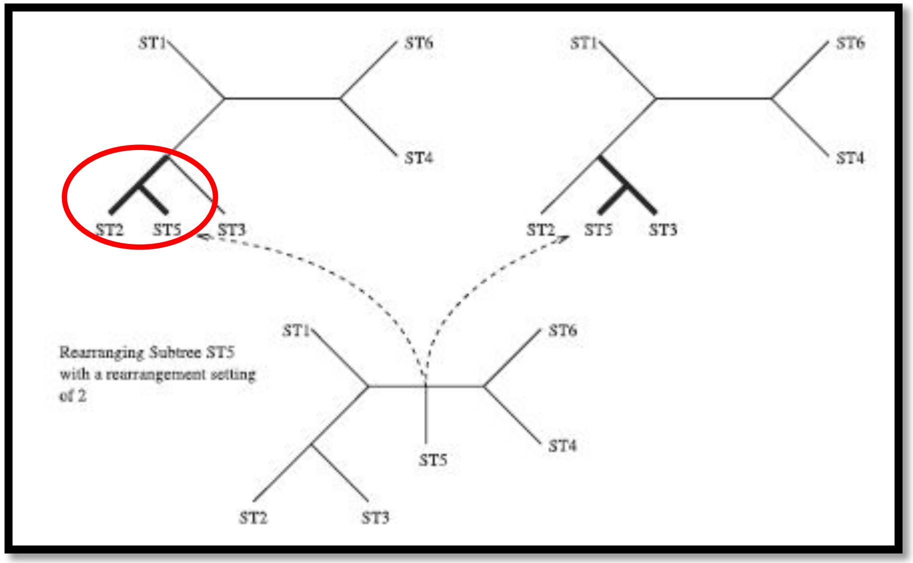
1



2



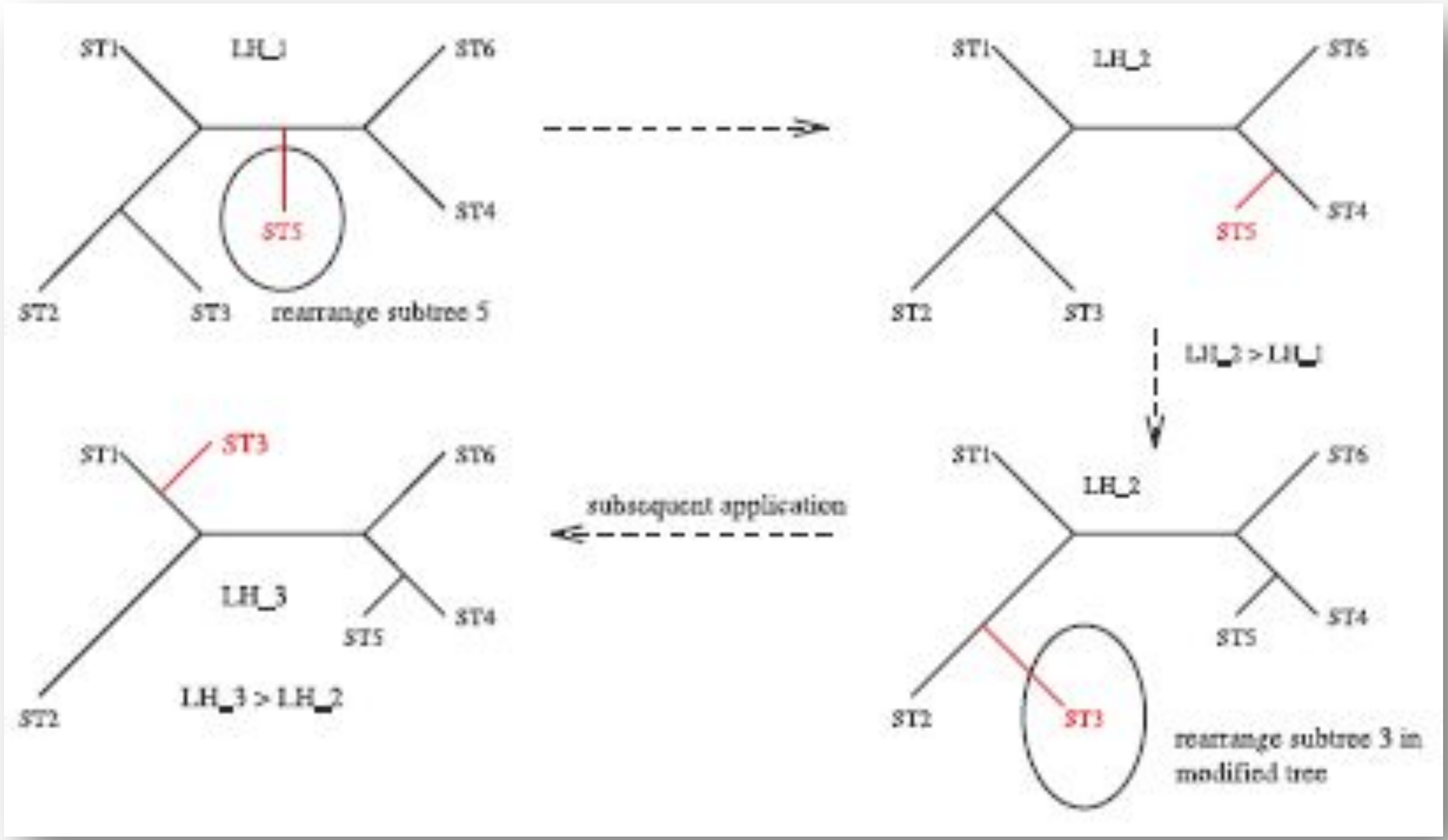
During the complete subtree search, only optimize the branch lengths that are directly implicated in the swap



- Rank all of the resulting trees based on their likelihood

- Choose the top 20 (?!?) for full branch length optimization

Short Circuit



Stopping conditions

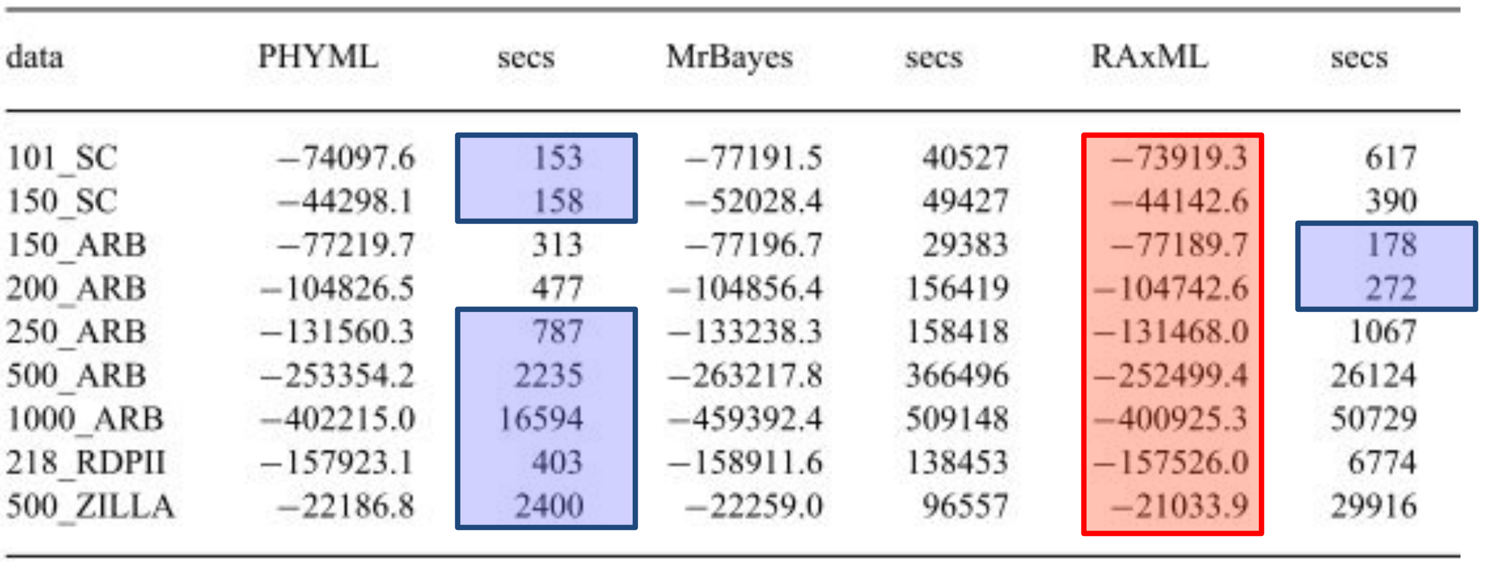
- Set a maximum value for *Rmax*

- If the tree does not improve during an iteration, increment *Rmin* and *Rmax*

- When *Rmax* = max(*Rmax*), stop!

Performance comparison

Stamatakis et al. (2005)
Bioinformatics



data	PHYML	secs	MrBayes	secs	RAxML	secs
101_SC	−74097.6	153	−77191.5	40527	−73919.3	617
150_SC	−44298.1	158	−52028.4	49427	−44142.6	390
150_ARB	−77219.7	313	−77196.7	29383	−77189.7	178
200_ARB	−104826.5	477	−104856.4	156419	−104742.6	272
250_ARB	−131560.3	787	−133238.3	158418	−131468.0	1067
500_ARB	−253354.2	2235	−263217.8	366496	−252499.4	26124
1000_ARB	−402215.0	16594	−459392.4	509148	−400925.3	50729
218_RDPII	−157923.1	403	−158911.6	138453	−157526.0	6774
500_ZILLA	−22186.8	2400	−22259.0	96557	−21033.9	29916

Log-likelihoods
(closer to 0 = better)

Running times

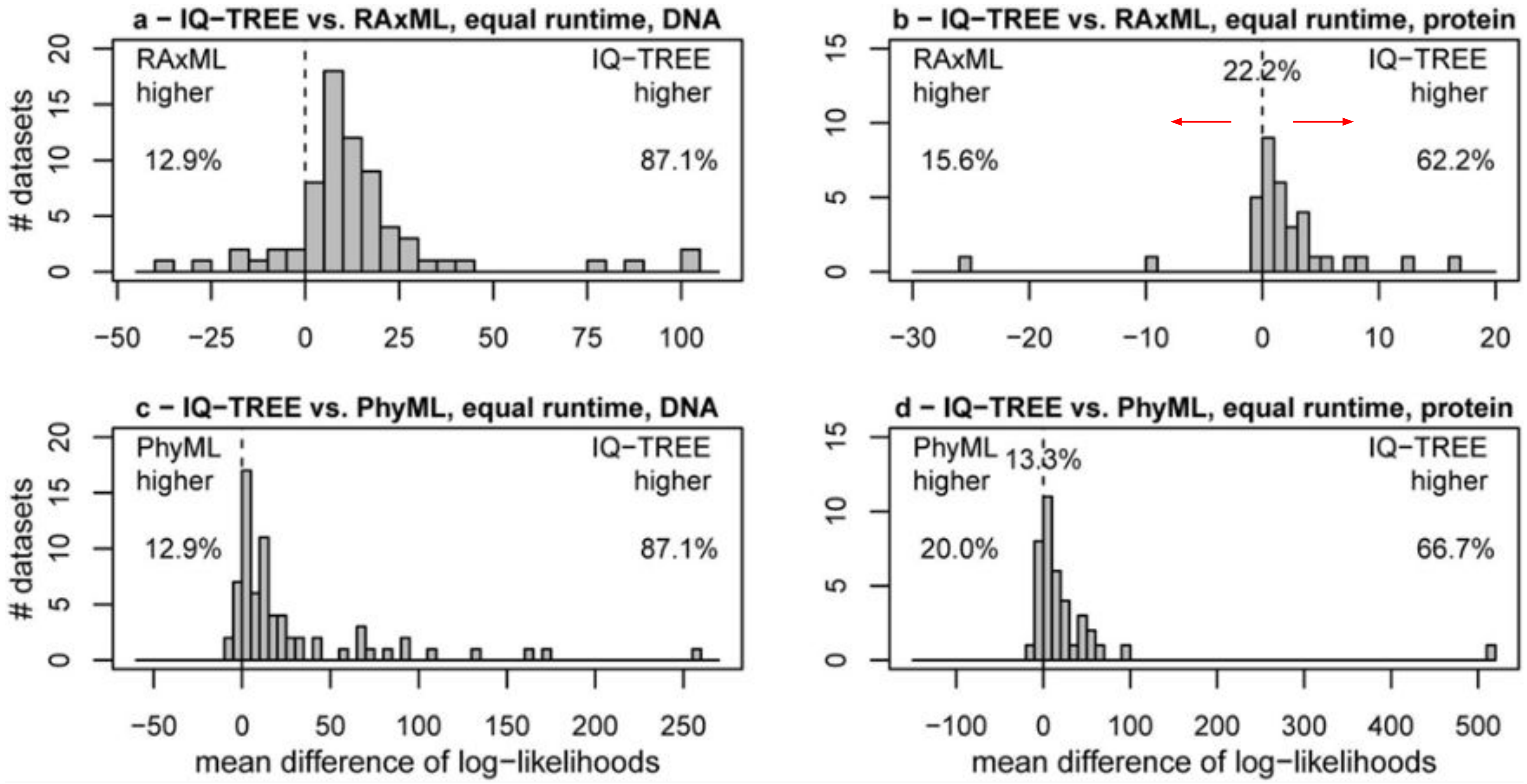
Why RAxML works

- The tree search is a compromise between a narrow, precise search and a broader search
- Only optimize when you need to
- Other stuff: different available models, parallelization, etc.

Cut material: ProML, MetaPIGA

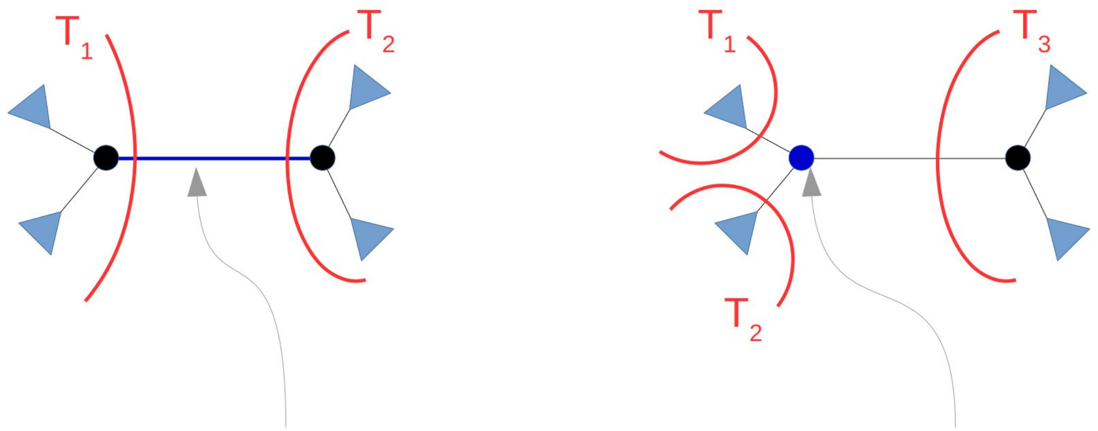
IQ-TREE (Nguyen et al., 2014)

- Key differences with RAxML:
  - Use 100 starting parsimony trees (rapidly inferred, avoid local optima)
  - Filter filter filter!! Optimize branch lengths using ML, purge, then *really* optimize the top 5 trees
  - Perturb these trees with a bunch of random NNIs, re-optimize
  - Stop if 100 rounds of this yield no improvement


a – IQ–TREE vs. RAxML, equal runtime, DNA
RAxML higher
IQ–TREE higher
12.9%
87.1%
# datasets
b – IQ–TREE vs. RAxML, equal runtime, protein
RAxML higher
IQ–TREE higher
22.2%
15.6%
62.2%
c – IQ–TREE vs. PhyML, equal runtime, DNA
PhyML higher
IQ–TREE higher
12.9%
87.1%
# datasets
mean difference of log–likelihoods
d – IQ–TREE vs. PhyML, equal runtime, protein
PhyML higher
IQ–TREE higher
13.3%
20.0%
66.7%
mean difference of log–likelihoods

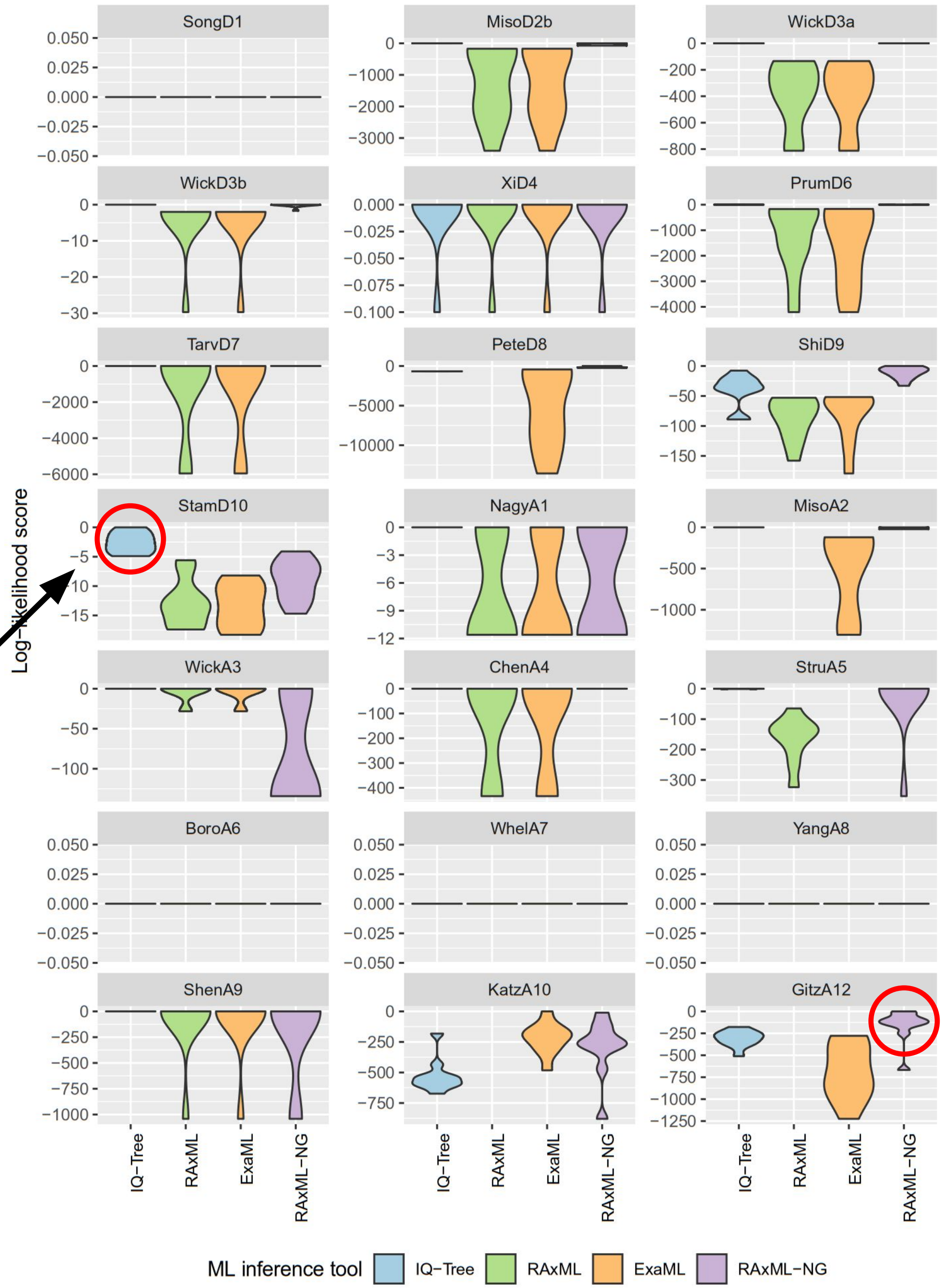
RAxML NG

RAxML/ExaML	RAxML-NG



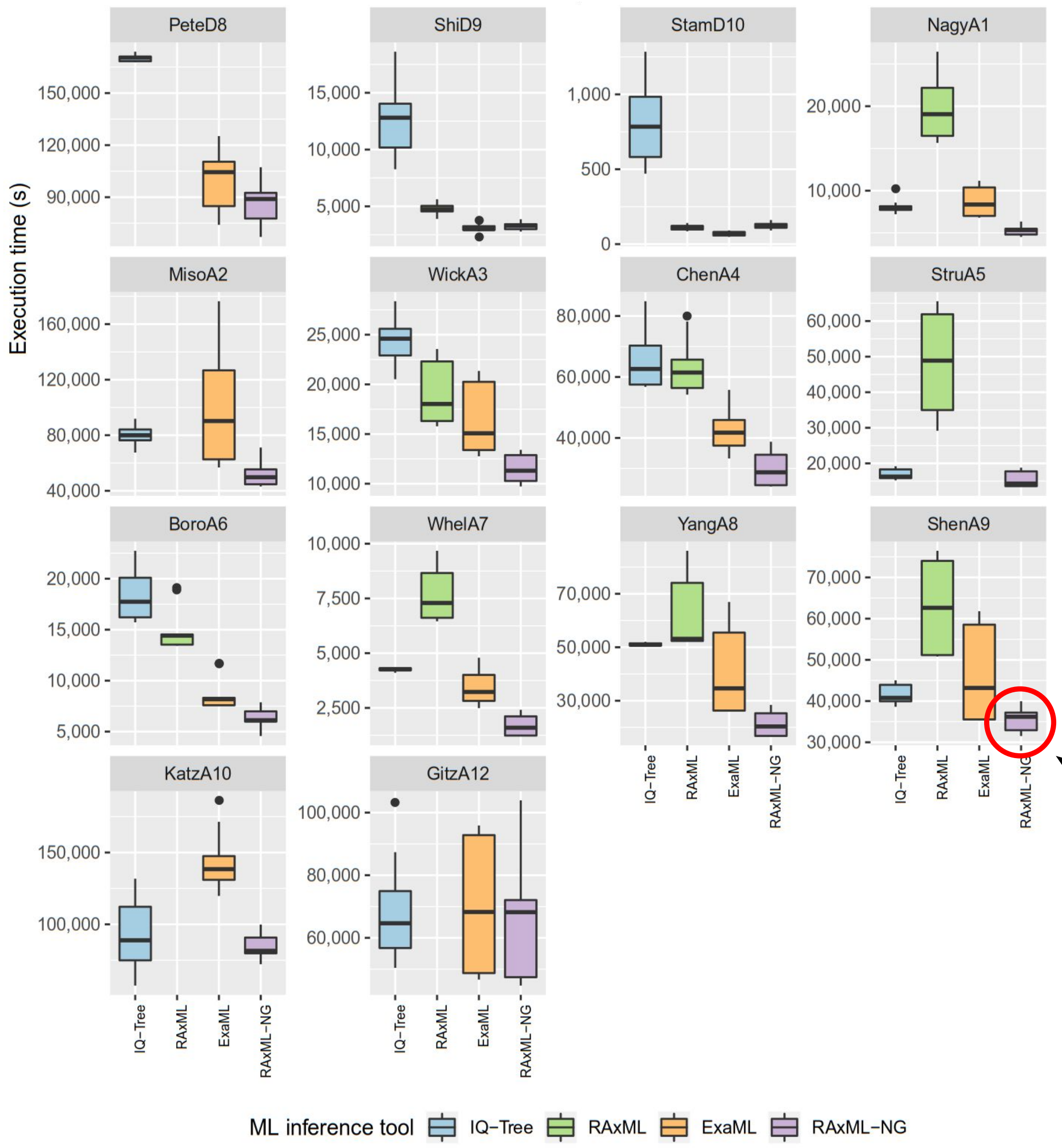
RAxML/ExaML	RAxML-NG
1) Prune, regraft and score two subtrees adjacent to each internal branch	1) Prune, regraft and score three subtrees adjacent to each internal node
$T_1 \rightarrow L(T_1)$ $T_2 \rightarrow L(T_2)$	$T_1 \rightarrow L(T_1)$ $T_2 \rightarrow L(T_2)$ $T_3 \rightarrow L(T_3)$
2) Select best-of-pair: $T_{best} = \underset{\{T_1, T_2\}}{\operatorname{argmax}} L(T)$	2) Consider each subtree individually
3) Store T_{best} in the global list of promising moves → at most 1 subtree per branch!	3) Store up to 3 subtrees per node in the list of promising moves

Further optimization of likelihood kernels
Better parallelization
Kozlov et al. (2019) *Bioinformatics*



IQ-Tree better

RAxML-NG better



RAxML-NG faster

Figure 4: Wall-clock execution times in seconds (16 threads / 1 compute node).

Summary

- Likelihood gives you the best of both worlds: model-based tree construction, and consideration of every character

- Likelihood-based methods are very time consuming, and imperfect heuristics are needed