Phylogenetics and phylogenomics: sequences, trees, networks and other combinatorial objets

Céline Scornavacca

ISE-M, Equipe Phylogénie & Evolution Moléculaires

May 13th, 2014

From Aristotle to Darwin

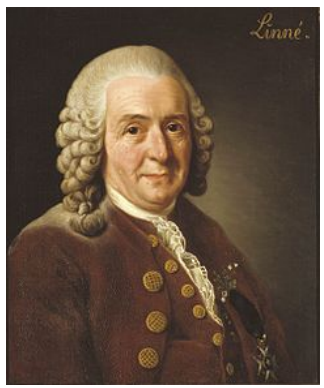
Since Aristotle, naturalists have always tried to classify the abundance of creatures that populate the Earth.

- Aristote: the *scala naturae*;

From Aristotle to Darwin

Since Aristotle, naturalists have always tried to classify the abundance of creatures that populate the Earth.

- Aristote: the *scala naturae*;
- Carl von Linné: classification of living;



From Aristotle to Darwin

Since Aristotle, naturalists have always tried to classify the abundance of creatures that populate the Earth.

- Aristote: the *scala naturae*;
- Carl von Linné: classification of living;
- Antoine Laurent de Jussieu;

From Aristotle to Darwin

Since Aristotle, naturalists have always tried to classify the abundance of creatures that populate the Earth.

- Aristote: the *scala naturae*;
- Carl von Linné: classification of living;
- Antoine Laurent de Jussieu;
- Leclerc de Buffon: the first to evoke the possibility that species can evolve;

From Aristotle to Darwin

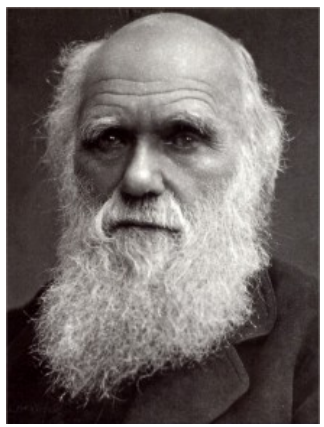
Since Aristotle, naturalists have always tried to classify the abundance of creatures that populate the Earth.

- Aristote: the *scala naturae*;
- Carl von Linné: classification of living;
- Antoine Laurent de Jussieu;
- Leclerc de Buffon: the first to evoke the possibility that species can evolve;
- Jean-Baptiste Lamarck: first theory of evolution;

From Aristotle to Darwin

Since Aristotle, naturalists have always tried to classify the abundance of creatures that populate the Earth.

- Aristote: the *scala naturae*;
- Carl von Linné: classification of living;
- Antoine Laurent de Jussieu;
- Leclerc de Buffon: the first to evoke the possibility that species can evolve;
- Jean-Baptiste Lamarck: first theory of evolution;
- Charles Darwin: *The Origins of Species* (1859).



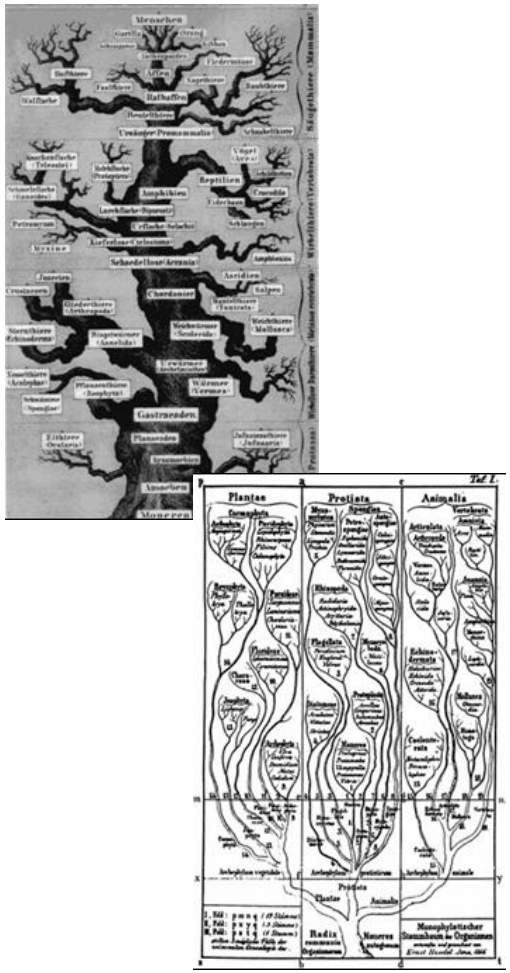
From "The Origin of Species"

- It is a truly wonderful fact . . . that all animals and all plants throughout all time and space should be related to each other in groups, subordinate to groups. [...]
- The affinities of all the beings of the same class have sometimes been represented by a great tree. [...] The green and budding twigs may represent existing species; and those produced during former years may represent the long succession of extinct species.

Charles Darwin, (1872), pp. 170-171. The Origin of Species. Sixth Edition. The Modern Library, New York.

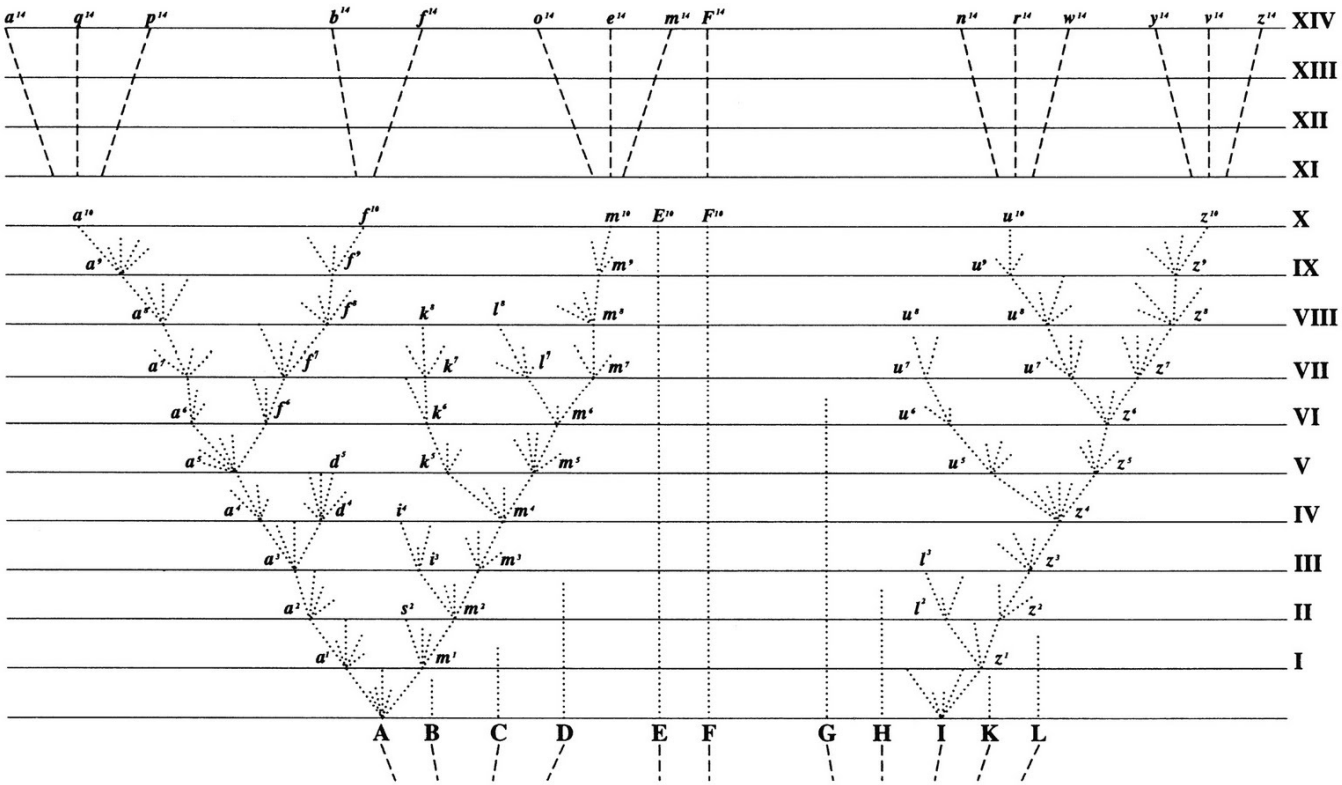
Tree and inter-species relationships

Others before Darwin used trees to illustrate species classifications in light of fixism or descent of some species from others.



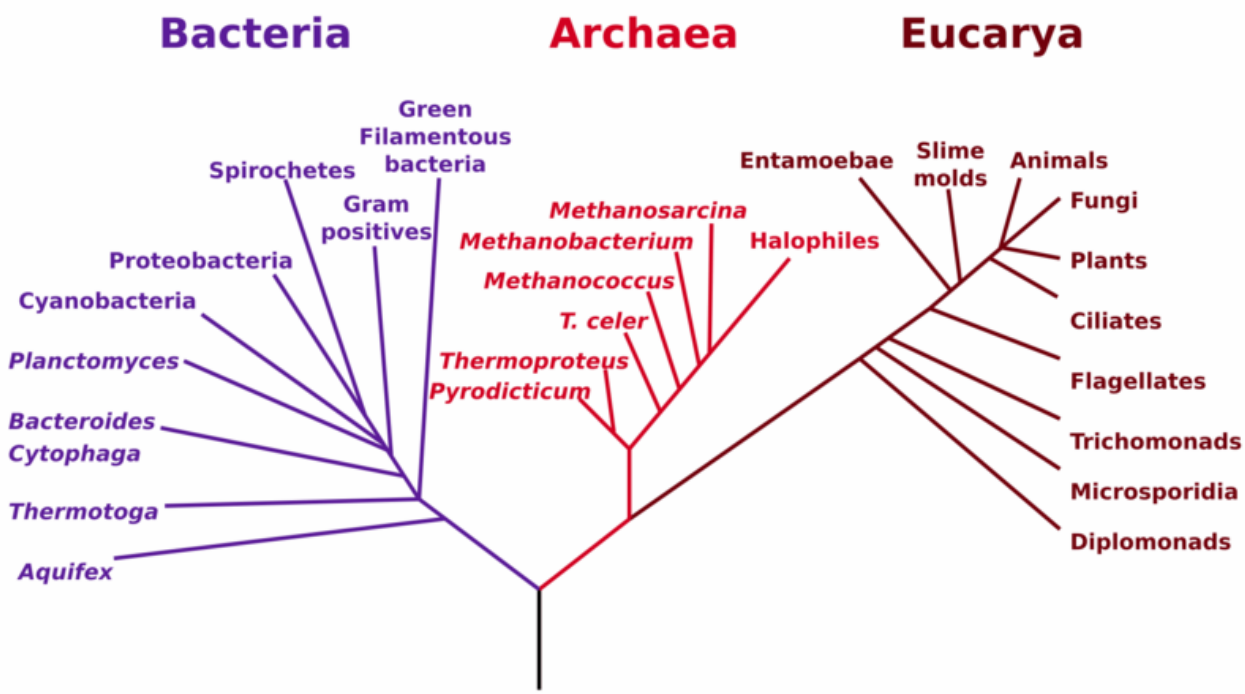
Tree and inter-species relationships

The originality of Darwin's tree is the coexistence, in the same figure, of the concepts of time and descent: the bifurcations in the tree follow one another over time.



Phylogenetics

Phylogenetics aims at clarifying, using molecular and morphological data, the evolutionary relationships that exist among different species. These relationships can be represented through phylogenetic trees or phylogenies.



Woese 1987; Barns et al. 1996; Brown et Doolittle 1997

Applications

TOL – Tree Of Life

Applications

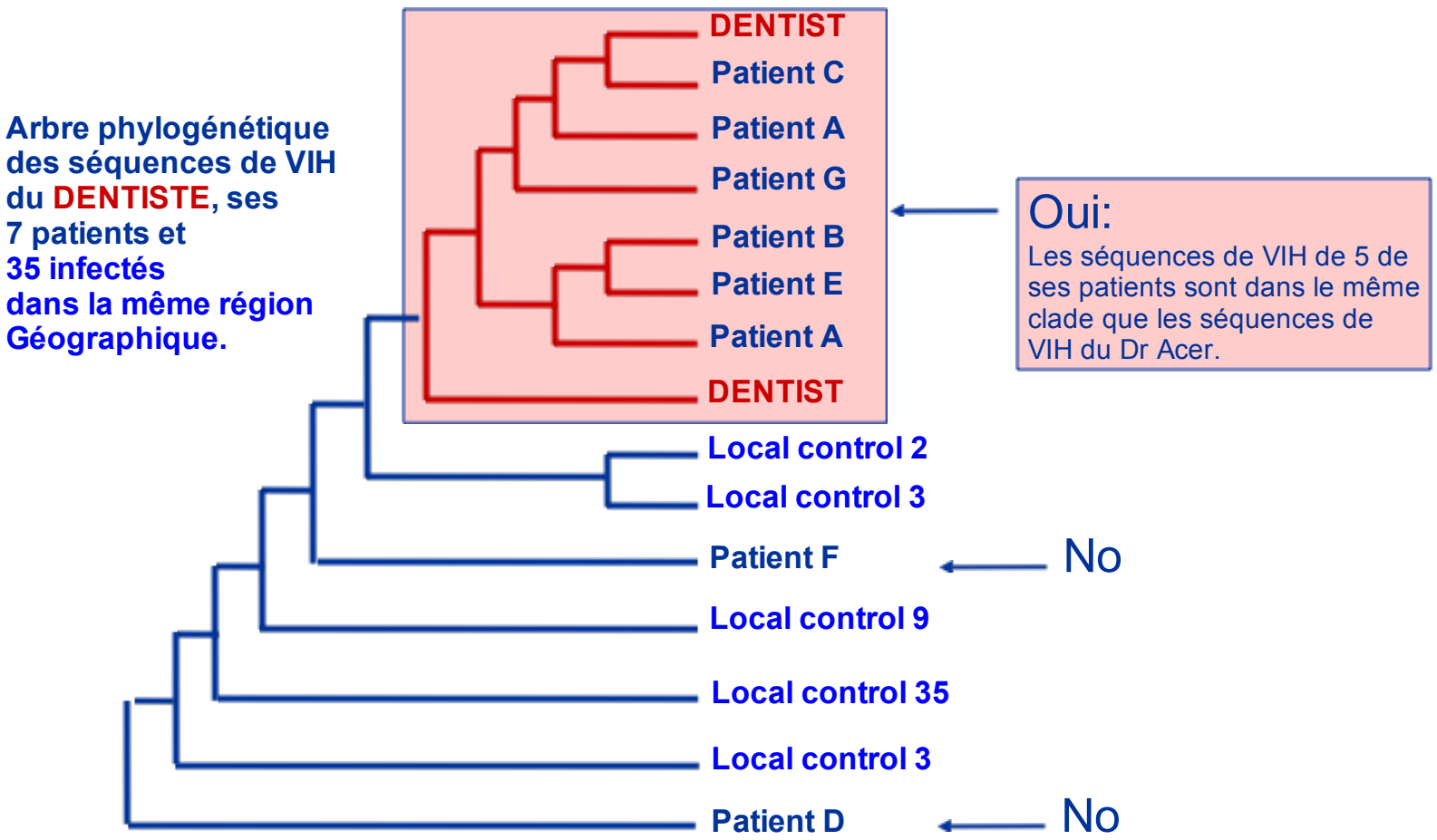
Study character evolution

Applications

Ecology : migrations, host-parasite relationships, ...

Applications

Recurrent example: Did Dr David Acer contaminate his patients?

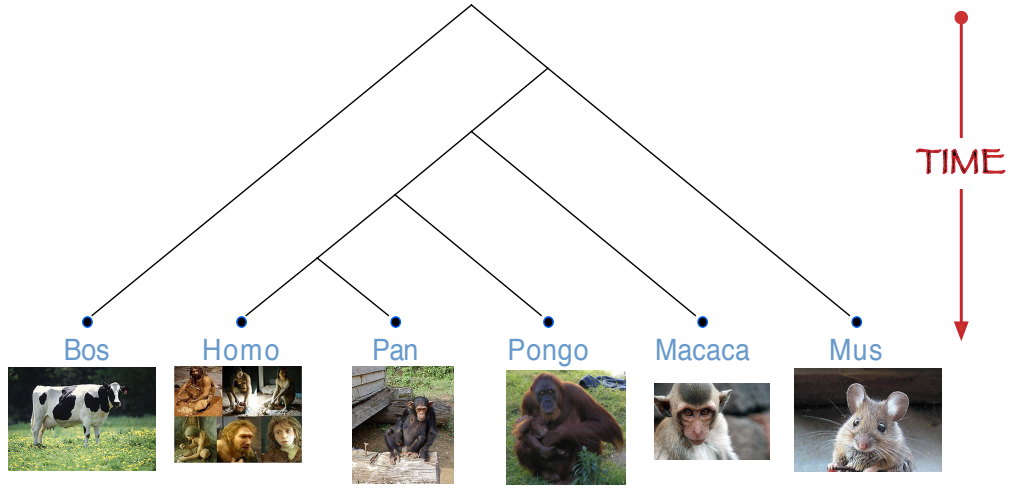


Ou et al. (1992), Page et Holmes (1998)

Rooted phylogenetic trees ...

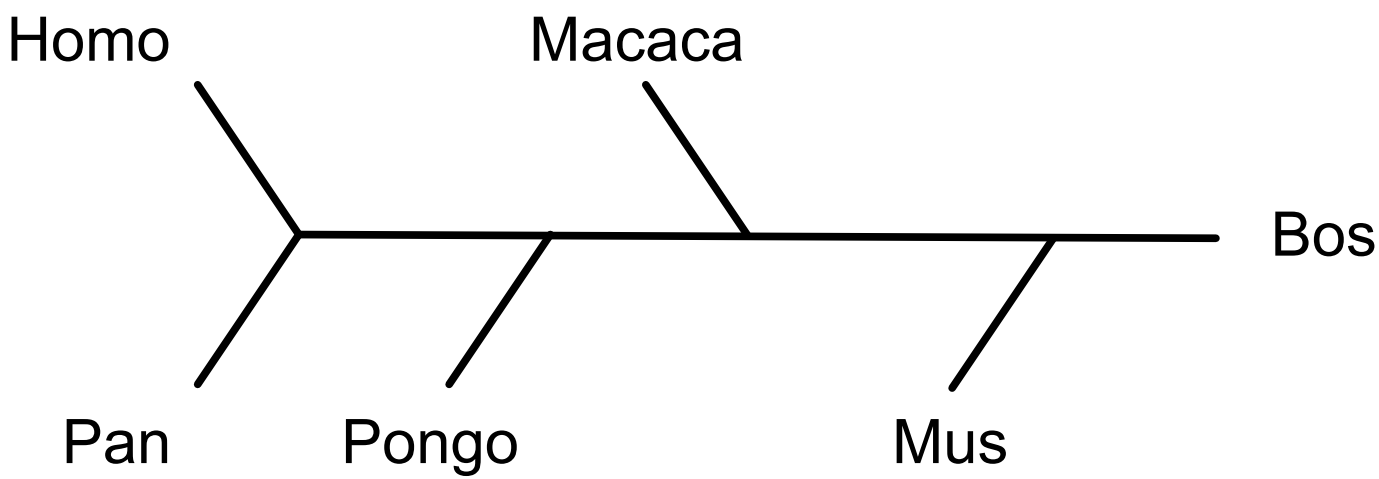
... are oriented connected and acyclic graphs, where terminal nodes are associated to a set of species:

- the leaves or taxa represent extant organisms
- internal nodes represent hypothetical ancestors
- the only node without ancestor is called root
- each internal node represents the lowest common ancestor of all taxa below it (clade)
- nodes and branches can have several kinds of information associated with them, such as time or amount of evolution estimates.



Unrooted phylogenetic trees ...

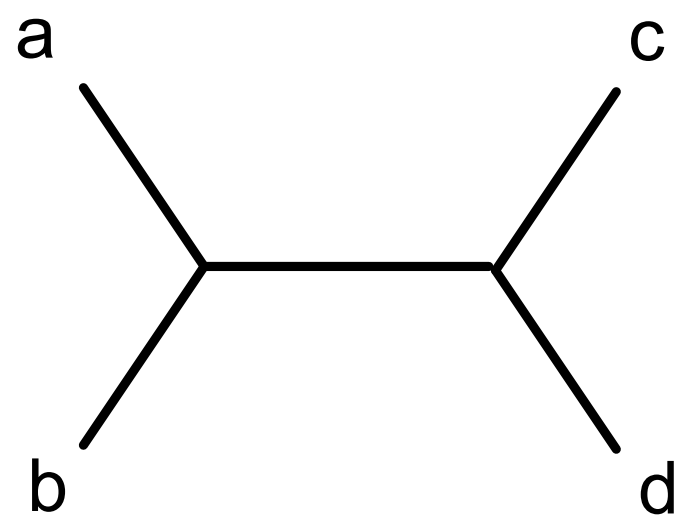
... connected and acyclic graphs, where terminal nodes are associated to a set of species.



Some numbers

An unrooted binary tree with n leaves, with $n \geq 3$, has :

- $2n - 2$ nodes;
- $2n - 3$ branches;
- $n - 3$ internal branches.

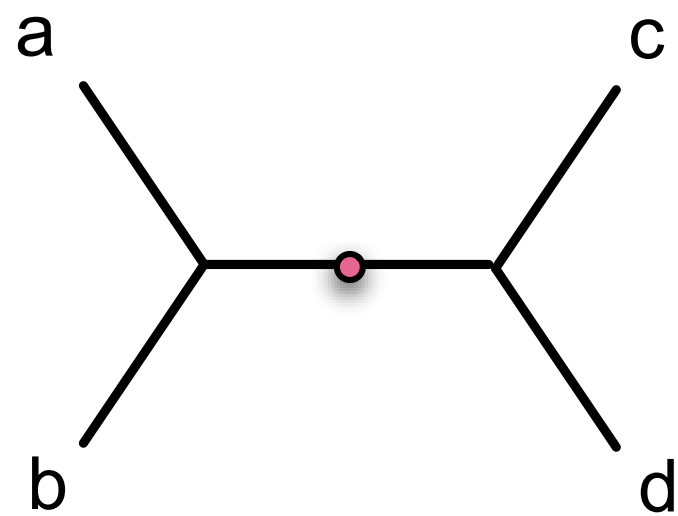


Some numbers

An unrooted binary tree with n leaves, with $n \geq 3$, has :

- $2n - 2$ nodes;
- $2n - 3$ branches;
- $n - 3$ internal branches.

A rooted binary tree with n leaves, with $n \geq 3$, has :



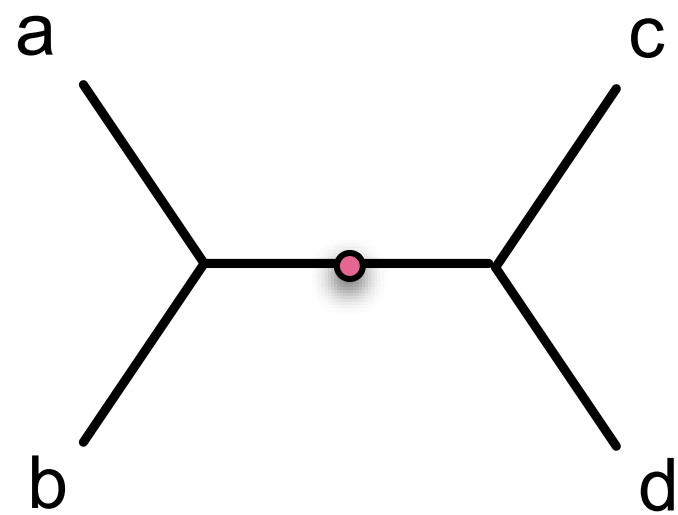
Some numbers

An unrooted binary tree with n leaves, with $n \geq 3$, has :

- $2n - 2$ nodes;
- $2n - 3$ branches;
- $n - 3$ internal branches.

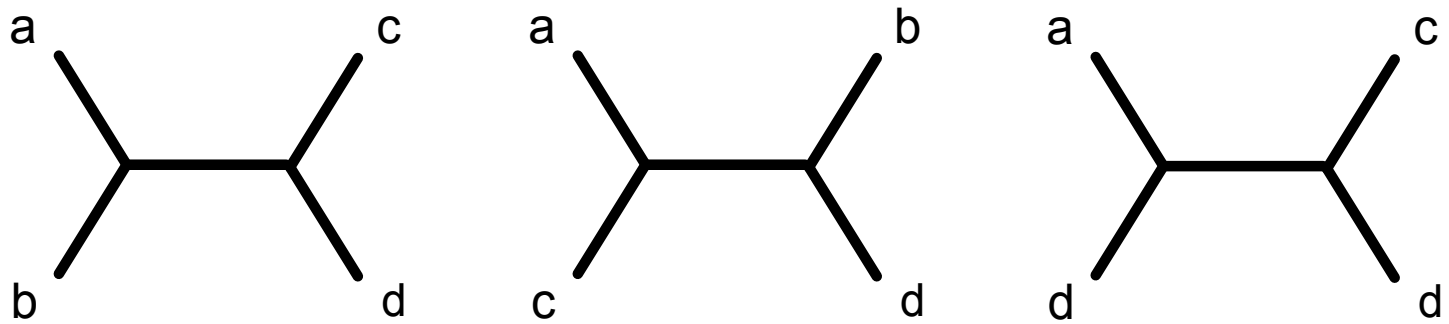
A rooted binary tree with n leaves, with $n \geq 3$, has :

- $2n - 1$ nodes;
- $2n - 2$ branches;
- $n - 2$ internal branches.



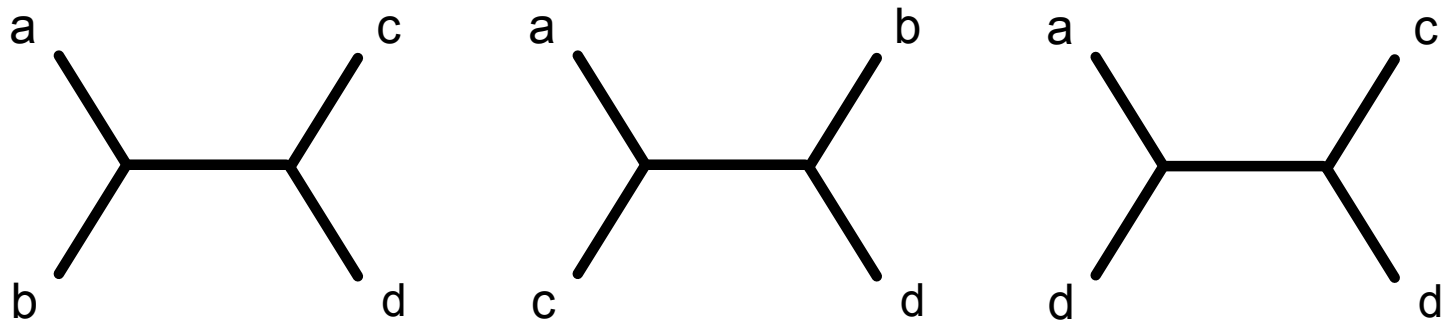
How many trees?

- How many unrooted trees with n leaves, with $n \geq 3$?



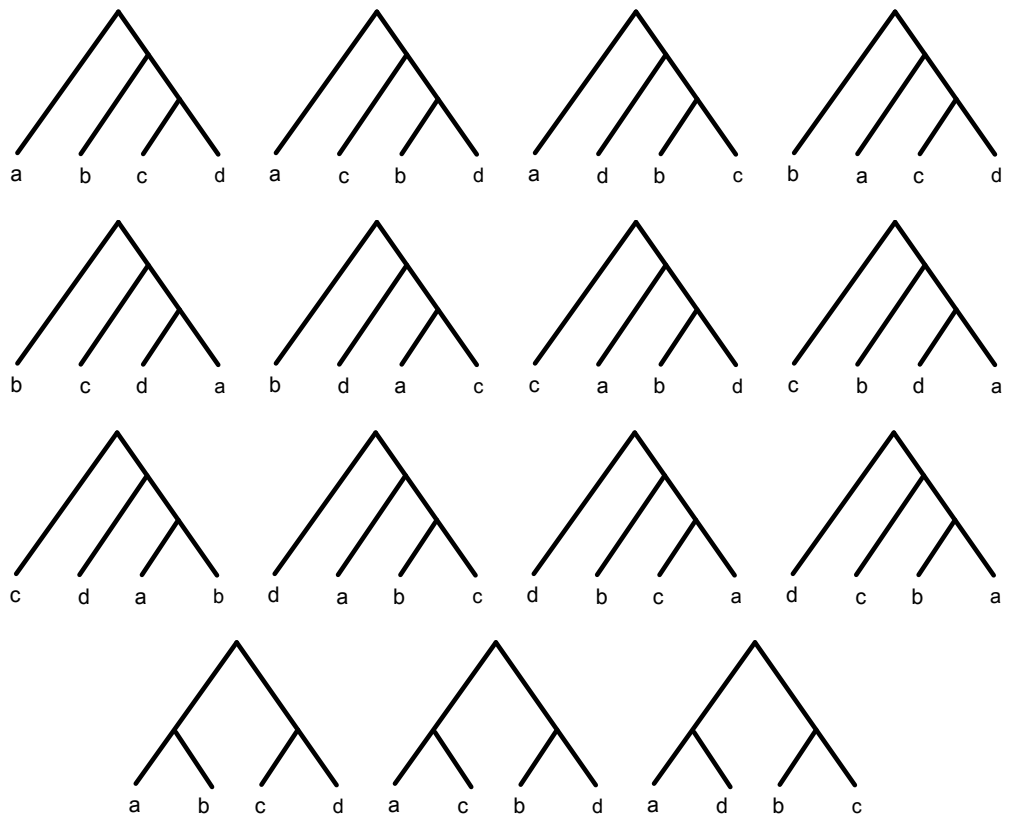
How many trees?

- How many unrooted trees with n leaves, with $n \geq 3$?
  - $U(n) = (2n-5)!! := 3 \cdot 5 \cdot 7 \cdot ... \cdot (2n-5)$.



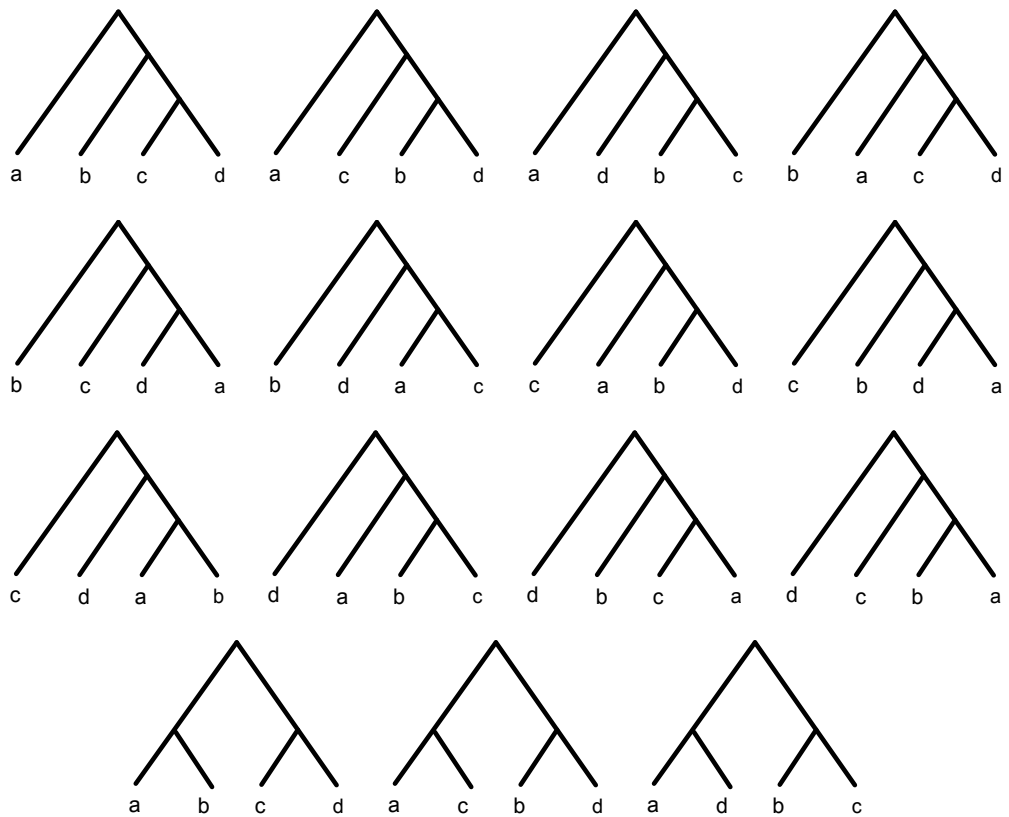
How many trees?

- How many unrooted trees with n leaves, with $n \geq 3$?
  - $U(n) = (2n-5)!! := 3 \cdot 5 \cdot 7 \cdot ... \cdot (2n-5)$.
- How many rooted trees with n leaves, with $n \geq 3$?



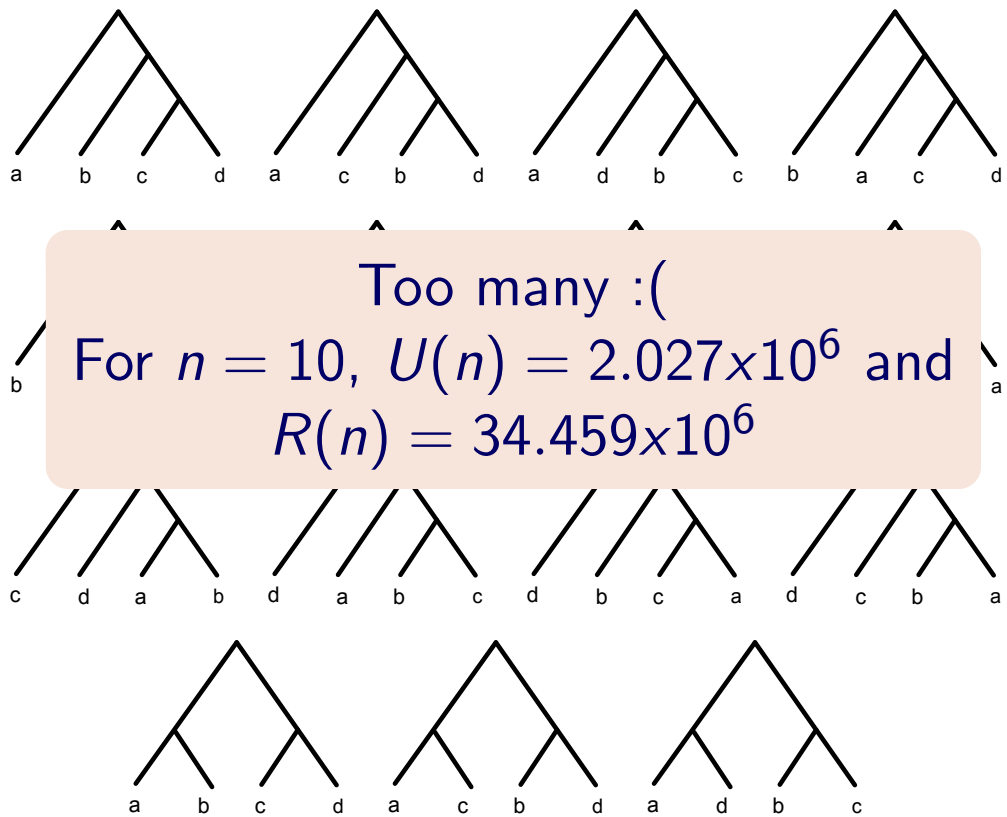
How many trees?

- How many unrooted trees with n leaves, with $n \geq 3$?
  - $U(n) = (2n-5)!! := 3 \cdot 5 \cdot 7 \cdot ... \cdot (2n-5)$.
- How many rooted trees with n leaves, with $n \geq 3$?
  - $R(n) = (2n-3)!! = U(n) \cdot (2n-3) = 3 \cdot 5 \cdot ... \cdot (2n-3)$.



How many trees?

- How many unrooted trees with n leaves, with $n \geq 3$?
  - $U(n) = (2n-5)!! := 3 \cdot 5 \cdot 7 \cdot ... \cdot (2n-5)$.
- How many rooted trees with n leaves, with $n \geq 3$?
  - $R(n) = (2n-3)!! = U(n) \cdot (2n-3) = 3 \cdot 5 \cdot ... \cdot (2n-3)$.

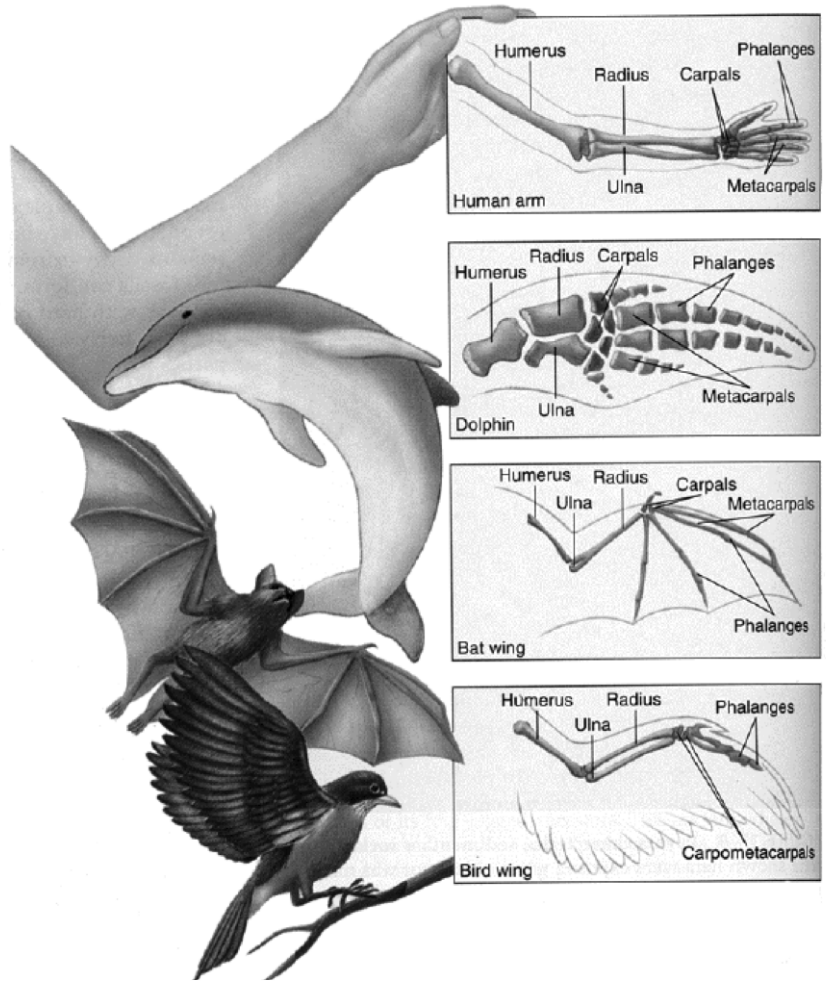


Too many :(
For $n = 10$, $U(n) = 2.027x10^6$ and $R(n) = 34.459x10^6$

Phylogenetics reconstruction

The first phylogenies were based on:

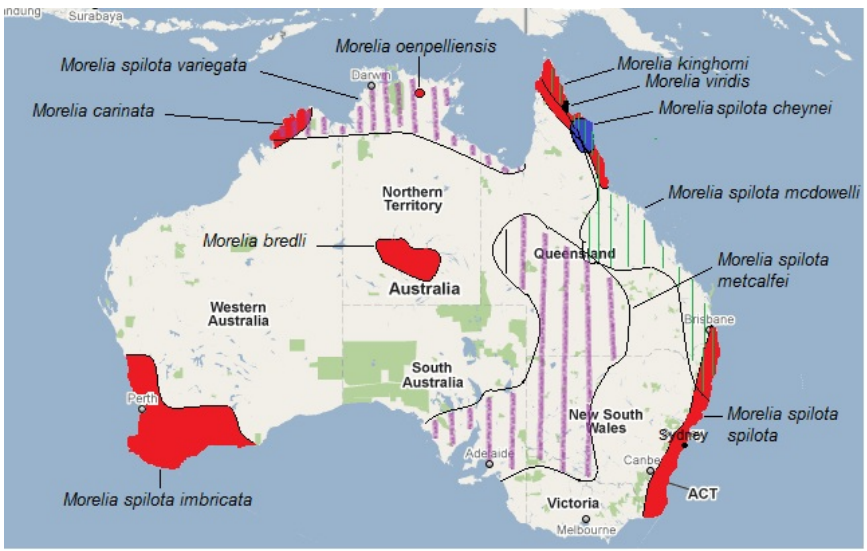
- the comparison of morphological traits (fossils);
- ontogeny;
- the comparison of behaviors;
- geographical repartitions...

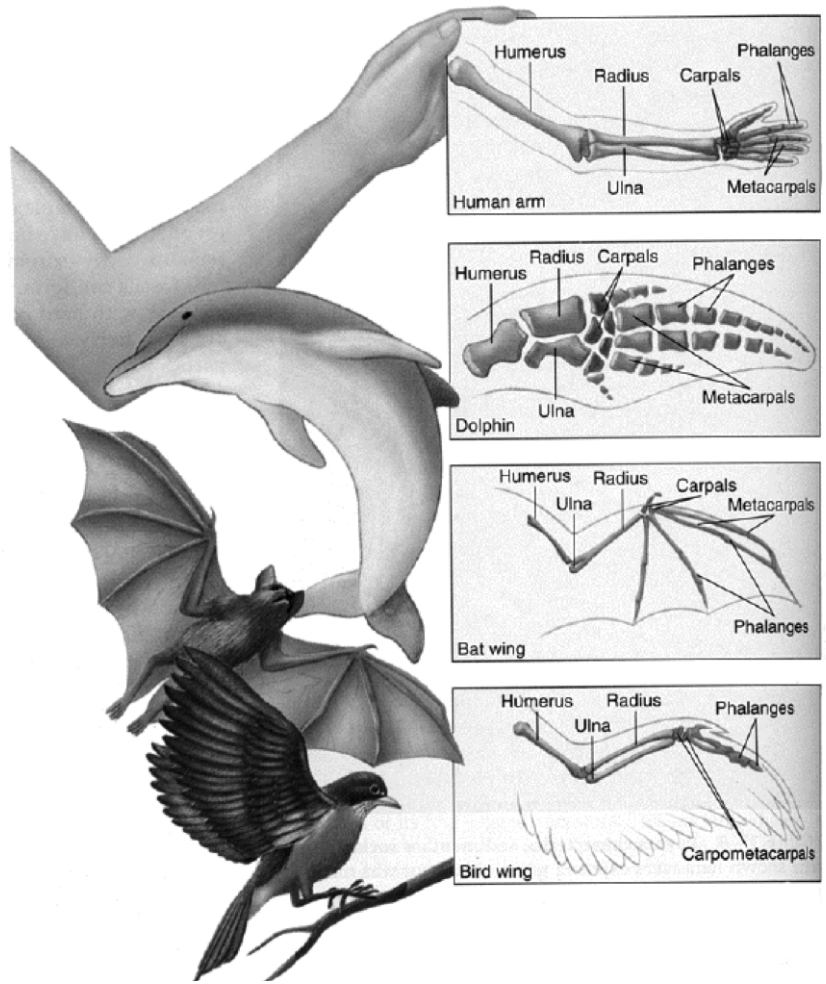


Phylogenetics reconstruction

The first phylogenies were based on:

- the comparison of morphological traits (fossils);
- ontogeny;
- the comparison of behaviors;
- geographical repartitions...





Phylogenetics reconstruction

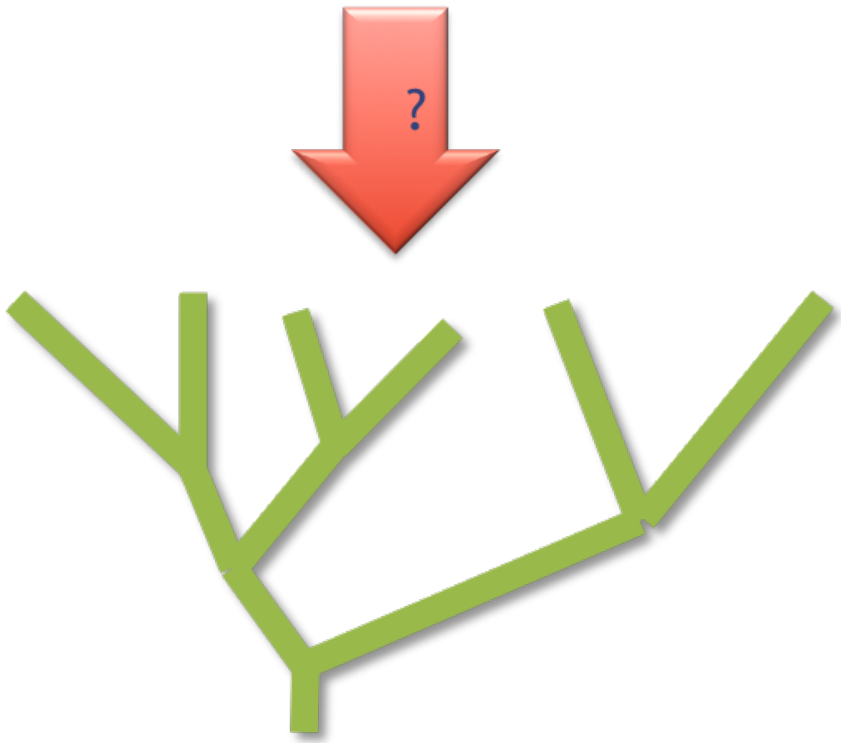
With the discovery of DNA by Watson and Crick in 1953 and the design of sequencing techniques, a new kind of information became available: molecular data.

Today, phylogenies are obtained by studying:

- molecular sequences;
- discrete characters;
- gene frequencies;
- restriction sites;
- microsatellites;
- ...

Molecular phylogenetics

Balene	AAAATCAATTAAACAAAAGCCGCTGCGATATGACAACGAGGAAAAAGAAGCGCGACGGCGGC
Homme	CCAATCAATTAAACAAAAGC----GCGATATGACAACGAGGA---AGAAGCGCGACGGCGGC
Lion	CCAATCAATTAAACAAAAGC----GCGATATG--AACGAGGAAAAAGAAGCGCGACGGTATA
Dauphin	AAAATCAATTAAACAAAAGCCGCTGCGATATGACAACGAGGAAAAAGAAGCTTGACGGCGGC
Vache	CCAATCAATTAAACAAAAGC-----CAAC---ACAACGAGGAAAAAGAAGCTTGACGGTATA



Molecular data – Characters

- Gene or protein alignements:
  - example: 3 taxa on 20 characters and 5 states (A, C, G, T, -)

Species A	ATGGCTATTC-TATAGTACG
Species B	ATCGCT-GTCTTATATTACA
Species C	TTCACT–ACCTGTGGTCCA

  - taxa $\rightarrow$ lines of the matrix, characters $\rightarrow$ columns.
- Morphological characters:

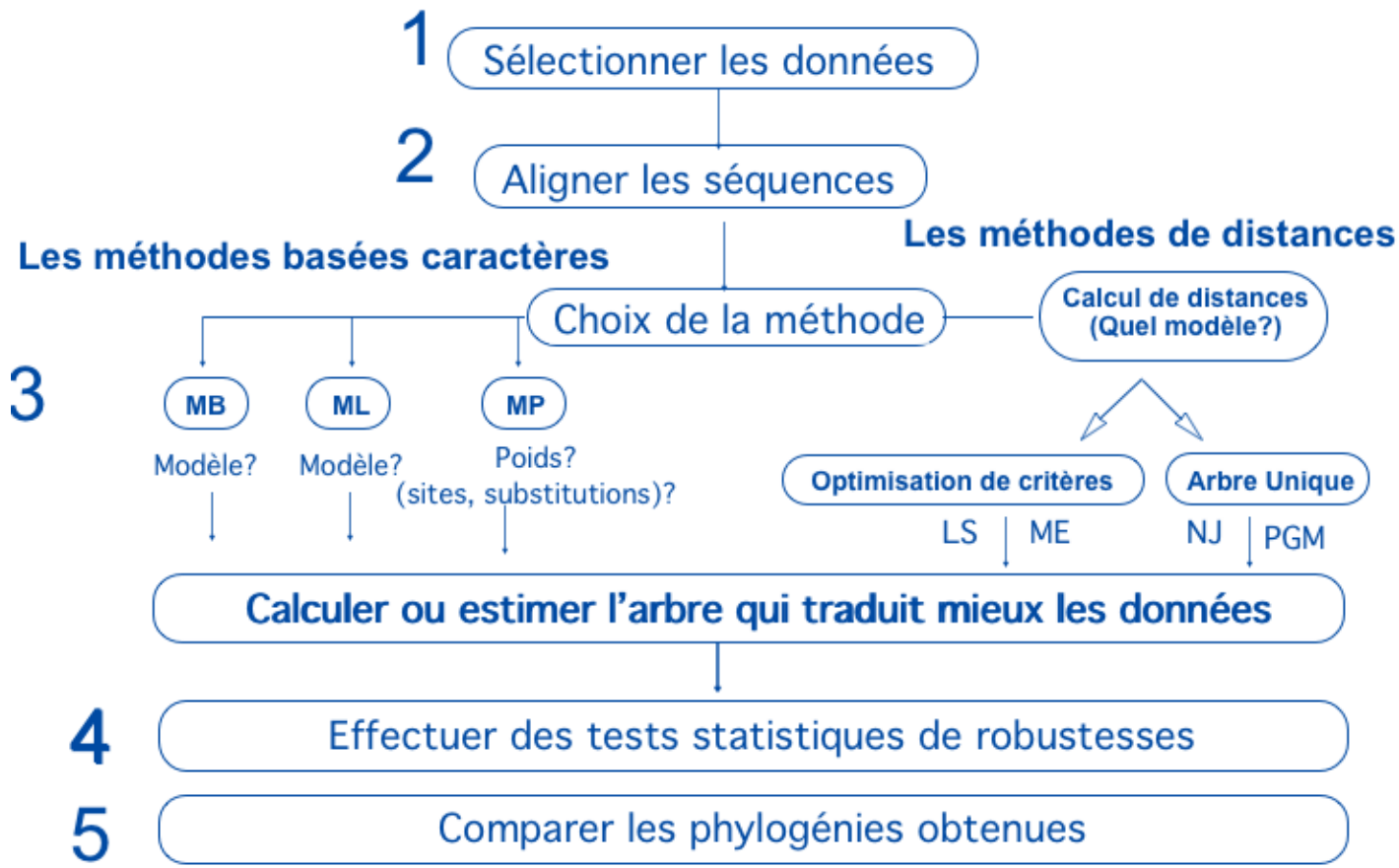
	gills	fins	teeth	...
whale	n	y	n	...
shark	y	y	y	...

Molecular data – Distances

- For n taxa, a matrix D with n lines and n columns;
- case $d_{i,j}$ contains the estimation of the evolutionary distance between taxa i et j;
- examples of distances:
  - Hamming distance;
  - edit (PAM matrices, Dayhoff, Blosum);
  - distances in terms of genome rearrangements (inversions, translocations and substitutions);
- loss of information w.r.t. reconstruction based on an alignment, but permits to use fast algorithms.

	A	B	C	D	E	F
A	0	3	3	5	5	4
B	3	0	3	5	5	4
C	3	3	0	4	4	3
D	5	5	4	0	2	3
E	5	5	4	2	0	3
F	4	4	3	3	3	0

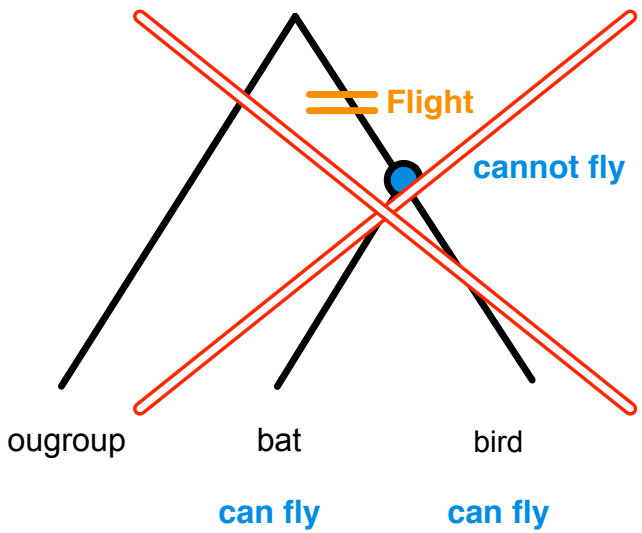
The 4 big steps of phylogenetics reconstruction

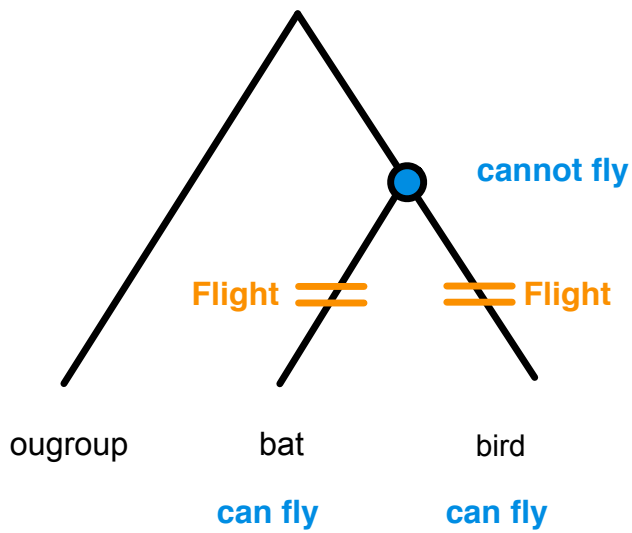


Adapté de Hillis et al., (1993)

STEP 1: Choose the data - homology

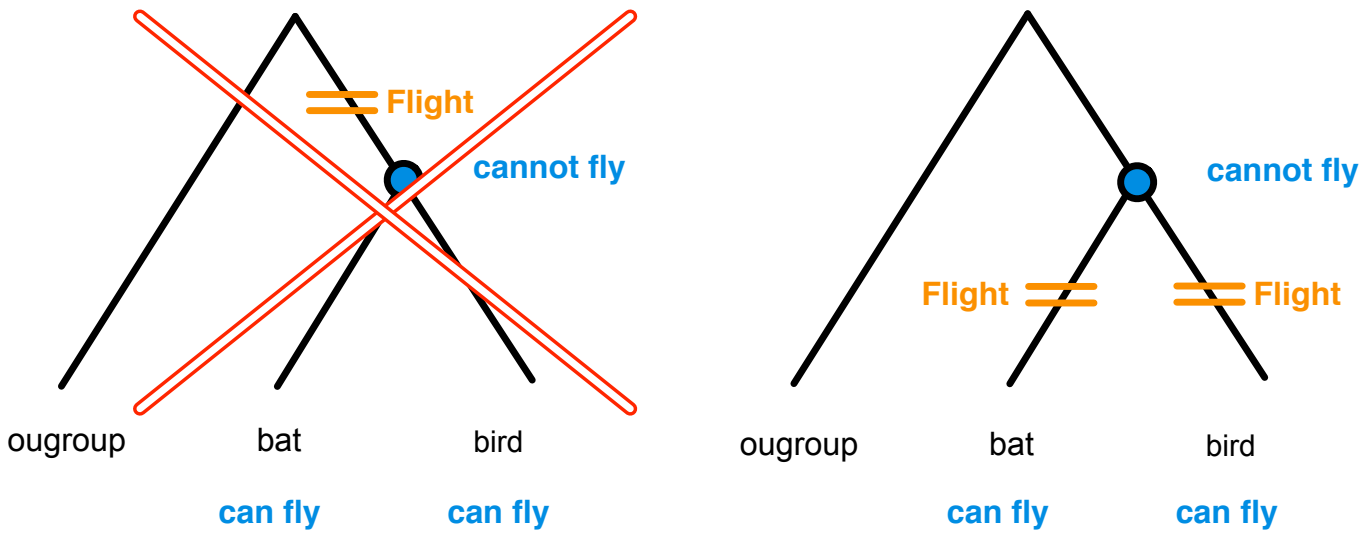
- We should comparer only characters that are similar because they have been inherited from a common ancestor (homology) and not characters that are similar because of convergent evolution (homoplasy) :
  - bat and bird wings (as wings – homoplasy, their commun ancestor cannot fly) ;





STEP 1: Choose the data - homology

- We should comparer only characters that are similar because they have been inherited from a common ancestor (homology) and not characters that are similar because of convergent evolution (homoplasy) :
  - bat and bird wings (as wings – homoplasy, their commun ancestor cannot fly) ;
  - bat and bird wings (as forearms – homology).

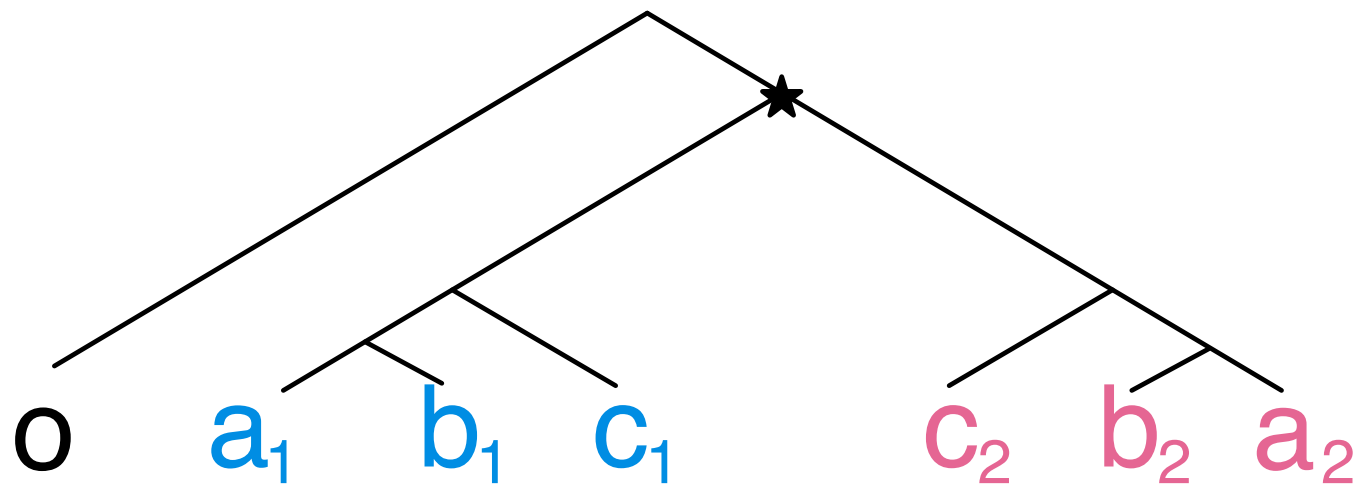


STEP 1: Choose the data - homology

- The same thing apply to molecular characters :
  - 2 proteins in two species can be encoded by the same gene inherited from a common ancestor (homology) ;
  - 2 genes can be similar and have a similar function acquired independly (homoplasy).

STEP 1: Choose the data - homology

- The same thing apply to molecular characters :
  - 2 proteins in two species can be encoded by the same gene inherited from a common ancestor (homology) ;
  - 2 genes can be similar and have a similar function acquired independly (homoplasy).
- Several genes can belong to a gene family (homologues and not orthologues, because of duplications).



STEP 2: Align the sequences

- In molecular sequence evolution, we not only have substitutions but also insertions and deletions of certain sites. This leads to sequences with different lengths.
- To compare the same character in all sequences, we have to align them.
- Several methods have been proposed to do so (dynamic programmation, heuristics, probabilistic methods...).

STEP 2: Align the sequences

- In molecular sequence evolution, we not only have substitutions but also insertions and deletions of certain sites. This leads to sequences with different lengths.
- To compare the same character in all sequences, we have to align them.
- Several methods have been proposed to do so (dynamic programmation, heuristics, probabilistic methods...).

A bad alignment strongly affects the phylogenetics reconstruction

There is a big effort in producing good alignements

The alignment problem (for two sequences)

- An alignment creates a correspondence between two sequences to identify "homologous" characters.

R	D	I	S	L	V	$-$	$-$	$-$	K	N	A	G	I
|		|		|	|				|	|		|	|
R	N	I	$-$	L	V	S	D	A	K	N	V	G	I

The alignment problem (for two sequences)

- An alignment creates a correspondence between two sequences to identify "homologous" characters.

- Three operations are permitted:
  - substitutions;
  - insertions [indels];
  - deletions [indels];

The alignment problem (for two sequences)

- An alignment creates a correspondence between two sequences to identify "homologous" characters.

- Three operations are permitted:
  - substitutions;
  - insertions [indels];
  - deletions [indels];
- More formally:
  - INPUT: Two sequences and a cost scheme.
  - OUTPUT: Quantify and localize the similarity: score + alignment.

The formal definition of an alignment

- For two sequences $X = \{X_1, \cdots, X_n\}$ and $Y = \{Y_1, \cdots, Y_n\}$, a global alignment between X and Y is a matrix with 2 lines and L columns, with $max(m, n) \leq L \leq n + m$, where:
  - each columns is of the type $\binom{X_i}{Y_j}$, $\binom{X_i}{-}$ or $\binom{-}{Y_j}$, with $1 \leq i \leq n$, $1 \leq j \leq m$;
  - for each column with $\binom{X_i}{Y_j}$, the next column has to be $\binom{X_{i+1}}{-}$, $\binom{X_{i+1}}{Y_{j+1}}$ or $\binom{-}{Y_{j+1}}$.

The formal definition of an alignment

- For two sequences $X = \{X_1, \cdots, X_n\}$ and $Y = \{Y_1, \cdots, Y_n\}$, a global alignment between X and Y is a matrix with 2 lines and L columns, with $max(m, n) \leq L \leq n + m$, where:
  - each columns is of the type $\binom{X_i}{Y_j}$, $\binom{X_i}{-}$ or $\binom{-}{Y_j}$, with $1 \leq i \leq n$, $1 \leq j \leq m$;
  - for each column with $\binom{X_i}{Y_j}$, the next column has to be $\binom{X_{i+1}}{-}$, $\binom{X_{i+1}}{Y_{j+1}}$ or $\binom{-}{Y_{j+1}}$.
- Note that, depending on the number of symbols $-$, the number of columns may vary.

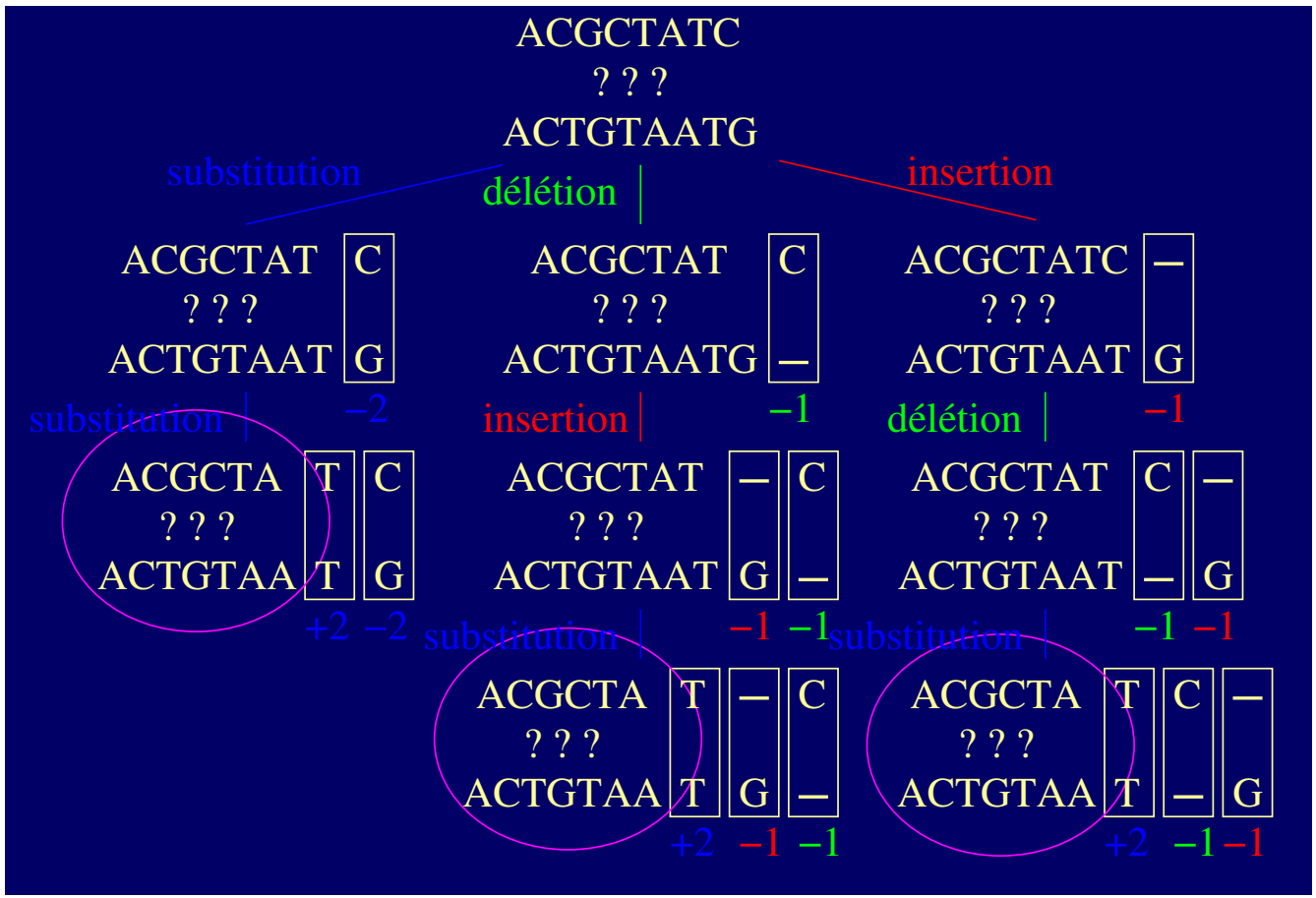
$$\begin{pmatrix} - & I & L \\ L & - & V \end{pmatrix} \quad \begin{pmatrix} I & - & - & L \\ - & L & V & - \end{pmatrix}$$

Scoring scheme

- Identity/substitution: positive /negative scores:
  - a similarity scoring matrix s;
  - $s(a, b)$ is the score of aligning a with b.
- Identity/substitution: positive /negative scores:
  - elementary scoring: one for each indel;
  - complex scoring: affine, logarithmic.
- Score of an alignment: sum of all elementary events composing it.

Alignment problem - a naive solution

- Sequences: $X = ACGCTATC$ and $Y = ACTGTAATG$, with scores -1,-2, 2, respectively for indel, substitution and identity.



Dynamic programing

- stores the sub solutions in a matrix;
- avoids to recompute several times the same sub solutions;
- sub solutions = score of the alignment between prefixes.

		0	1	2	3	4	5	6	7	8	9
			A	C	G	G	C	T	A	T	C
0											
1	A										
2	C										
3	T										
4	G										
5	T										
6	A										
7	A										
8	T										
9	G										

Creation of a table indexed by the two sequences

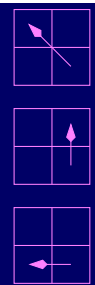
Case (i, j) : score of aligning the first i characters of $ACTGTAATG$ and the first j characters of $ACGGCTATC$

Dynamic programing

If s is the similarity scoring matrix, g is the score of an indel

- Initial conditions : $M(0,0) = 0$, $M(0,j) = g \times j$, $M(i,0) = g \times i$
- Recursion:

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(X_i, Y_j) & \text{identité/substitution} \\ M(i-1,j) + g & \text{délétion} \\ M(i,j-1) + g & \text{insertion} \end{cases}$$



Alignment problem - an example

- Case (9,9) : score of the global alignment between *ACGGCTATC* and *ACTGTAATG*, with scores -1,-2, 2, respectively for indel, substitution and identity

		A	C	G	G	C	T	A	T	C
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
A	-1	2	1	0	-1	-2	-3	-4	-5	-6
C	-2	1	4	3	2	1	0	-1	-2	-3
T	-3	0	3	3	2	1	3	2	1	0
G	-4	-1	2	5	5	4	3	2	1	0
T	-5	-2	1	4	4	4	6	5	4	3
A	-6	-3	0	3	3	3	5	8	7	6
A	-7	-4	-1	2	2	2	4	7	7	6
T	-8	-5	-2	1	1	1	4	6	9	8
G	-9	-6	-3	0	3	2	3	5	8	8

Backtracking

- finds the alignment, given the matrix;
- works by:
  - starting at the lowest cell of the last column;
  - moving to a neighbor cell with maximum score such that the score of the corresponding mutation + the score of the cell we want to move in gives the score of the cell where we are;
  - doing this until we arrive at cell $(0, 0)$.
- If we can backtrack in several ways, there exist several optimal solutions.
- Overall, a $O(n \times m)$ complexity (including matrix computation).

The backtracking - an example

- Sequences: $X = ACGCTATC$ and $Y = ACTGTAATG$, with scores -1,-2, 2, respectively for indel, substitution and identity

		A	C	G	G	C	T	A	T	C
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
A	-1	**2**	1	0	-1	-2	-3	-4	-5	-6
C	-2	1	**4**	3	2	1	0	-1	-2	-3
T	-3	0	3	**3**	2	1	3	2	1	0
G	-4	-1	2	5	**5**	**4**	3	2	1	0
T	-5	-2	1	4	4	4	**6**	5	4	3
A	-6	-3	0	3	3	3	**5**	8	7	6
A	-7	-4	-1	2	2	2	4	**7**	7	6
T	-8	-5	-2	1	1	1	4	6	**9**	8
G	-9	-6	-3	0	3	2	3	5	8	**8**

$$\begin{array}{cccccccccc} A & C & T & G & _ & T & A & A & T & G \\ | & | & & | & & | & & | & | & \\ A & C & G & G & C & T & _ & A & T & C \end{array}$$

More on the alignment problem

- local alignment: alignment of the best subsequences;
- different gap (i.e./ insertions or deletions in a row) scoring schemes
- heuristics, e.g. BLAST et FASTA;
- Multiple Sequence Alignment or MSA (generalization in $O(k^2 2^k n^k)$, NP-complet)

Reconstruction of phylogenies for one dataset

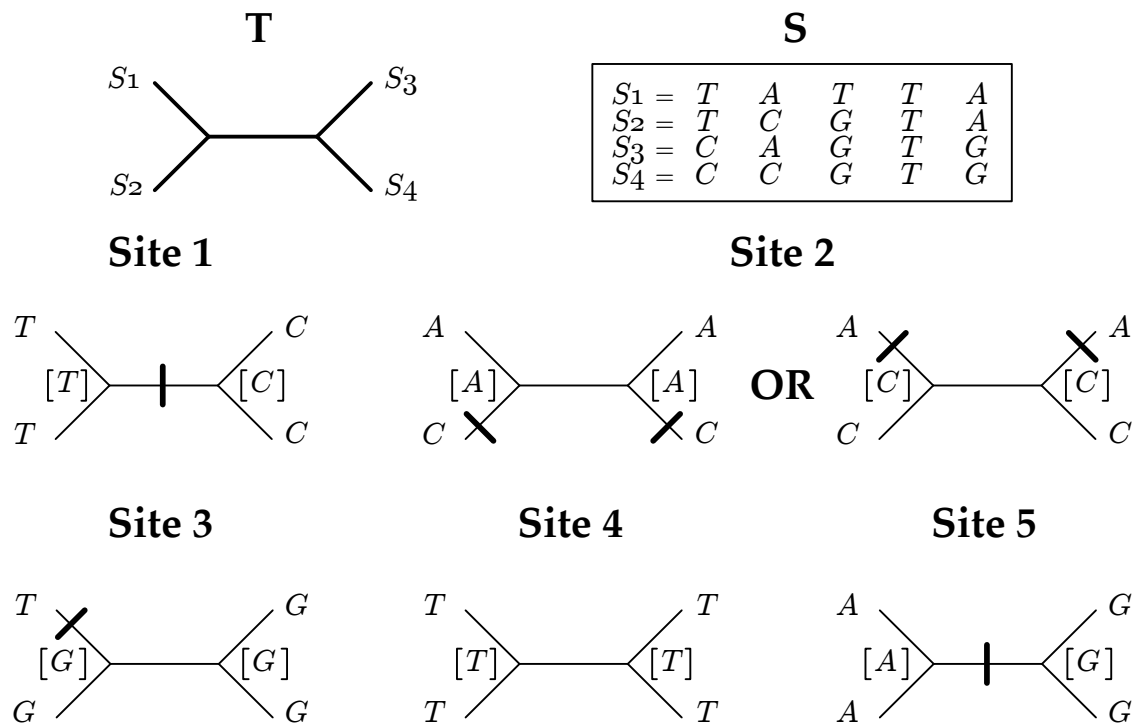
- character-based methods, which retrieve similarities comparing the states taken by species at different characters:
  - without modeling sequence evolution:
    - parsimony methods (e.g. Fitch parsimony)
  - using models of sequence evolution:
    - likelihood methods (e.g. methods implemented in PhyML or RaxML)
    - bayesian methods (e.g. methods implemented in MrBayes or BEAST)
- distance-based methods, which use pairwise distances to quantify the amount of evolution separating species (e.g. BIONJ, FASTME)

Parsimony methods

- General principle: Given a set S of n character sequences (one per studied species) $S_1, \cdots, S_n$ of length m and a cost for each transformation $C_{x \to y}$, find the most parsimonious phylogeny(ies).
  - Fitch parsimony ($C_{x \to y} = 1$), Sankoff parsimony ($C_{x \to y} = C_{y \to x}$), Dollo's, Camin-Sokal's ...
- If the tree T is given, the problem of computing $P(S|T)$ is easy:
  - each character can be analyze independently, so $P(S|T) = \sum_{j=1}^{m} w_j P(c_j|T)$, where $P(c_j|T)$ is the sum of $P(c_j|e)$ over all branches T, weighted by the corresponding substitution costs $C_{x \to y}$;
  - the choice of the root does not change the parsimony value $P(S|T)$ if $C_{x \to y} = C_{y \to x}$ (usually true);
  - the number of possible states for a character is limited.

Parsimony methods - an example

(1) compute $P(S|T)$: an $O(nm)$ algorithm to compute $P(S|T)$ has been proposed by Fitch in 1971.

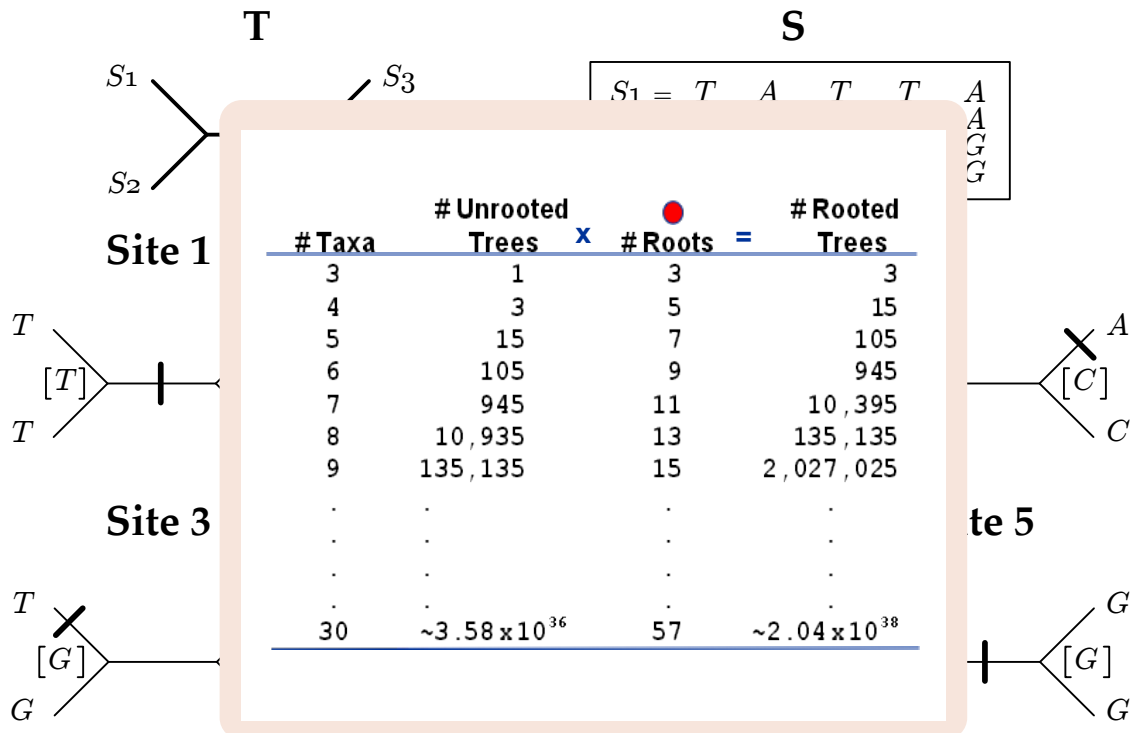


Parsimony methods - an example

(1) compute $P(S|T)$: an $O(nm)$ algorithm to compute $P(S|T)$ has been proposed by Fitch in 1971.

(2) compute the best tree:

- find the tree minimizing $P(S|T)$ is an NP-hard problem;
- parsimony methods are not consistent.



# Taxa	# Unrooted Trees	x	# Roots	=	# Rooted Trees
3	1		3		3
4	3		5		15
5	15		7		105
6	105		9		945
7	945		11		10,395
8	10,935		13		135,135
9	135,135		15		2,027,025
.	.		.		.
.	.		.		.
.	.		.		.
.	.		.		.
30	~3.58 x 10^{36}		57		~2.04 x 10^{38}

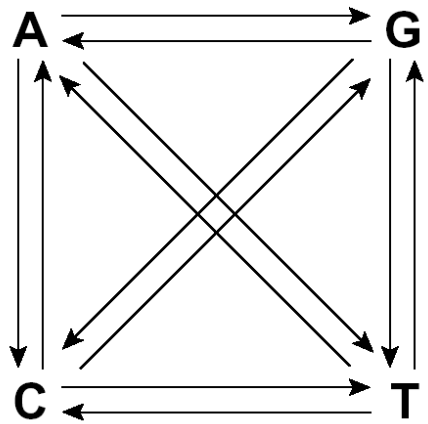
Nucleotide models

Common hypothesis :

- sequences evolve exclusively through nucleotide substitutions, no INDELs;
- each site is a i.i.d. variable;
- substitution process is a first-order Markov model;
- substitution process is homogeneous;
- substitution process is stationary;
- the substitution probability during an infinitesimal time interval dt is proportional to dt;
- there is at most one substitution per infinitesimal time interval dt.

These models are characterized by a 4x4 Q matrix, where Q_{xy} is the frequency to which base x substitute to base y :

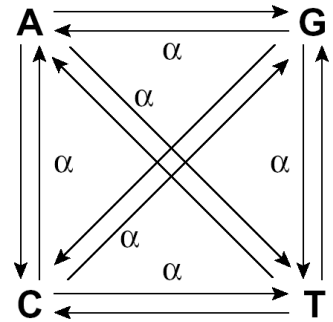
$$Q = \begin{pmatrix} \lambda_A & Q_{AC} & Q_{AG} & Q_{AT} \\ Q_{CA} & \lambda_C & Q_{CG} & Q_{CT} \\ Q_{GA} & Q_{GC} & \lambda_G & Q_{GT} \\ Q_{TA} & Q_{TC} & Q_{TG} & \lambda_T \end{pmatrix}$$



Nucleotide models

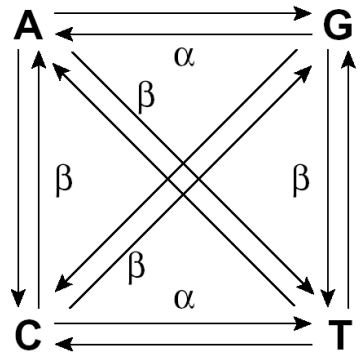
JC

$$Q = \begin{pmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{pmatrix}$$



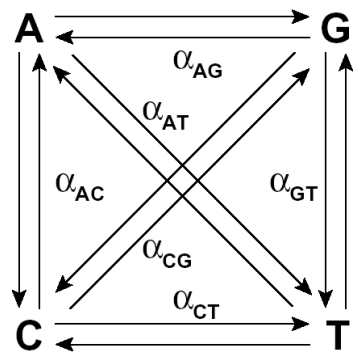
K2P

$$Q = \begin{pmatrix} -\alpha - 2\beta & \beta & \alpha & \beta \\ \beta & -\alpha - 2\beta & \beta & \alpha \\ \alpha & \beta & -\alpha - 2\beta & \beta \\ \beta & \alpha & \beta & -\alpha - 2\beta \end{pmatrix}$$

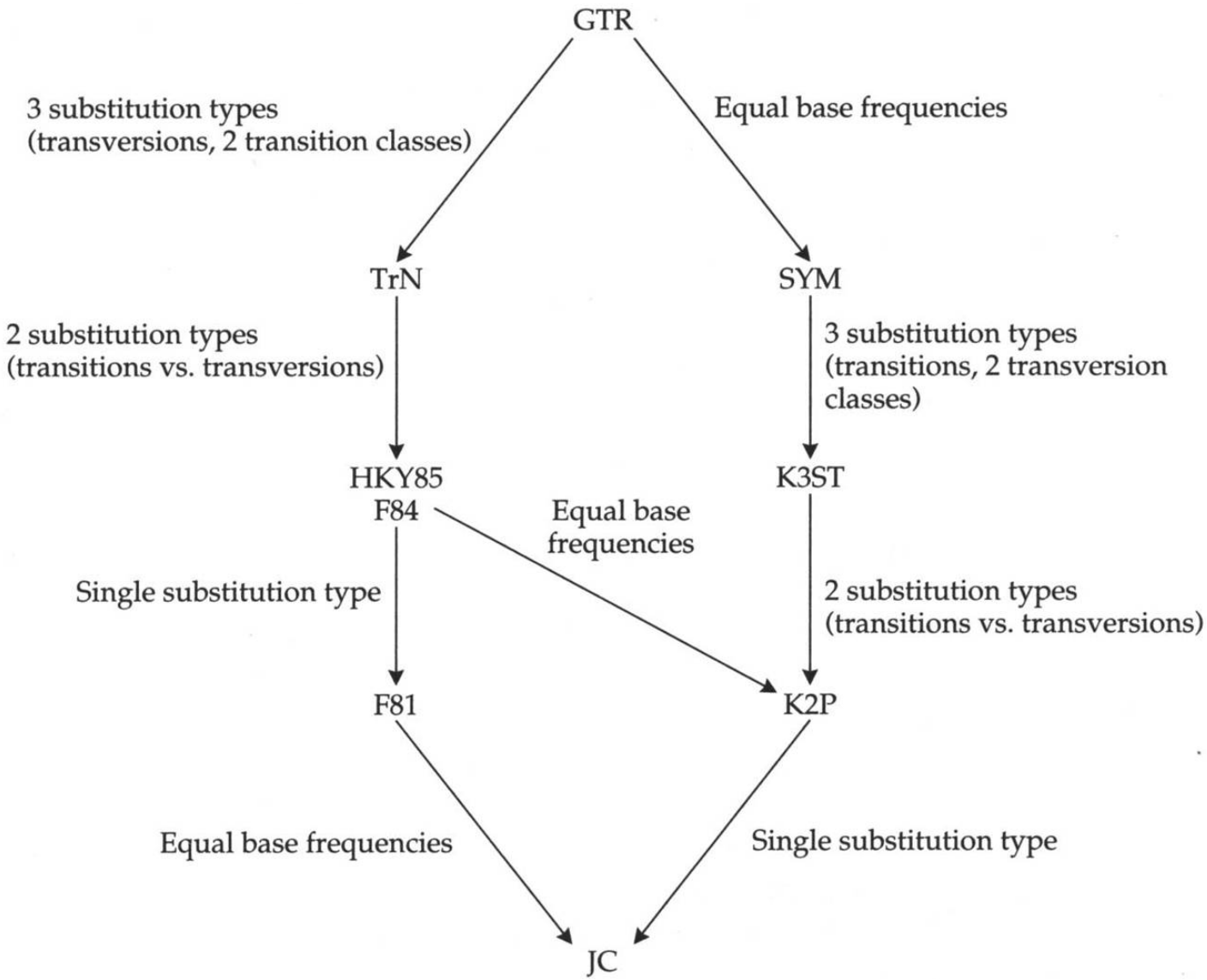


GTR

$$Q = \begin{pmatrix} \lambda_A & \pi_C R_{AC} & \pi_G R_{AG} & \pi_T R_{AT} \\ \pi_A R_{AC} & \lambda_C & \pi_G R_{CG} & \pi_T R_{CT} \\ \pi_A R_{AG} & \pi_C R_{CG} & \lambda_G & \pi_T R_{GT} \\ \pi_A R_{AT} & \pi_C R_{CT} & \pi_G R_{GT} & \lambda_T \end{pmatrix}$$



Nucleotide models



Nucleotide models

More complex models

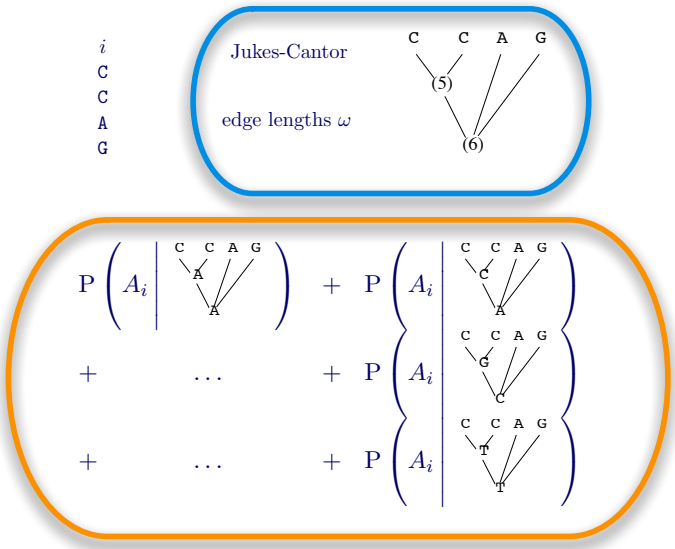
- protein models;
- codon models;
- ...

Likelihood methods

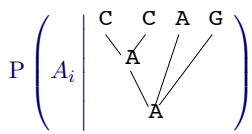
- General principle: Given a set S of n character sequences (one per studied species) $S_1, \cdots, S_n$ of length m, find the most lihehood phylogeny(ies).
- If the parameter vector θ (evolutionary model, topology T + branch lengths ω) is given, the problem of computing $\mathbb{P}(S|\theta)$ is easy:
  - each character can be analyze independently, so $\mathbb{P}(S|\theta) = \prod_{j=1}^{m} \mathbb{P}(c_j|\theta)$;
  - the choice of the root does not change the parsimony value $\mathbb{P}(S|\theta)$ if the model is reversible (usually true);
  - the number of possible states for a character is limited.

Likehood methods - an exemple

(1) compute $\mathbb{P}(S|\theta)$: an $O(nm)$ algorithm to compute $\mathbb{P}(S|\theta)$ has been proposed by Felsenstein in 1981.



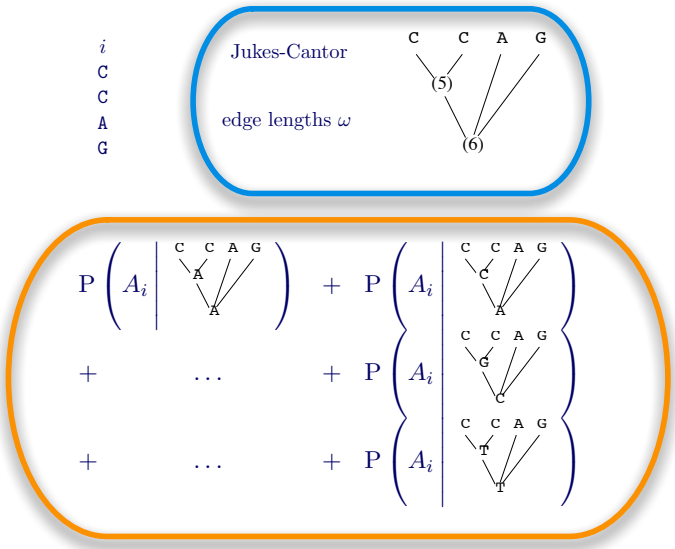
where



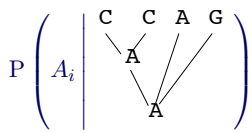
since $P(x, y)(t)$ is equal to $\frac{1}{4}(1 - e^{-4\alpha t})$ if $x \neq y$ and $\frac{1}{4}(1 + 3e^{-4\alpha t})$ otherwise.

Likehood methods - an exemple

(1) compute $\mathbb{P}(S|\theta)$: an $O(nm)$ algorithm to compute $\mathbb{P}(S|\theta)$ has been proposed by Felsenstein in 1981.



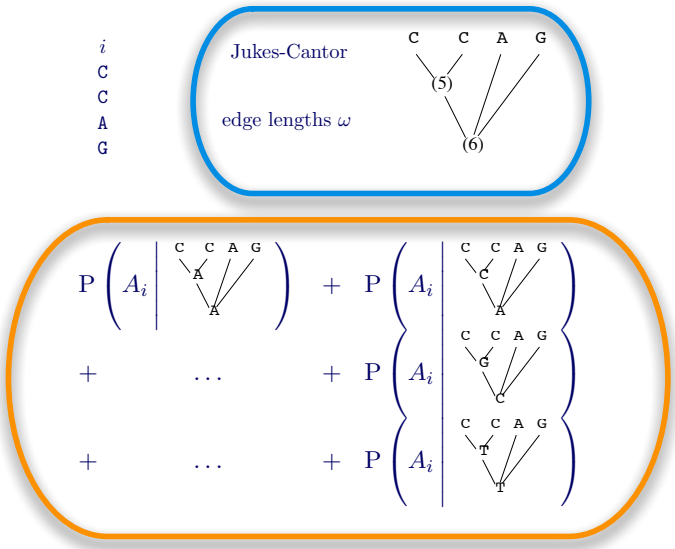
where



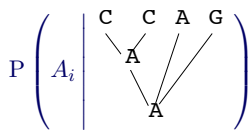
since $P(x, y)(t)$ is equal to $\frac{1}{4}(1 - e^{-4\alpha t})$ if $x \neq y$ and $\frac{1}{4}(1 + 3e^{-4\alpha t})$ otherwise.
$= P(A)P_{e_1}(A, C)P_{e_2}(A, C)P_{e_3}(A, A)P_{e_4}(A, A)P_{e_5}(A, G)$

Likehood methods - an exemple

(1) compute $\mathbb{P}(S|\theta)$: an $O(nm)$ algorithm to compute $\mathbb{P}(S|\theta)$ has been proposed by Felsenstein in 1981.



where



since $P(x, y)(t)$ is equal to $\frac{1}{4}(1 - e^{-4\alpha t})$ if $x \neq y$ and $\frac{1}{4}(1 + 3e^{-4\alpha t})$ otherwise.

$= P(A)P_{e_1}(A, C)P_{e_2}(A, C)P_{e_3}(A, A)P_{e_4}(A, A)P_{e_5}(A, G)$

$= \frac{1}{4^6}(1 - e^{-4\alpha\omega_1})(1 - e^{-4\alpha\omega_2})(1 + 3e^{-4\alpha\omega_3})(1 + 3e^{-4\alpha\omega_4})(1 - e^{-4\alpha\omega_5})$

Likelihood methods - limits

- Limits: to find the most likelihood phylogeny(ies), we need to consider ALL possible parameter vectors θ (evolutionary model, topology T + branch lengths ω);
- A solution: Heuristics, such as PAUP$*$, PHYML, IQPNNI, RAxML, GARLI ...
- Limits: for some evolutionary models, we cannot find analytical solutions (pour $P_{xy}(\cdot)$, π_x ...);
- A solution: bayesian methods.

Bayesian methods

Based on the MCMC algorithm [Metropolis et al., 1953]:

1. start with a random vector of parameters θ_i;
2. select a new vector θ_j by modifying θ_i in some way;
3. compute the *acceptance ratio*

$$R = \frac{\mathbb{P}(\theta_j|S)}{\mathbb{P}(\theta_i|S)} = \frac{\mathbb{P}(\theta_j)\mathbb{P}(S|\theta_j)}{\mathbb{P}(\theta_i)\mathbb{P}(S|\theta_i)};$$

4. accept θ_j with a probability $\rho{=}\max(R, 1)$;
5. every k generations, save the current tree and all parameters;
6. return to step 2.

Variants: Metropolis-Hasting, Metropolis Coupled MCMC (called also MC^3).

Distance methods

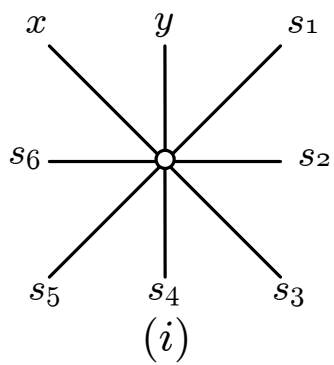
Given a matrix S of m characters for n sequences :

- we compute a distance matrix D nxn where the distance $D_{s,z}$ between two sequences s and z is defined as the number of average mutations for site that occurred since the divergence of s and z (we use the information contained in S and a model of evolution M to correct the observed distances);
- we want to reconstruct a phylogeny T such that the distances in D are identical to that in T;
- not always possible (if the matrix is not additive);
- in this case, we want to get the tree that is *closer* to D (i.e. a tree optimizing a given criterion);
- algorithms based on clustering (e.g. UPGMA, WPGMA, NJ, BIONJ) or not (e.g. FASTME).

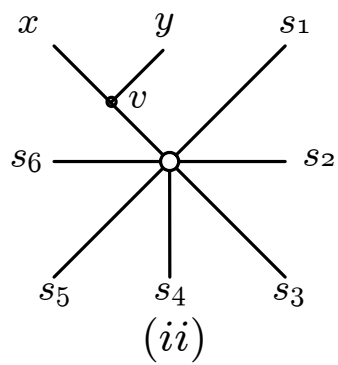
Distance methods based on clustering

Different methods, based on different answers to the following questions:

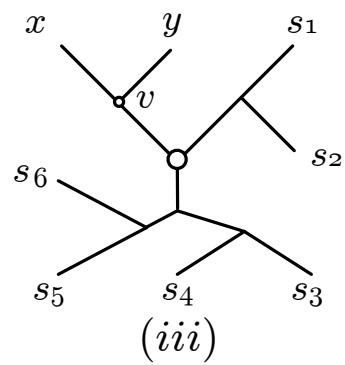
- How to choose the 2 groups to fusion?
- How to replace 2 groups by one in the distance matrix?
- How to evaluate the branch lengths?
  - Least squared methods.



(i)



(ii)

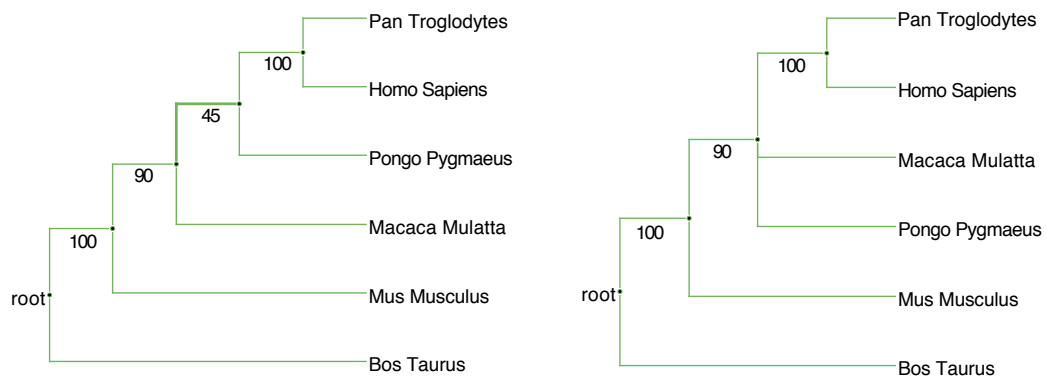


(iii)

Confidence values – bootstrap

Given a tree T obtained with an inference method I from a sequence matrix M with n rows (one per species) and m columns (one per site):

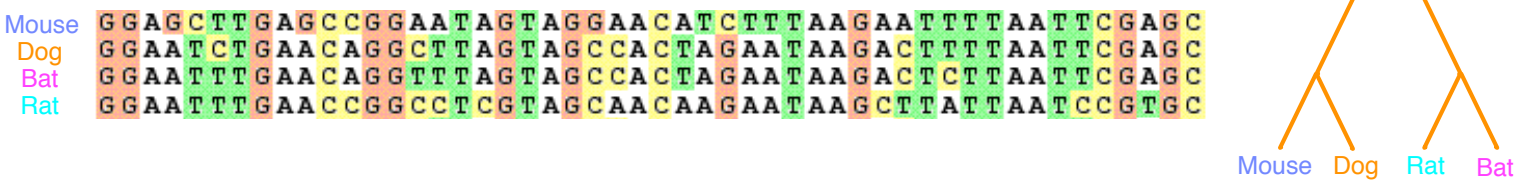
- a set of bootstrap replicates $M = \{M_1, \cdots, M_k\}$ is derived from M by sampling, with replacement, columns of M until obtaining a matrix with m columns;
- from each bootstrap replicate M_i a tree T_i is inferred, employing the inference method I;
- the so-obtained forest $\mathcal{T} = \{T_1, \cdots T_k\}$ is used to estimate the reliability of each branch e of T, with the percentage of trees in $\mathcal{T}$ containing e. This value is called the bootstrap value of e.



Variant : (delete-half) jackknife : it consists in obtaining a set of pseudo matrices randomly by sampling without replacement half of the columns of M.

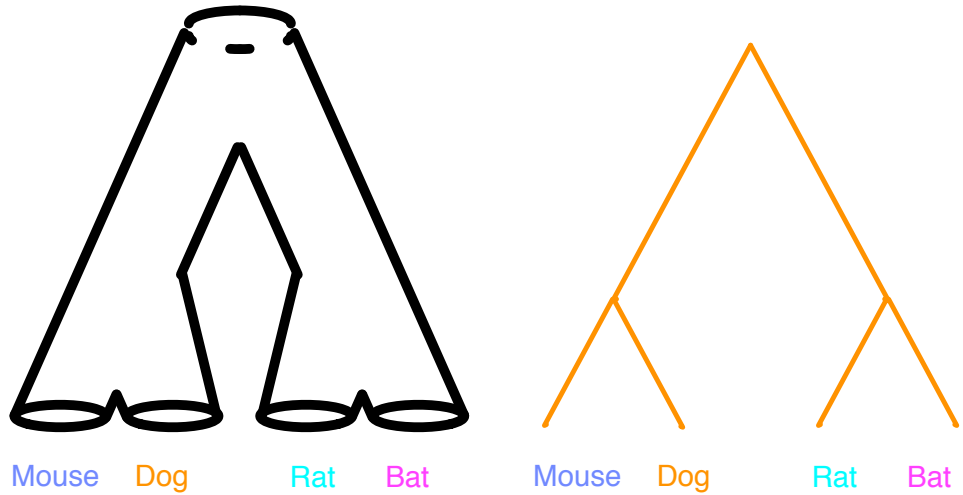
Gene trees

- Gene trees are built by analyzing a gene family, i.e., homologous molecular sequences appearing in the genome of different organisms.



Gene trees

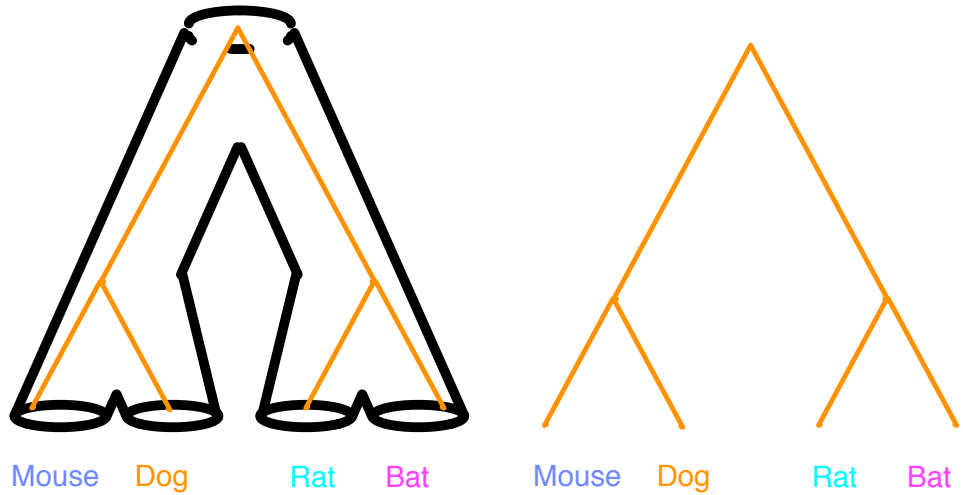
- Gene trees are built by analyzing a gene family.



- Often used to estimate species trees.

Gene trees

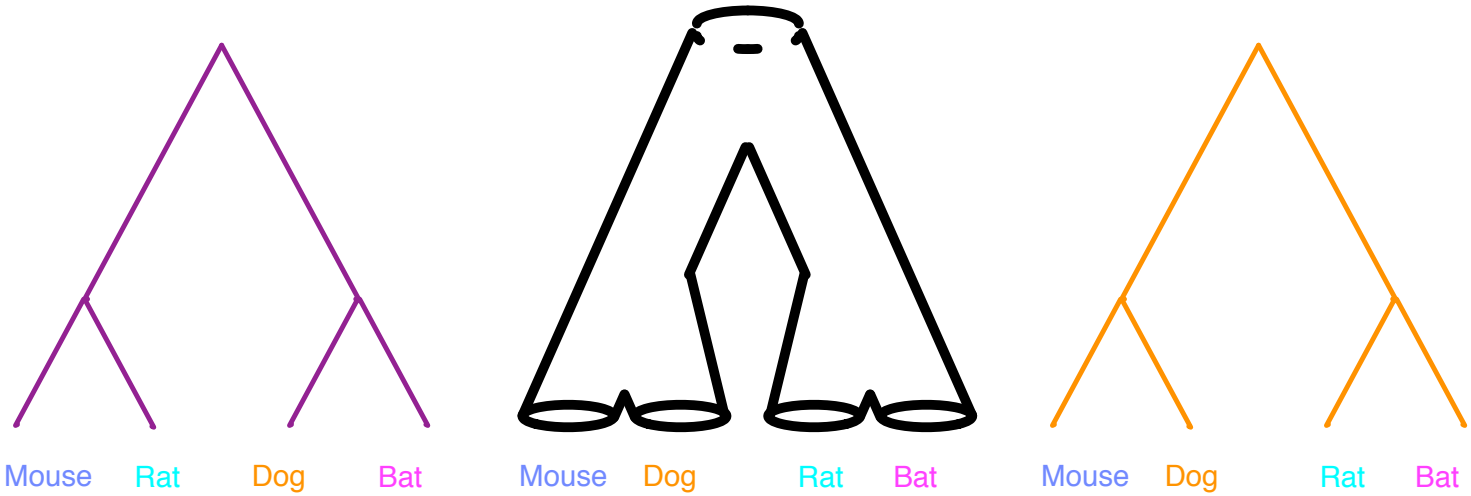
- Gene trees are built by analyzing a gene family.



- Often used to estimate species trees.

Gene trees

- Gene trees are built by analyzing a gene family.



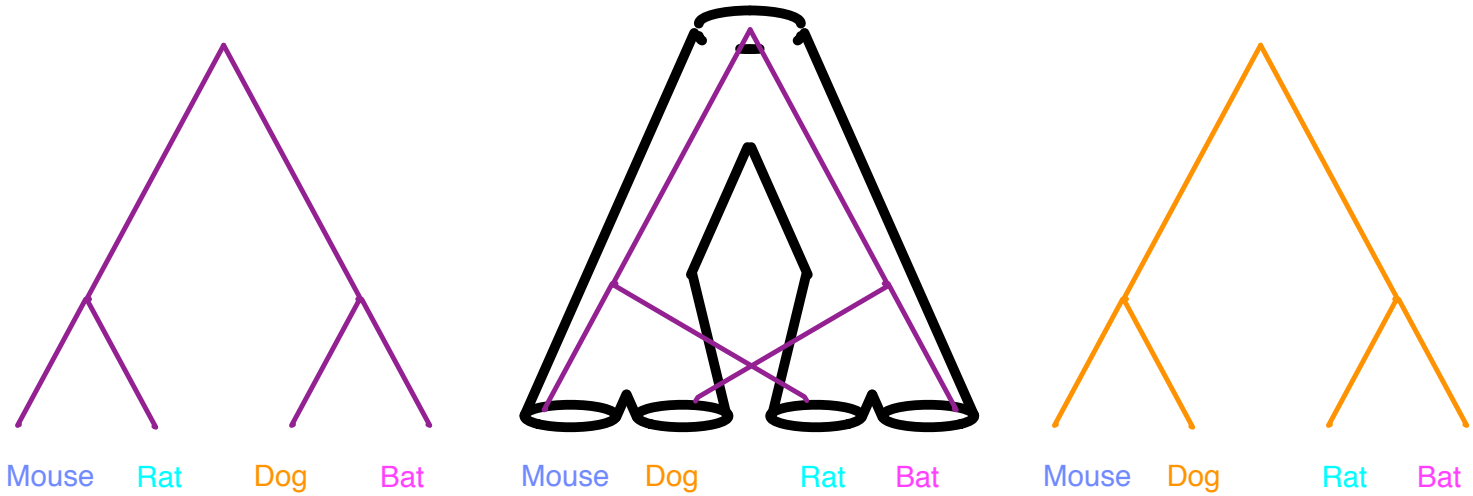
- Often used to estimate species trees.

Gene trees can significantly differ from the species tree for:

- methodological reasons
- biological reasons

Gene trees

- Gene trees are built by analyzing a gene family.



- Often used to estimate species trees.

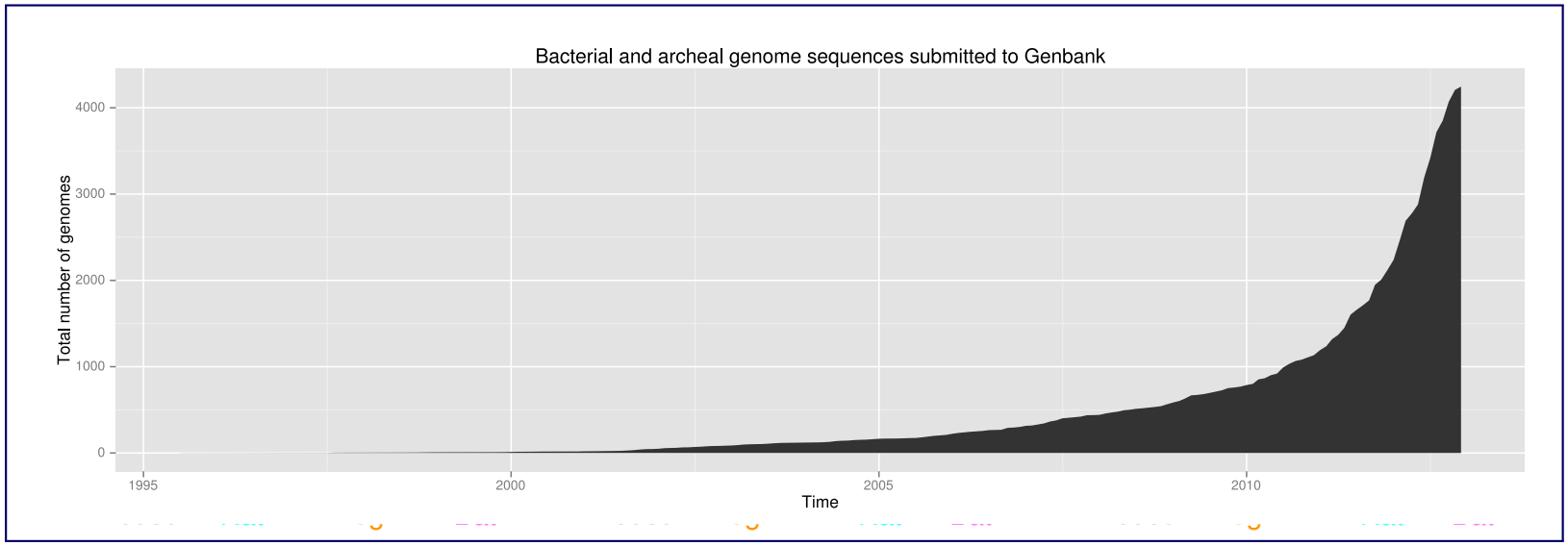
Gene trees can significantly differ from the species tree for:

- methodological reasons
- biological reasons

- We usually use several gene families...

Gene trees

- Gene trees are built by analyzing a gene family.



- Often used to estimate species trees.

Gene trees can significantly differ from the species tree for:

- methodological reasons
- biological reasons

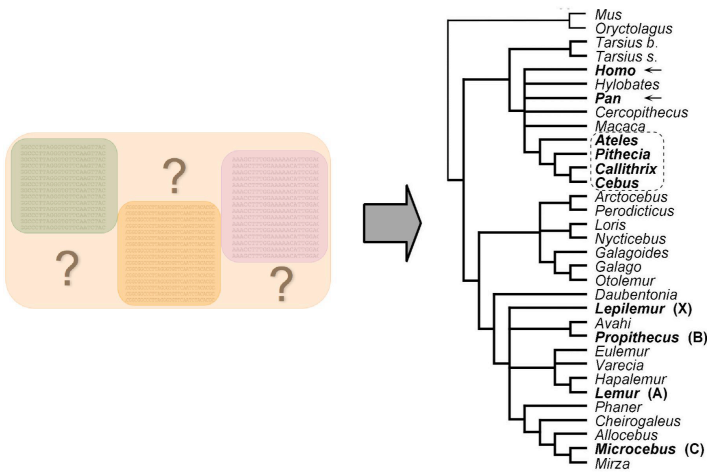
- We usually use several gene families...

Reconstruction of phylogenies for multiple datasets

- from sequences

from alignments or distance matrices

- super matrix (total evidence) methods, i.e. character methods
- distance methods (e.g. SDM)



Reconstruction of phylogenies for multiple datasets

- from sequences

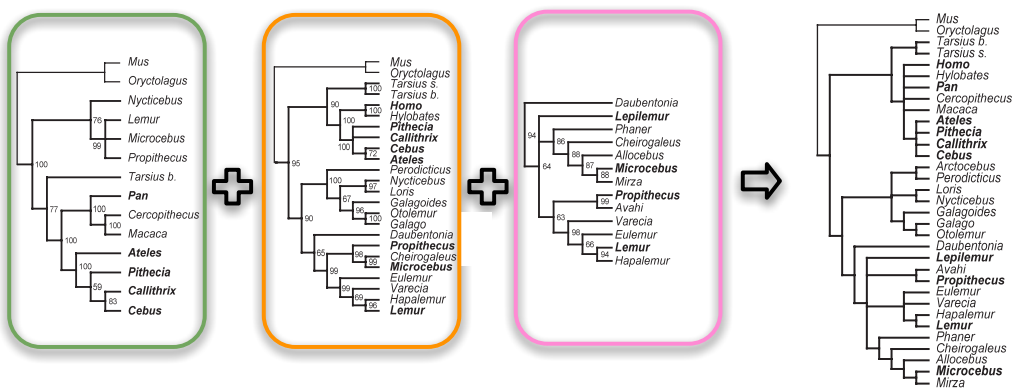
from alignments or distance matrices

- super matrix (total evidence) methods, i.e. character methods
- distance methods (e.g. SDM)

- from trees

via triplets, quartets, clusters, splits etc.

- consensus methods (e.g. majority/strict consensus)
- super tree methods (e.g. MRP)



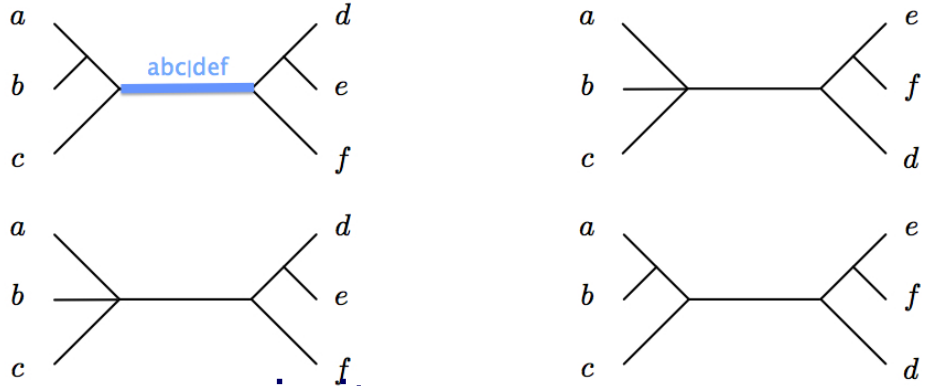
Interest of supertrees

Supertrees are useful for:

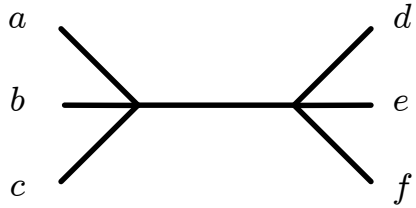
- Combining heterogeneous data
- Obtaining a phylogeny using several genes:
  - Avoids having to deal with too much missing data
  - Evolutionary models adapted for each gene sequence
- Pointing out problematic areas of the phylogeny
  - agreement and disagreement among input trees.
  - measuring taxon overlap

Consensus methods on unrooted trees (bipartitions)

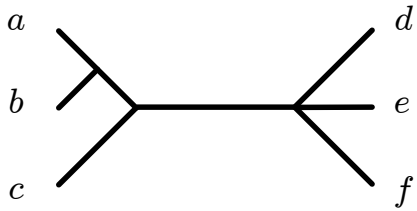
All source trees have the same taxa



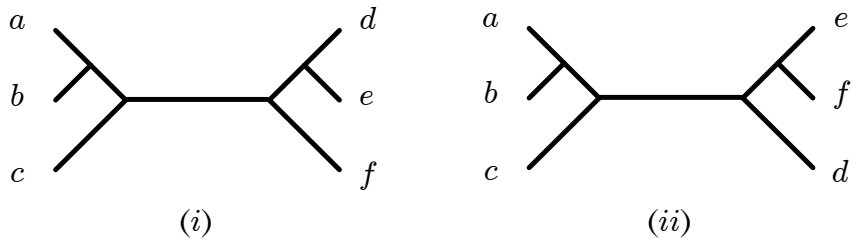
(1) strict consensus, majority consensus



(2) semistrict consensus



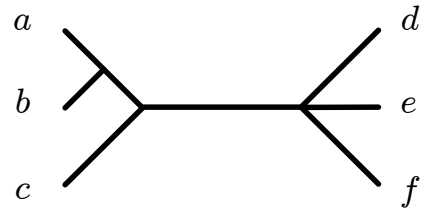
(3) greedy consensus



(4) Nelson consensus

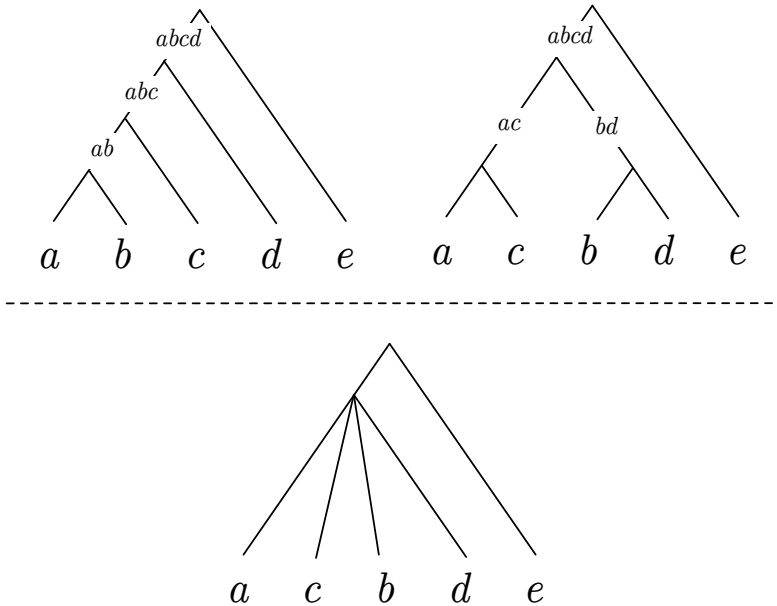
none

(5) Nelson-Page consensus



Consensus methods on rooted trees (clusters)

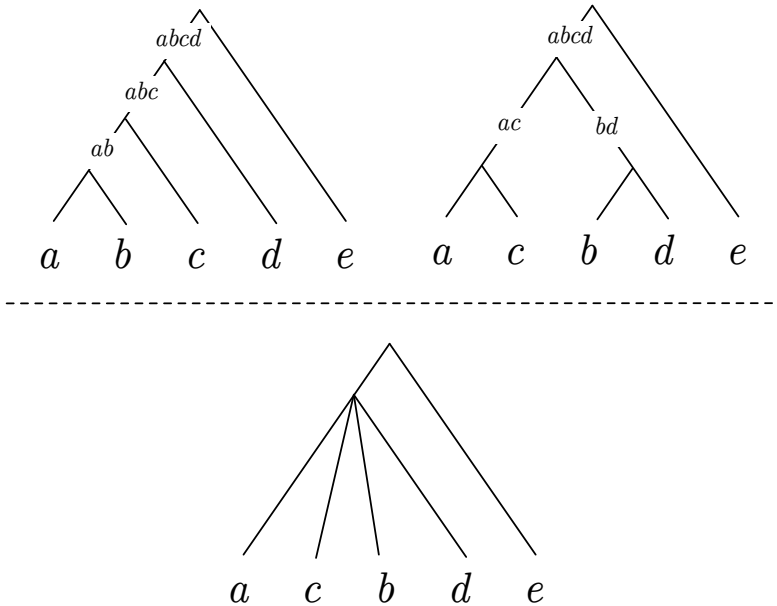
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition.

Consensus methods on rooted trees (clusters)

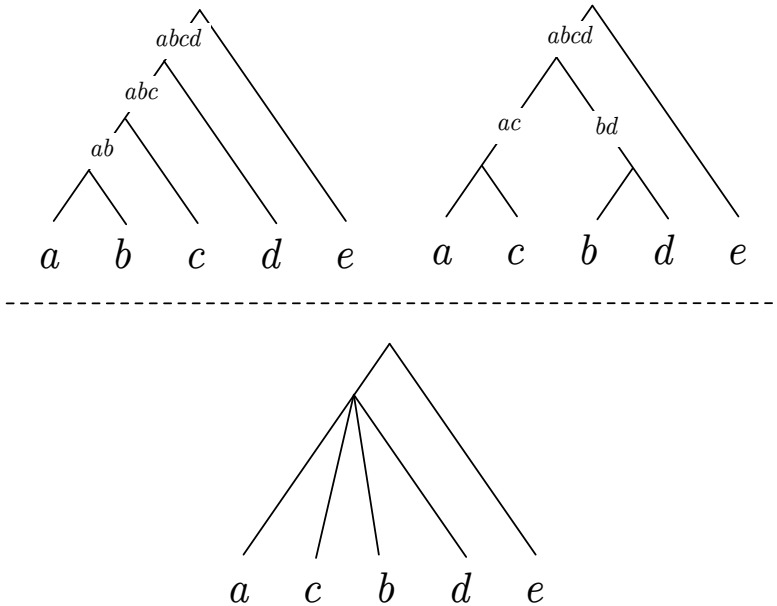
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition. $O(kn^2)$: postfix traversal of each of the k trees, gathering a $O(n)$ information for each of the $O(n)$ nodes.

Consensus methods on rooted trees (clusters)

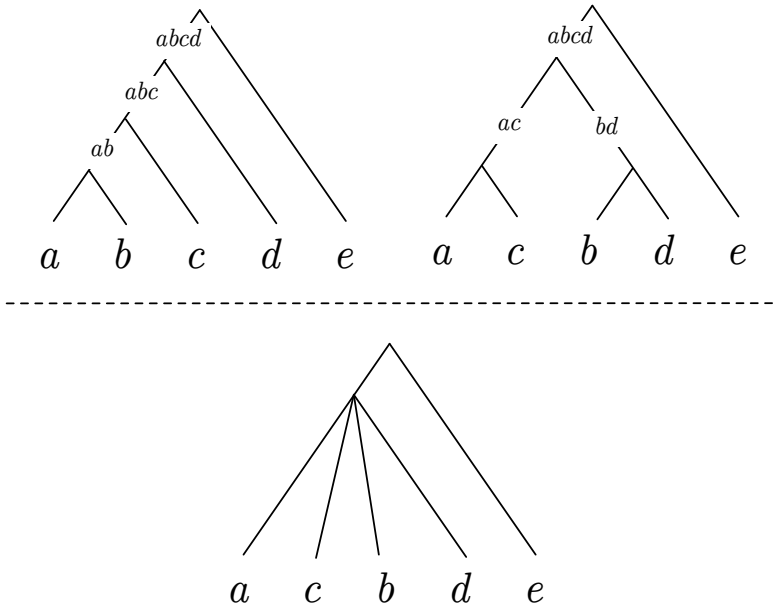
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition. $O(kn^2)$: postfix traversal of each of the k trees, gathering a $O(n)$ information for each of the $O(n)$ nodes.
2. Collect the characters of all branches of the forest in a matrix M.

Consensus methods on rooted trees (clusters)

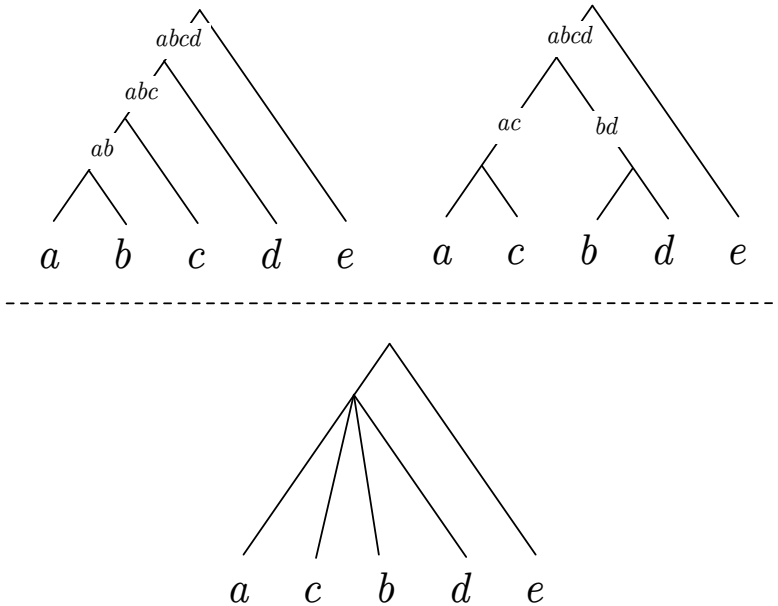
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition. $O(kn^2)$: postfix traversal of each of the k trees, gathering a $O(n)$ information for each of the $O(n)$ nodes.
2. Collect the characters of all branches of the forest in a matrix M. $O(kn^2)$: each of the $O(kn)$ characters is created in $O(n)$ from the list of leaves below the node.

Consensus methods on rooted trees (clusters)

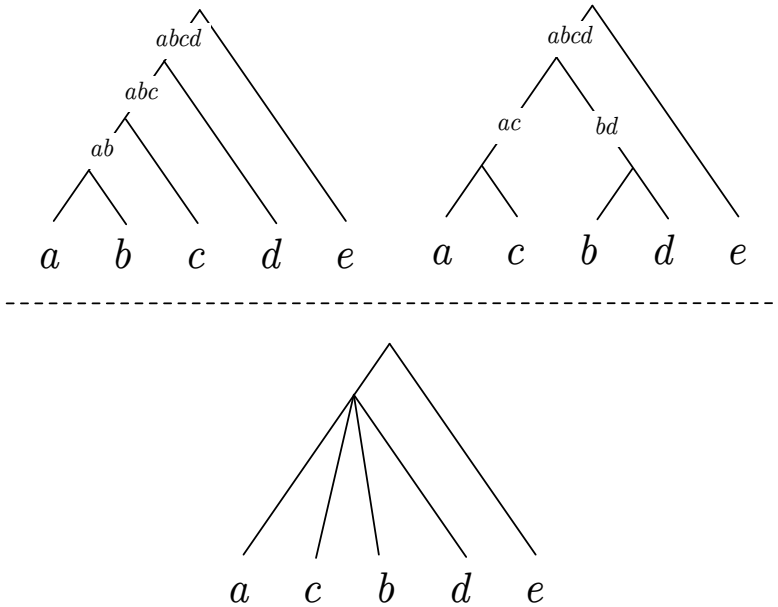
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition. $O(kn^2)$: postfix traversal of each of the k trees, gathering a $O(n)$ information for each of the $O(n)$ nodes.
2. Collect the characters of all branches of the forest in a matrix M. $O(kn^2)$: each of the $O(kn)$ characters is created in $O(n)$ from the list of leaves below the node.
3. Sort the characters in lexicographical order.

Consensus methods on rooted trees (clusters)

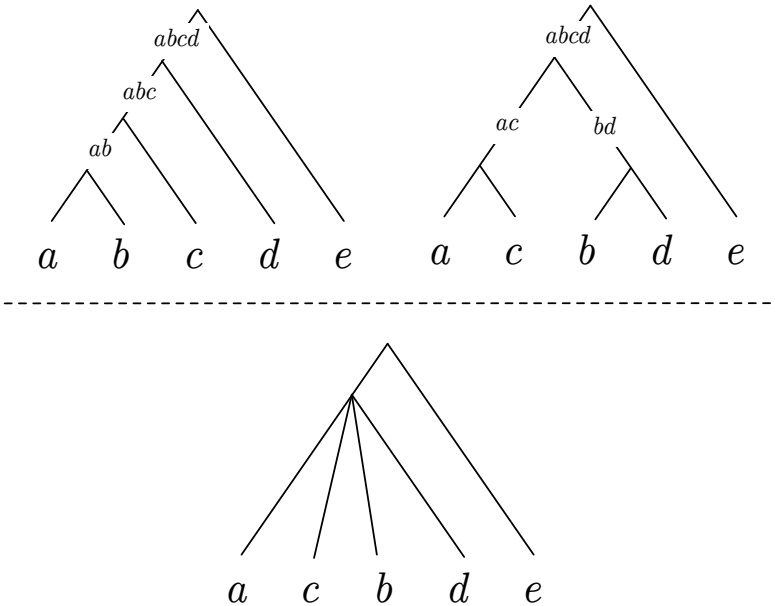
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition. $O(kn^2)$: postfix traversal of each of the k trees, gathering a $O(n)$ information for each of the $O(n)$ nodes.
2. Collect the characters of all branches of the forest in a matrix M. $O(kn^2)$: each of the $O(kn)$ characters is created in $O(n)$ from the list of leaves below the node.
3. Sort the characters in lexicographical order. $O(kn^2)$: radix sort (linear in the input)

Consensus methods on rooted trees (clusters)

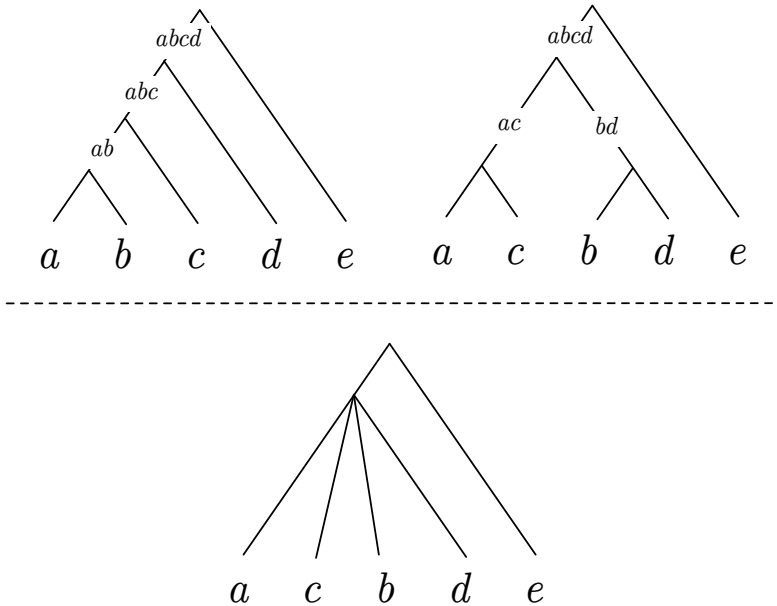
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition. $O(kn^2)$: postfix traversal of each of the k trees, gathering a $O(n)$ information for each of the $O(n)$ nodes.
2. Collect the characters of all branches of the forest in a matrix M. $O(kn^2)$: each of the $O(kn)$ characters is created in $O(n)$ from the list of leaves below the node.
3. Sort the characters in lexicographical order. $O(kn^2)$: radix sort (linear in the input)
4. Find the characters that are present k times in a row.

Consensus methods on rooted trees (clusters)

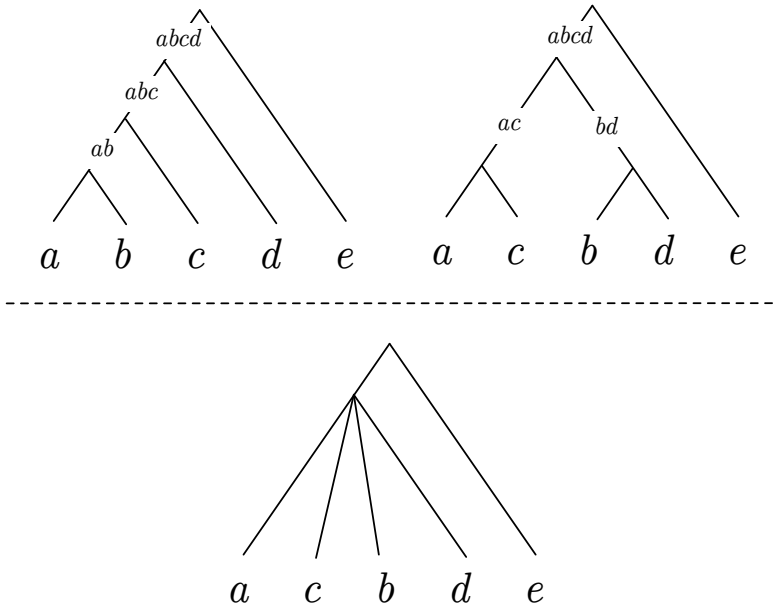
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition. $O(kn^2)$: postfix traversal of each of the k trees, gathering a $O(n)$ information for each of the $O(n)$ nodes.
2. Collect the characters of all branches of the forest in a matrix M. $O(kn^2)$: each of the $O(kn)$ characters is created in $O(n)$ from the list of leaves below the node.
3. Sort the characters in lexicographical order. $O(kn^2)$: radix sort (linear in the input)
4. Find the characters that are present k times in a row. $O(kn^2)$: a simple traversal of the matrix M

Consensus methods on rooted trees (clusters)

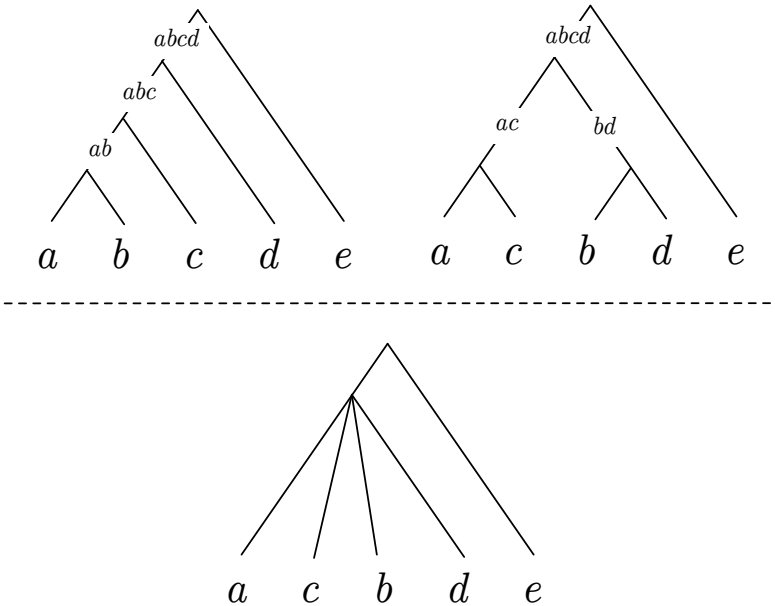
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition. $O(kn^2)$: postfix traversal of each of the k trees, gathering a $O(n)$ information for each of the $O(n)$ nodes.
2. Collect the characters of all branches of the forest in a matrix M. $O(kn^2)$: each of the $O(kn)$ characters is created in $O(n)$ from the list of leaves below the node.
3. Sort the characters in lexicographical order. $O(kn^2)$: radix sort (linear in the input)
4. Find the characters that are present k times in a row. $O(kn^2)$: a simple traversal of the matrix M
5. Find the tree corresponding to the compatible characters.

Consensus methods on rooted trees (clusters)

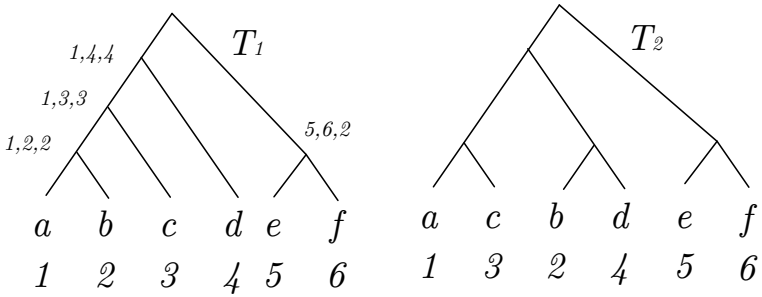
A $O(kn^2)$ algorithm for the strict consensus



1. Code each branch in the tree forest as a bipartition. $O(kn^2)$: postfix traversal of each of the k trees, gathering a $O(n)$ information for each of the $O(n)$ nodes.
2. Collect the characters of all branches of the forest in a matrix M. $O(kn^2)$: each of the $O(kn)$ characters is created in $O(n)$ from the list of leaves below the node.
3. Sort the characters in lexicographical order. $O(kn^2)$: radix sort (linear in the input)
4. Find the characters that are present k times in a row. $O(kn^2)$: a simple traversal of the matrix M
5. Find the tree corresponding to the compatible characters. $O(n^2)$: apply a linear algorithm to construct a tree from a compatibility matrix on the $O(n)$ characters (each of size $O(n)$).

Consensus methods on rooted trees (clusters)

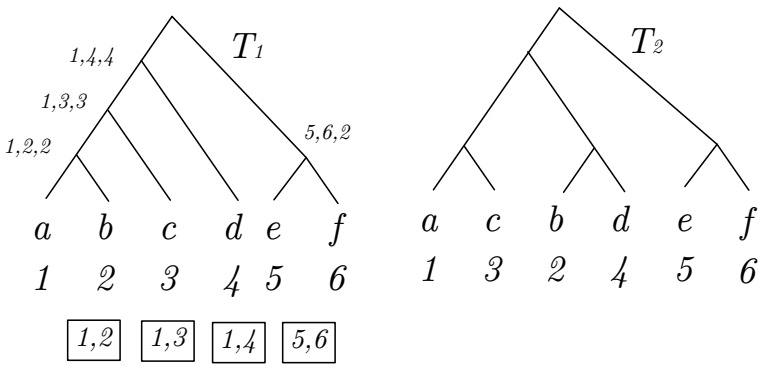
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;

Consensus methods on rooted trees (clusters)

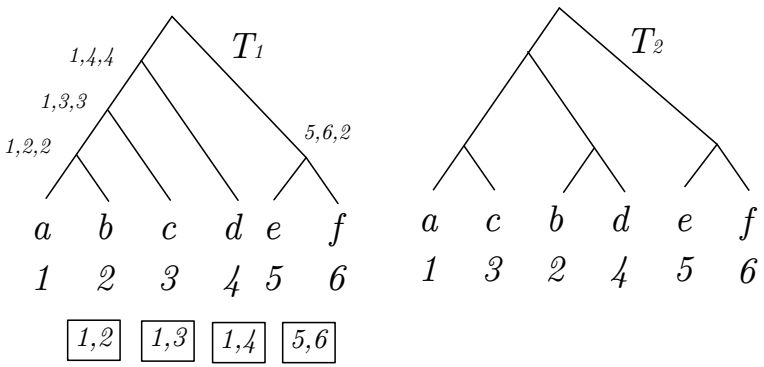
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;

Consensus methods on rooted trees (clusters)

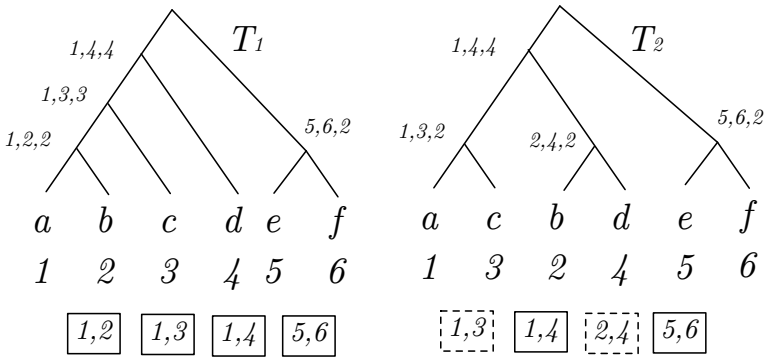
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;
3. Construct the clade list L_1 for T_1, in lexicographical order;

Consensus methods on rooted trees (clusters)

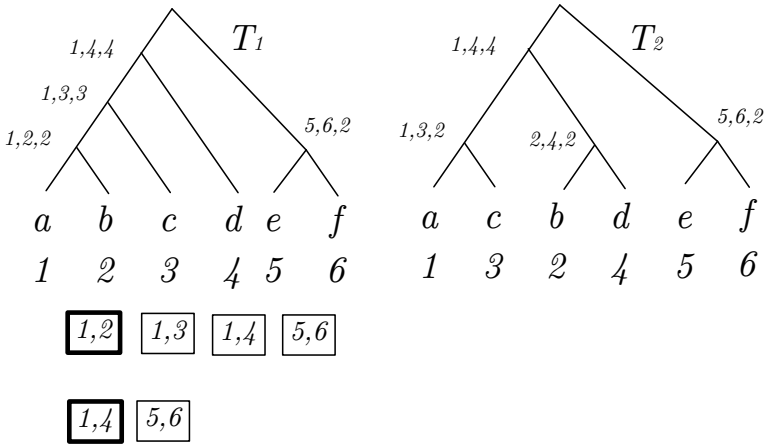
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;
3. Construct the clade list L_1 for T_1, in lexicographical order;
4. Construct the clade list L_2 for T_2, with clades such that $M(u) - m(u) \leq nb(u) - 1$, in lexicographical order;

Consensus methods on rooted trees (clusters)

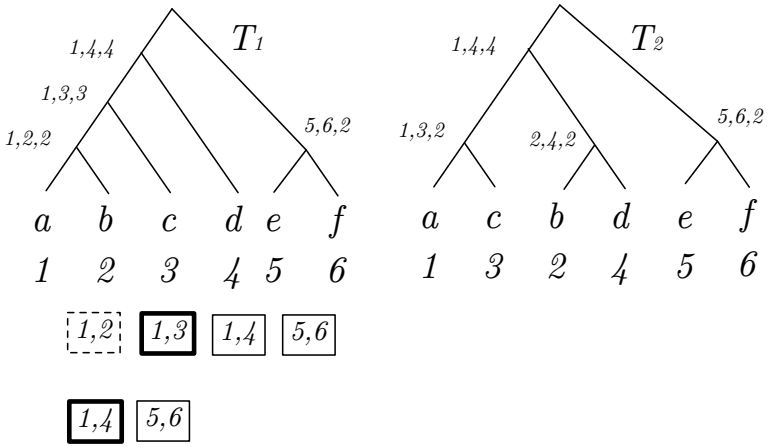
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;
3. Construct the clade list L_1 for T_1, in lexicographical order;
4. Construct the clade list L_2 for T_2, with clades such that $M(u) - m(u) \leq nb(u) - 1$, in lexicographical order;
5. Joint traverse the two lists to keep only the clades present in both;

Consensus methods on rooted trees (clusters)

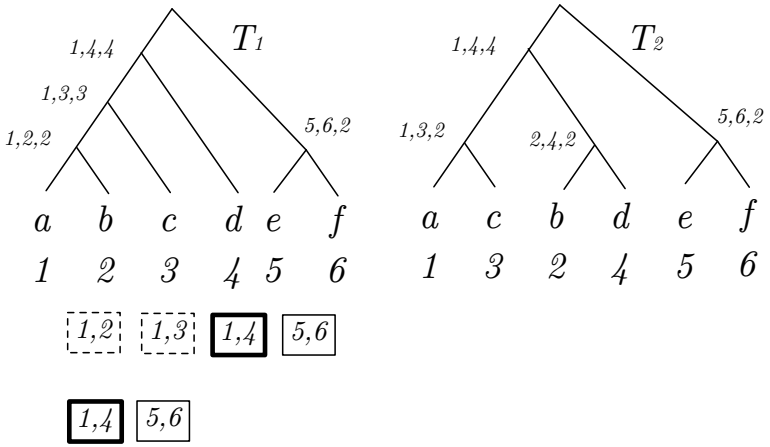
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;
3. Construct the clade list L_1 for T_1, in lexicographical order;
4. Construct the clade list L_2 for T_2, with clades such that $M(u) - m(u) \leq nb(u) - 1$, in lexicographical order;
5. Joint traverse the two lists to keep only the clades present in both;

Consensus methods on rooted trees (clusters)

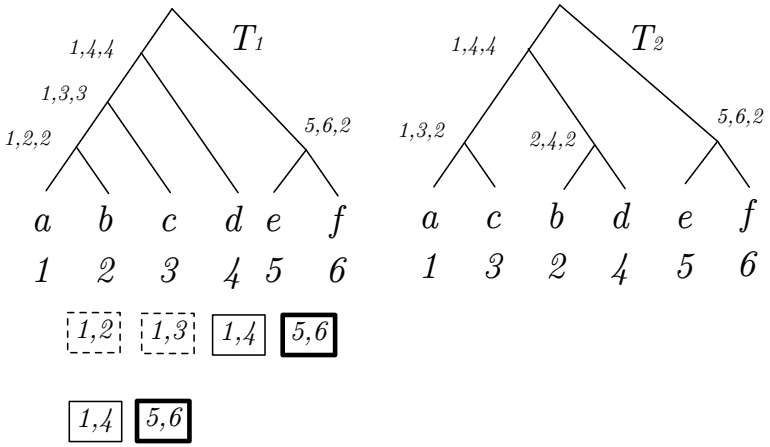
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;
3. Construct the clade list L_1 for T_1, in lexicographical order;
4. Construct the clade list L_2 for T_2, with clades such that $M(u) - m(u) \leq nb(u) - 1$, in lexicographical order;
5. Joint traverse the two lists to keep only the clades present in both;

Consensus methods on rooted trees (clusters)

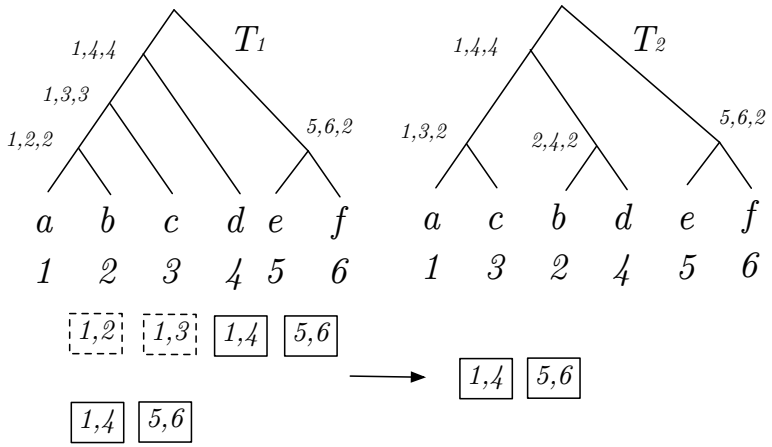
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;
3. Construct the clade list L_1 for T_1, in lexicographical order;
4. Construct the clade list L_2 for T_2, with clades such that $M(u) - m(u) \leq nb(u) - 1$, in lexicographical order;
5. Joint traverse the two lists to keep only the clades present in both;

Consensus methods on rooted trees (clusters)

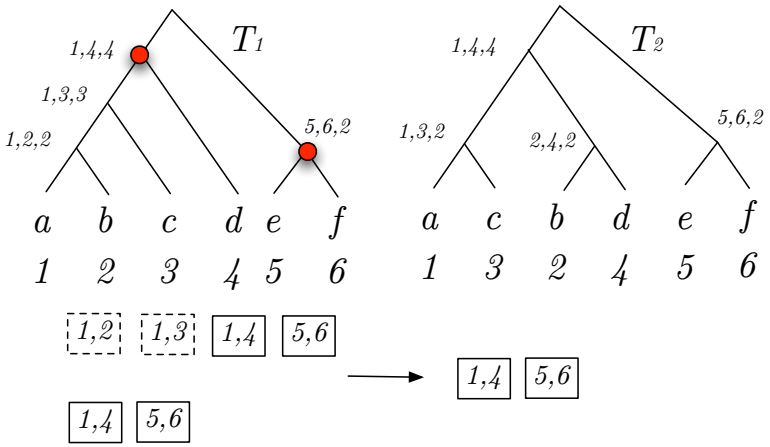
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;
3. Construct the clade list L_1 for T_1, in lexicographical order;
4. Construct the clade list L_2 for T_2, with clades such that $M(u) - m(u) \leq nb(u) - 1$, in lexicographical order;
5. Joint traverse the two lists to keep only the clades present in both;

Consensus methods on rooted trees (clusters)

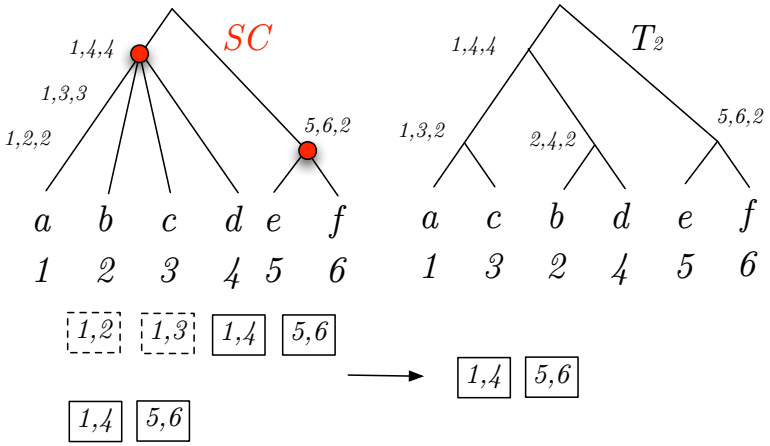
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;
3. Construct the clade list L_1 for T_1, in lexicographical order;
4. Construct the clade list L_2 for T_2, with clades such that $M(u) - m(u) \leq nb(u) - 1$, in lexicographical order;
5. Joint traverse the two lists to keep only the clades present in both;
6. Mark the nodes to keep in T_1;

Consensus methods on rooted trees (clusters)

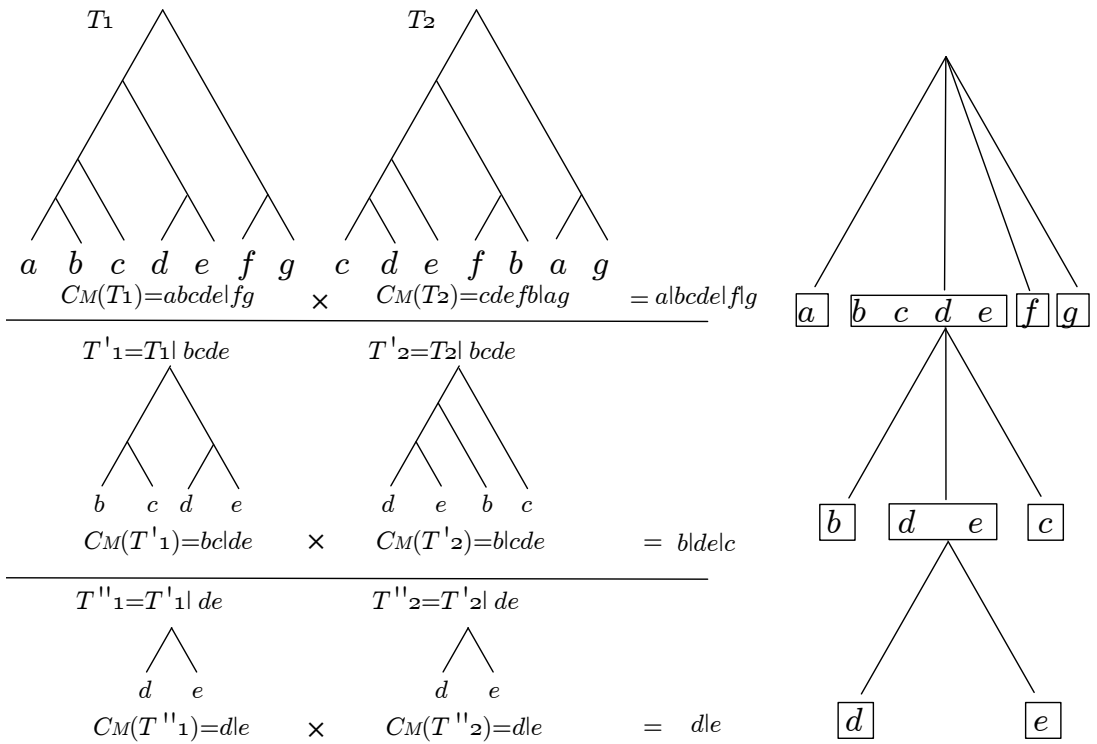
A $O(n)$ algorithm for the strict consensus of 2 trees



1. Number the leaves of T_1 in the left-right order of appearance;
2. For each node u of T_1, compute $M(u)$, $m(u)$, $nb(u)$;
3. Construct the clade list L_1 for T_1, in lexicographical order;
4. Construct the clade list L_2 for T_2, with clades such that $M(u) - m(u) \leq nb(u) - 1$, in lexicographical order;
5. Joint traverse the two lists to keep only the clades present in both;
6. Mark the nodes to keep in T_1;
7. Collapse the other nodes to obtain the strict consensus.

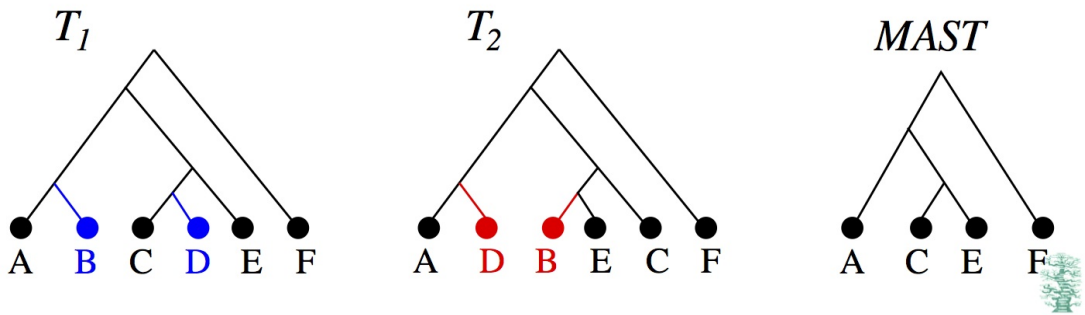
Consensus methods on rooted trees (clusters)

A $O(n)$ algorithm for the Adams consensus of 2 rooted trees



Consensus methods on rooted trees (MAST)

The MAST (Maximum Agreement SubTree) problem consists in finding the maximum agreement subset tree for a forest $\mathcal{F}$. Given a forest of trees $\mathcal{F}$, an agreement subtree T is a tree such that $L(T) \subseteq L(\mathcal{F})$ and $T = T_i|_{L(T)}$, $\forall T_i \in \mathcal{F}$.



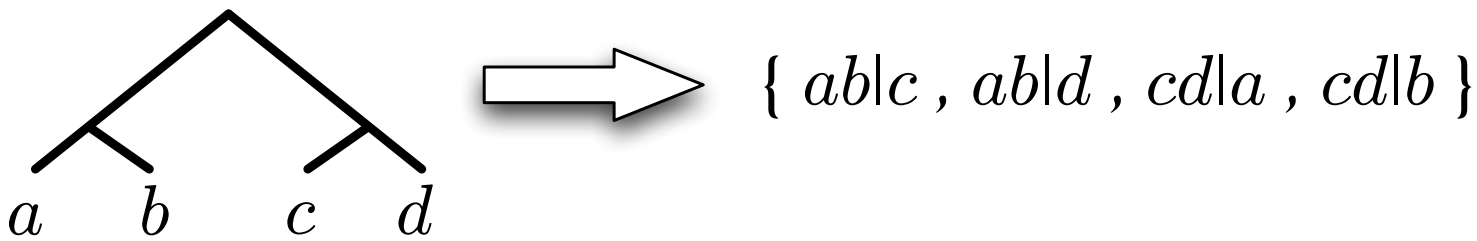
Consensus methods on rooted trees (MAST)

The MAST (Maximum Agreement SubTree) problem consists in finding the maximum agreement subset tree for a forest $\mathcal{F}$. Given a forest of trees $\mathcal{F}$, an agreement subtree T is a tree such that $L(T) \subseteq L(\mathcal{F})$ and $T = T_i|_{L(T)}, \forall T_i \in \mathcal{F}$.

- an $O(\sqrt{d}nlog(n)$ algorithm [Przytycka, 1997] and an $O(\sqrt{d}nlog^2(\frac{n}{d})$ algorithm [Kao et al., 2001] for two rooted trees, where $n = |L(\mathcal{F})|$ and d is the maximum degree of the input tree nodes;
- an $O(n^{1.5})$ algorithm for two unrooted trees [Kao et al., 1999].
- NP-hard for more than two trees
- can be solved in polynomial time for forests with bounded degree [e.g., Bryant, 1997; Guillemot and Nicolas, 2006].
- an FPT algorithm has been proposed for unbounded degree forests [Guillemot and Nicolas, 2006].

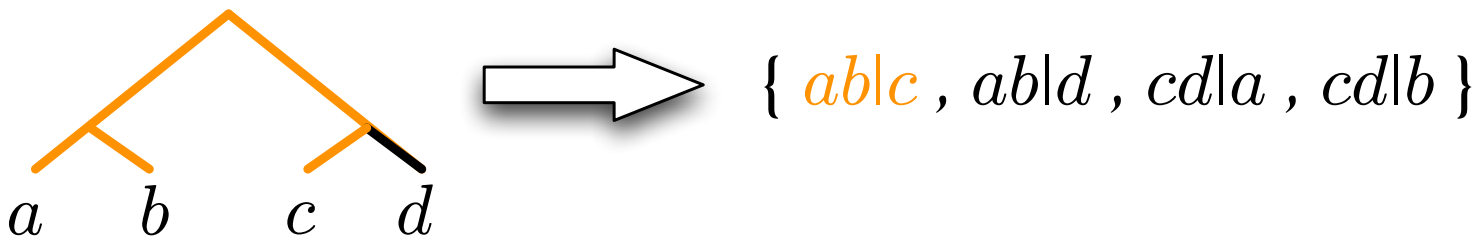
Triplet information

- Any rooted tree T can be decomposed in the set of its triplets, denoted by $\mathcal{R}(T)$.



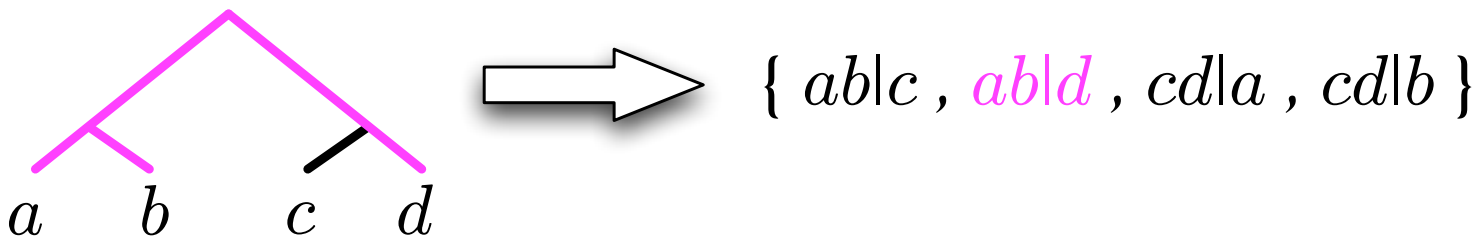
Triplet information

- Any rooted tree T can be decomposed in the set of its triplets, denoted by $\mathcal{R}(T)$.



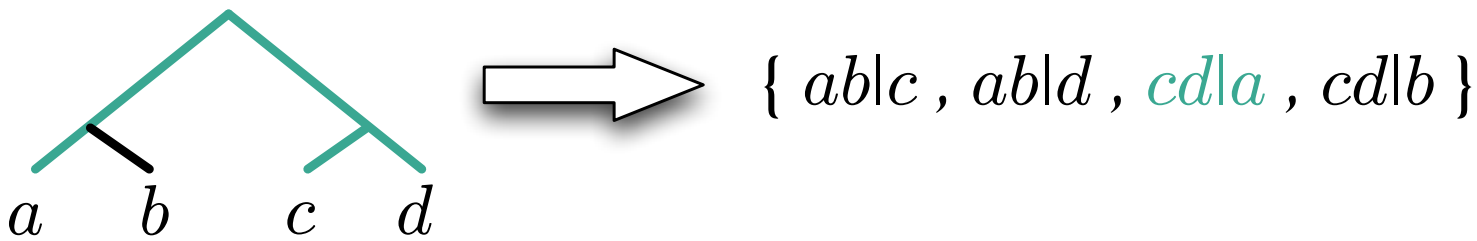
Triplet information

- Any rooted tree T can be decomposed in the set of its triplets, denoted by $\mathcal{R}(T)$.



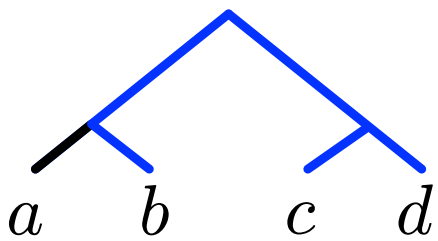
Triplet information

- Any rooted tree T can be decomposed in the set of its triplets, denoted by $\mathcal{R}(T)$.



Triplet information

- Any rooted tree T can be decomposed in the set of its triplets, denoted by $\mathcal{R}(T)$.



$\{\ ab|c\ ,\ ab|d\ ,\ cd|a\ ,\ cd|b\ \}$

Triplet information

- Any rooted tree T can be decomposed in the set of its triplets, denoted by $\mathcal{R}(T)$.
- The set of triplets of a forest of trees $\mathcal{T}$ is denoted by $\mathcal{R}(\mathcal{T})$.

Triplet information

- Any rooted tree T can be decomposed in the set of its triplets, denoted by $\mathcal{R}(T)$.
- The set of triplets of a forest of trees $\mathcal{T}$ is denoted by $\mathcal{R}(\mathcal{T})$.
- A conflict of $\mathcal{F}$ is a set of three leaves $\{a, b, c\}$ such that $Ti, Tj \in \mathcal{F}$, $T_i|\{a, b, c\} \neq T_j|\{a, b, c\}$.

Property

Two binary rooted trees T_1 and T_2 are isomorphic iff $\mathcal{R}(T_1) = \mathcal{R}(T_2)$

Corollary

Two binary rooted trees T_1 and T_2 are isomorphic iff there are no conflicts of $\mathcal{F}$.

Triplet information

- Any rooted tree T can be decomposed in the set of its triplets, denoted by $\mathcal{R}(T)$.
- The set of triplets of a forest of trees $\mathcal{T}$ is denoted by $\mathcal{R}(\mathcal{T})$.
- A conflict of $\mathcal{F}$ is a set of three leaves $\{a, b, c\}$ such that $Ti, Tj \in \mathcal{F}$, $T_i|\{a, b, c\} \neq T_j|\{a, b, c\}$.

Property

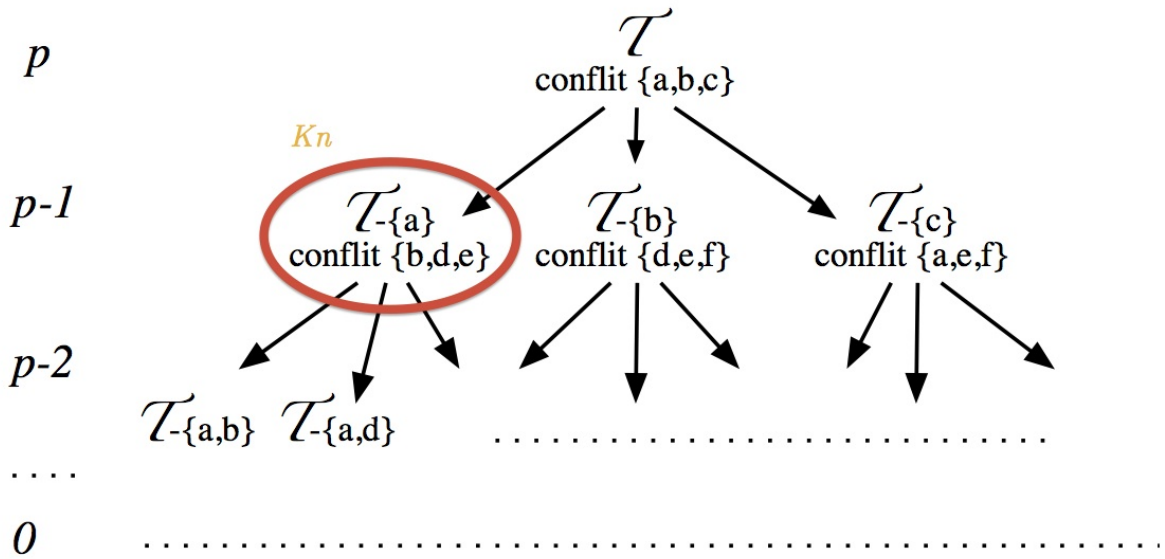
Two binary rooted trees T_1 and T_2 are isomorphic iff $\mathcal{R}(T_1) = \mathcal{R}(T_2)$

Corollary

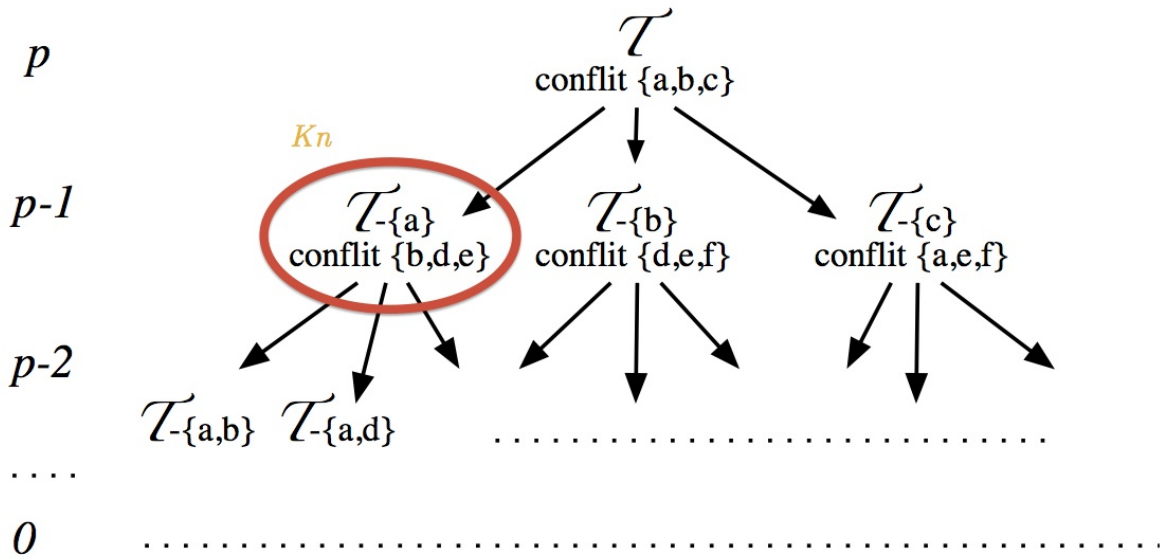
Two binary rooted trees T_1 and T_2 are isomorphic iff there are no conflicts of $\mathcal{F}$.

- Conclusion : if $\{a, b, c\}$ is a conflict of $\mathcal{F}$, then any MAST cannot contain these three leaves at once (thus we have to remove at least one of these leaves).

A FPT algorithm in ρ for the rooted MAST



A FPT algorithm in ρ for the rooted MAST



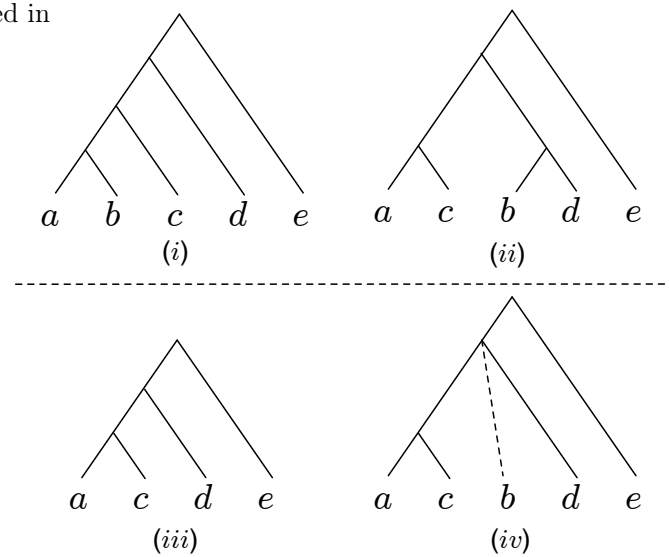
The search tree has $O(3^p)$ nodes, so the overall complexity is $O(3^p kn)$.

The common pruned-and-regrafted tree (CPRT)

For rooted forests, terminal taxa not included in a MAST tree can be regrafted to the tree.

Data: A forest of rooted trees $\mathcal{F} = \{T_1, T_2, \cdots, T_k\}$.
Result: The set of clusters $\mathcal{C}$ induced by the CPRT of $\mathcal{F}$, as defined in Gordon [1980].

```
1  C ← ∅;
2  T_M ← MAST(F);
3  L' ← L(F) − L(T_M);
4  foreach (cluster C_i ∈ C(T_M)) do
5      A_i ← C_i;
6      foreach (taxon l_j ∈ L') do
7          T_S ← strict consensus tree for F|_(L(T_M)∪l_j);
8          if ({C_i ∪ l_j} ∈ C(T_S)) then
9              A_i ← A_i ∪ l_j;
10     C ← C ∪ A_i;
11 T_SC ← strict consensus tree for F;
12 return C ∪ C(T_SC);
```

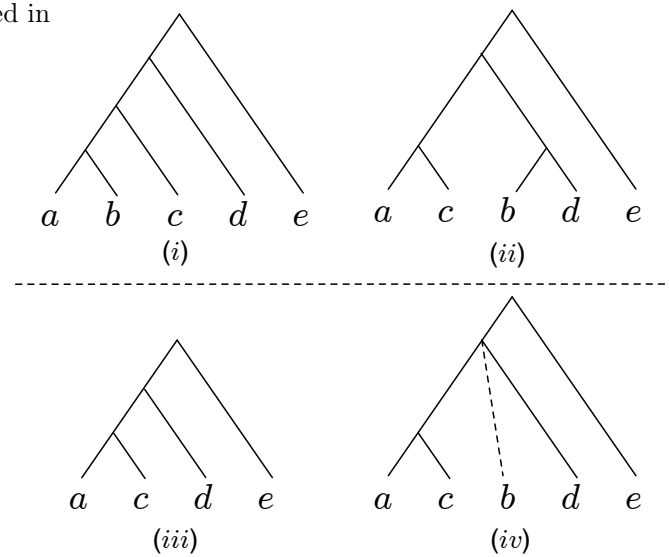


The common pruned-and-regrafted tree (CPRT)

For rooted forests, terminal taxa not included in a MAST tree can be regrafted to the tree.

Data: A forest of rooted trees $\mathcal{F} = \{T_1, T_2, \cdots, T_k\}$.
Result: The set of clusters $\mathcal{C}$ induced by the CPRT of $\mathcal{F}$, as defined in Gordon [1980].

```
1  C ← ∅;
2  T_M ← MAST(F);
3  L' ← L(F) − L(T_M);
4  foreach (cluster C_i ∈ C(T_M)) do
5      A_i ← C_i;
6      foreach (taxon l_j ∈ L') do
7          T_S ← strict consensus tree for F|_(L(T_M)∪l_j);
8          if ({C_i ∪ l_j} ∈ C(T_S)) then
9              A_i ← A_i ∪ l_j;
10     C ← C ∪ A_i;
11 T_SC ← strict consensus tree for F;
12 return C ∪ C(T_SC);
```



The CPRT might not be unique. It can be computed only for collections of rooted trees since its computation requires the use of clusters.

Supertree methods – DECISION methods

Supertree methods can be classified depending on the way they deal with incongruent data:

Supertree methods – DECISION methods

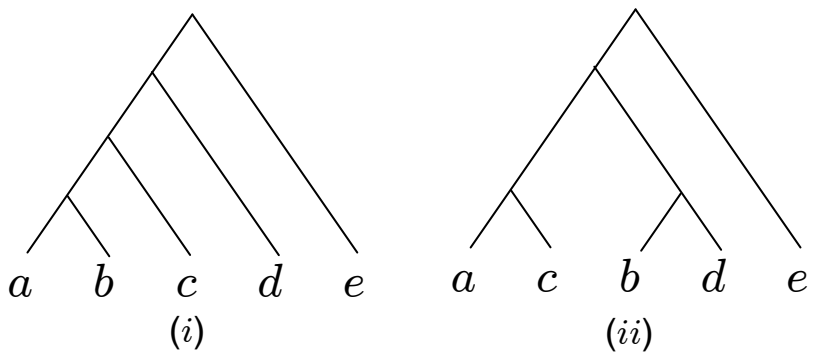
Supertree methods can be classified depending on the way they deal with incongruent data:

1. The first suite of methods cannot handle incompatible source trees, e.g., Build (Aho et al., 1981).

Supertree methods – DECISION methods

Supertree methods can be classified depending on the way they deal with incongruent data:

1. The first suite of methods cannot handle incompatible source trees, e.g., Build (Aho et al., 1981).
   - Since phylogenies usually conflict with one another, those methods are of limited use.



Build or OneTree algorithm

Data: A triplet set $\mathcal{R}$ on a leaf set L.
Result: A tree $T : L(T) = L$ and $\mathcal{R} \subseteq \mathcal{R}(T)$ or a statement that no such a tree exists.

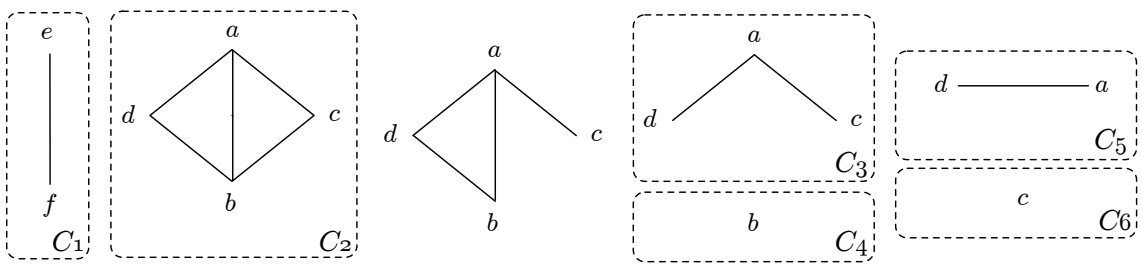
```
1  if (|L| = 1) then return a single node labeled by the label of L;
2  else
3      if (|L| = 2) then
4          return a tree with two leaves respectively labeled by the labels of L;
5      else
6          create a new tree T composed by a single unlabeled node r;
7          construct G(R, L);
8          if (|CC(G(R, L))| = 1) then return "no such a tree exists" ;
9          else
10             foreach (connected component C_i ∈ CC(G(R, L))) do
11                 if (Build(R|_{V(C_i)}, V(C_i)) returns a tree T_{C_i} ) then
12                     add the root node of T_{C_i} as son of r in T;
13                 else
14                     return "no such a tree exists";
15         return T;
```

Build or OneTree algorithm

Data: A triplet set $\mathcal{R}$ on a leaf set L.
Result: A tree $T : L(T) = L$ and $\mathcal{R} \subseteq \mathcal{R}(T)$ or a statement that no such a tree exists.

```
1  if (|L| = 1) then return a single node labeled by the label of L;
2  else
3      if (|L| = 2) then
4          return a tree with two leaves respectively labeled by the labels of L;
5      else
6          create a new tree T composed by a single unlabeled node r;
7          construct G(R, L);
8          if (|CC(G(R, L))| = 1) then return "no such a tree exists" ;
9          else
10             foreach (connected component C_i ∈ CC(G(R, L))) do
11                 if (Build(R|_{V(C_i)}, V(C_i)) returns a tree T_{C_i} ) then
12                     add the root node of T_{C_i} as son of r in T;
13                 else
14                     return "no such a tree exists";
15         return T;
```

- $O(\mathcal{R}(\mathcal{F}))L(\mathcal{F}))$
- $mlog^2(L(\mathcal{F}))$, with m number of interior vertices in $\mathcal{F}$ for binary trees

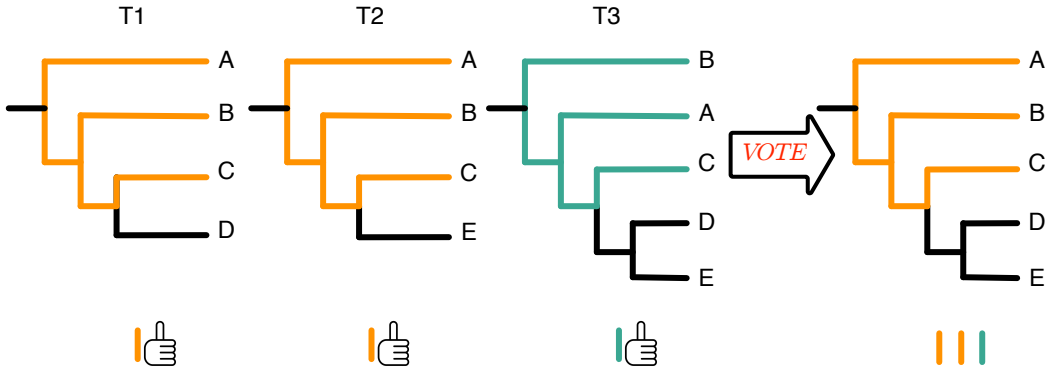


(i) (ii) (iii) (iv)

1. $(((a, c), b), (e, f))$ and $(((a, d), b), c)$ is none
2. $(((a, c), b), (e, f))$ and $((a, d), b, c)$ is $((((a, d), c), b), (e, f))$.

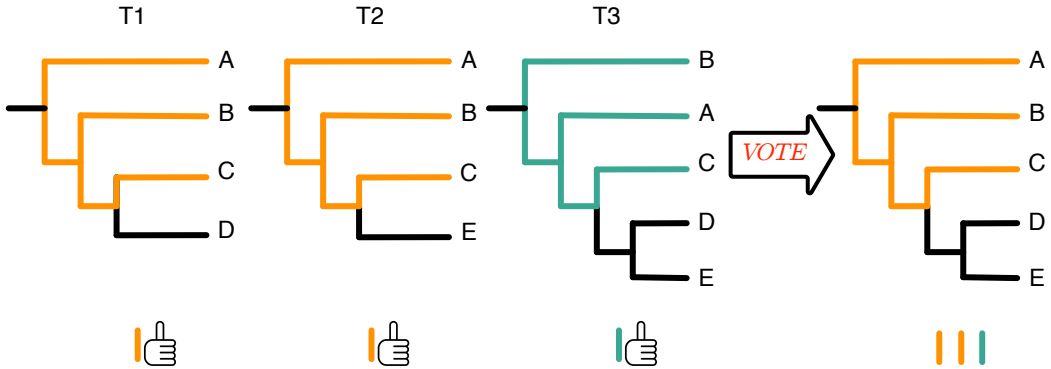
Supertree methods – VOTE methods

2. Vote methods resolve conflicts, opting for the resolution that maximizes their optimization criteria



Supertree methods – VOTE methods

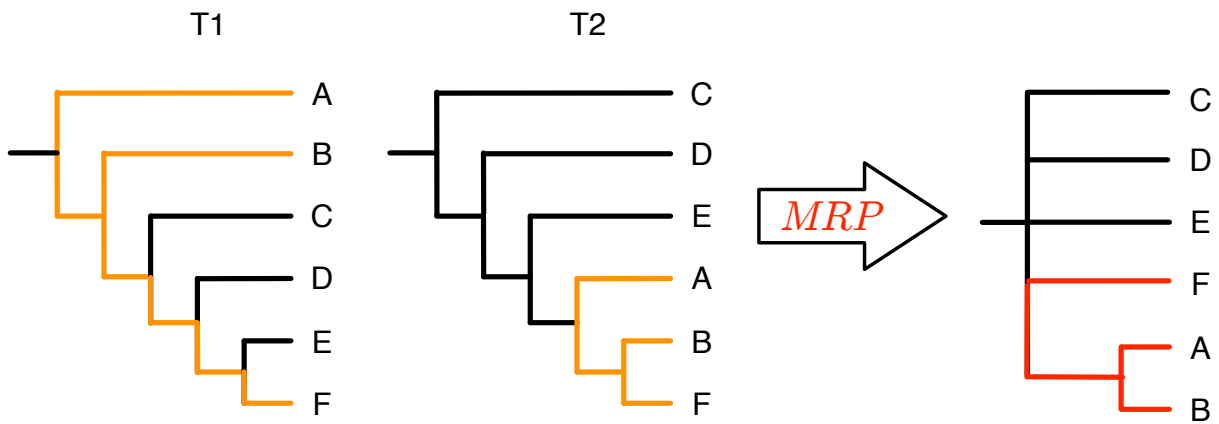
2. Vote methods resolve conflicts, opting for the resolution that maximizes their optimization criteria



- pros: usually, highly resolved and accurate supertrees.

Supertree methods – VOTE methods

2. Vote methods resolve conflicts, opting for the resolution that maximizes their optimization criteria
   - pros: usually, highly resolved and accurate supertrees.
   - cons: sometimes ... clades contradicting all source trees (Goloboff and Pol, 2002).



Variants of the Build algorithm

- MinCut (MC), Modified MC and strict consensus supertrees

Algorithm 4: $\text{MC}(\mathcal{F}, w)$

Data: A set of rooted trees $\mathcal{F}$ and a weighed function $w : \{1, \cdots, |\mathcal{F}|\} \to (\mathbb{Q}^+)^{|\mathcal{F}|}$.

Result: A tree T_{MC} that is the MC supertree for the pair $(\mathcal{F}, w)$.

1 **if** $(|L(\mathcal{F})| = 1)$ **then** return a single node labeled by the label of $L(\mathcal{F})$;
2 **else**
3 **if** $(|L(\mathcal{F})| = 2)$ **then**
4 return a tree with two leaves respectively labeled by the labels of $L(\mathcal{F})$;
5 **else**
6 create a new tree T_{MC} composed by an unlabeled node r;
7 construct $\mathcal{G}(\mathcal{R}(\mathcal{F}), L(\mathcal{F}))$;
8 **if** $(|CC(\mathcal{G}(\mathcal{R}(\mathcal{F}), L(\mathcal{F})))| = 1)$ **then**
9 construct $\mathcal{G}(\mathcal{F}, w)$;
10 $\mathcal{G} \leftarrow \hat{\mathcal{G}}(\mathcal{F}, w)$;
11 construct the set E' of edges of $\mathcal{G}$ that lie in at least one minimum-weight cut set of $\mathcal{G}$;
12 $\mathcal{C} \leftarrow CC(\mathcal{G} \backslash E')$;
13 **else**
14 $\mathcal{C} \leftarrow CC(\mathcal{G}(\mathcal{R}(\mathcal{F}), L(\mathcal{F})))$;
15 **foreach** (*connected component* $C_i \in \mathcal{C}$) **do**
16 $T_{C_i} \leftarrow \text{MC}(\mathcal{F}|_{V(C_i)}, w))$;
17 add the root node of T_{C_i} as son of r in T_{MC};
18 **return** T_{MC};

Variants of the Build algorithm

- MinCut (MC), Modified MC and strict consensus supertrees

Algorithm 4: $\text{MC}(\mathcal{F}, w)$

Data: A set of rooted trees $\mathcal{F}$ and a weigh
$w : \{1, \cdots, |\mathcal{F}|\} \to (\mathbb{Q}^+)^{|\mathcal{F}|}$.
Result: A tree T_{MC} that is the MC supertr

```
1  if (|L(F)| = 1) then return a single node
2  else
3      if (|L(F)| = 2) then
4          return a tree with two leaves respec
           L(F);
5      else
6          create a new tree T_MC composed by
7          construct G(R(F), L(F));
8          if (|CC(G(R(F), L(F)))| = 1) ther
9              construct G(F, w);
10             G ← Ĝ(F, w);
11             construct the set E' of edges of
               minimum-weight cut set of G;
12             C ← CC(G\E');
13         else
14             C ← CC(G(R(F), L(F)));
15         foreach (connected component C_i ∈ C) do
16             T_{C_i} ← MC(F|_{V(C_i)}, w));
17             add the root node of T_{C_i} as son of r in T_MC;
18 return T_MC;
```

Algorithm 5: $\mathcal{G}_{\text{MMC}}(\mathcal{F}, \hat{\mathcal{G}}(\mathcal{F}, w))$

Data: A set of rooted trees $\mathcal{F}$ and a weighted graph $\hat{\mathcal{G}}(\mathcal{F}, w)$.
Result: A weighted graph $\mathcal{G}_{\text{MMC}}$.

```
1  G_MMC ← Ĝ(F, w);
2  E' ← ∅;
3  compute the set C of contradicted edges in G_MMC;
4  if (G_MMC\C is disconnected) then
5      find the connected components of G_MMC\C;
6      E' ← edges connecting two nodes of the same connected component;
7  else
8      build the set AC of all edges in G_MMC adjacent to a contradicted edge.;
9      if (G_MMC\(AC ∩ C) is disconnected) then
10         find the components of G_MMC\(AC ∩ C);
11         E' ← edges connecting two nodes of the same connected component;
12 return Ĝ(F, w) ⊙ E';
```

Variants of the Build algorithm

- MinCut (MC), Modified MC and strict consensus supertrees

Algorithm 4: MC$(\mathcal{F}, w)$

Data: A set of rooted trees $\mathcal{F}$ and a weigh
$w : \{1, \cdots, |\mathcal{F}|\} \rightarrow (\mathbb{Q}^+)^{|\mathcal{F}|}$.
Result: A tree T_{MC} that is the MC supertr

```
1  if (|L(F)| = 1) then return a single node
2  else
3      if (|L(F)| = 2) then
4          return a tree with two leaves respec
           L(F);
5      else
6          create a new tree T_MC composed by
7          construct G(R(F), L(F));
8          if (|CC(G(R(F), L(F)))| = 1) ther
9              construct Ĝ(F, w);
10             G ← Ĝ(F, w);
11             construct the set E' of edges of
               minimum-weight cut set of G;
12             C ← CC(G\E');
13         else
14             C ← CC(G(R(F), L(F)));
15         foreach (connected component C_i ∈
16             T_{C_i} ← MC(F|_{V(C_i)}, w));
17             add the root node of T_{C_i} as son
18 return T_MC;
```

Algorithm 5: $\mathcal{G}_{\text{MMC}}(\mathcal{F}, \hat{\mathcal{G}}(\mathcal{F}, w))$

Data: A set of rooted trees $\mathcal{F}$ and a weighted graph $\hat{\mathcal{G}}(\mathcal{F}, w)$.
Result: A weighted graph $\mathcal{G}_{\text{MMC}}$.

```
1  G_MMC ← Ĝ(F, w);
2  E' ← ∅;
3  compute the set C of contradicted edges in G_MMC;
4  if (G_MMC\C is disconnected) then
5      find the connected components of G_MMC\C;
6      E' ← edges connecting two nodes of the same connected component;
7  else
```

Algorithm 6: SCS$(\mathcal{F}, w)$

Data: A set of rooted trees $\mathcal{F}$.
Result: A tree T_{SCS} that is the strict consensus supertree for $\mathcal{F}$.

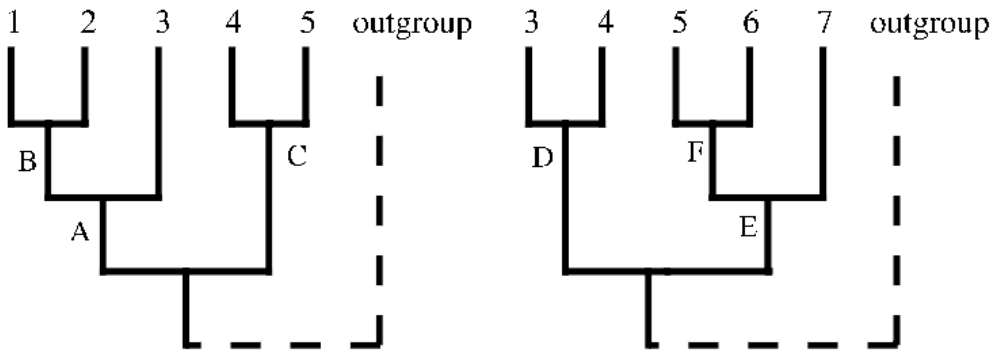
```
1  BC ← ∅ // Bad Clusters set;
2  T ← Build(R(F), L(F));
3  foreach (cluster C_i ∈ C(T)) do
4      x ← a leaf of C_i;
5      foreach (a, b ∈ L(F) such that a ∈ C_i − {x}, b ∈ L(F) − L(C_i)) do
6          if !(G(R(F) ∪ {ab|x}, L(F)) and G(R(F) ∪ {bx|a}, L(F))
           incompatible) then
7              BC ← BC ∪ C_i;
8  build the tree T_SCS such that C(T_SCS) = C(T) − BC;
9  return T_SCS;
```

$O(n6 \cdot log\hat{}2(n))$

MRP (Matrix Representation with Parsimony)

The clades of source trees (a) are coded as a matrix M of binary characters (plus question marks).

(a)



(b)

Taxon	Component A	B	C	D	E	F
outgroup	0	0	0	0	0	0
1	1	1	0	?	?	?
2	1	1	0	?	?	?
3	1	0	0	1	0	0
4	0	0	1	1	0	0
5	0	0	1	0	1	1
6	?	?	?	0	1	1
7	?	?	?	0	1	0

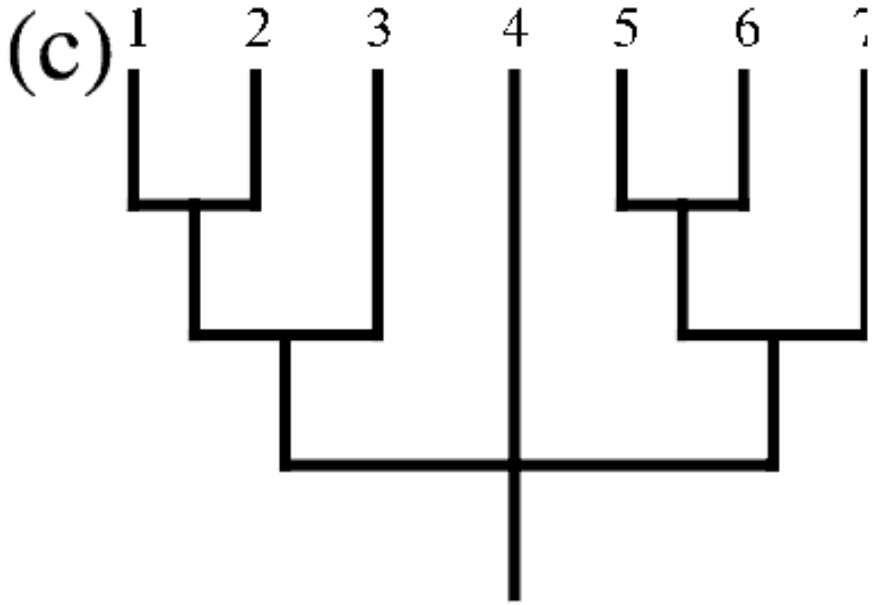
MRP : second step

We apply a maximum parsimony algorithm to M to obtain a supertree (c).

(b)

Taxon	Component A	B	C	D	E	F
outgroup	0	0	0	0	0	0
1	1	1	0	?	?	?
2	1	1	0	?	?	?
3	1	0	0	1	0	0
4	0	0	1	1	0	0
5	0	0	1	0	1	1
6	?	?	?	0	1	1
7	?	?	?	0	1	0

(c)



MRP : properties et problems

We can obtain an

exponential number of most parsimonious trees.
$\Longrightarrow$ In this case, MRP returns the strict consensus of the most parsimonious trees (to avoid to take arbitrary decisions).

MRP : properties et problems

We can obtain an

exponential number of most parsimonious trees.
$\Longrightarrow$ In this case, MRP returns the strict consensus of the most parsimonious trees (to avoid to take arbitrary decisions).

Properties

If $\mathcal{T}$ is compatible, we obtain the strict consensus supertree.
We can take into account confidence values associated to the clades of the trees in $\mathcal{T}$ (ex, bootstrap proportions).

MRP : properties et problems

We can obtain an Bad exponential number of most parsimonious trees.
$\Longrightarrow$ In this case, MRP returns the strict consensus of the most parsimonious trees (to avoid to take arbitrary decisions).

Problems

Bad The MRP supertree may contain Bad new clades, not present in any of source trees.
Bad Bias Bigger clades/ less equilibrated trees.

SUPERTRIPLETS

This method [Ranwez et al. 2009] aims at finding the asymmetric median supertree according to triplet dissimilarity. It consists of four steps:

1. input trees are encoded as a set of weighted triplets;
2. a starting binary supertree is proposed by an agglomerative procedure;
3. the candidate binary supertree is iterativly improved using small topological changes;
4. unsupported edges of the binary supertree are collapsed.

Other VOTE methods

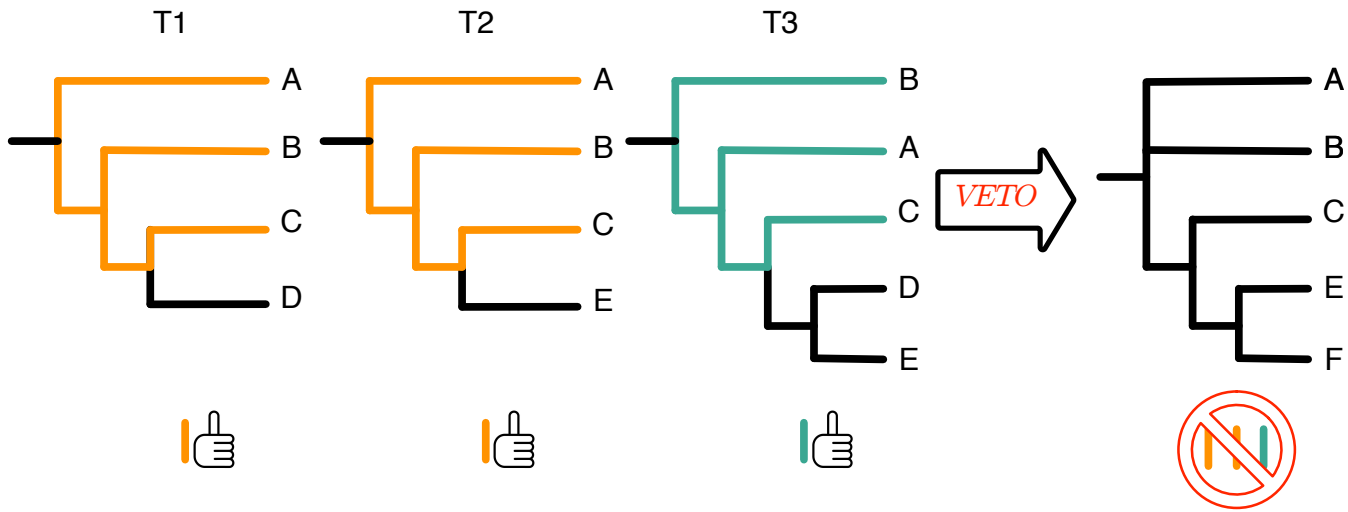
- The Matrix Representation with Flipping (MRF) supertree
- The Matrix Representation using Compability (MRC) supertree
- The t-MRP method
- The average consensus supertree or MRD
- Median supertrees
- Maximum Likelihood supertree
- Bayesian supertree
- Quartet supertrees
- ...

Supertree methods – VETO methods

3. Veto methods do not allow the resulting supertree to contain clades that a source tree would vote against.

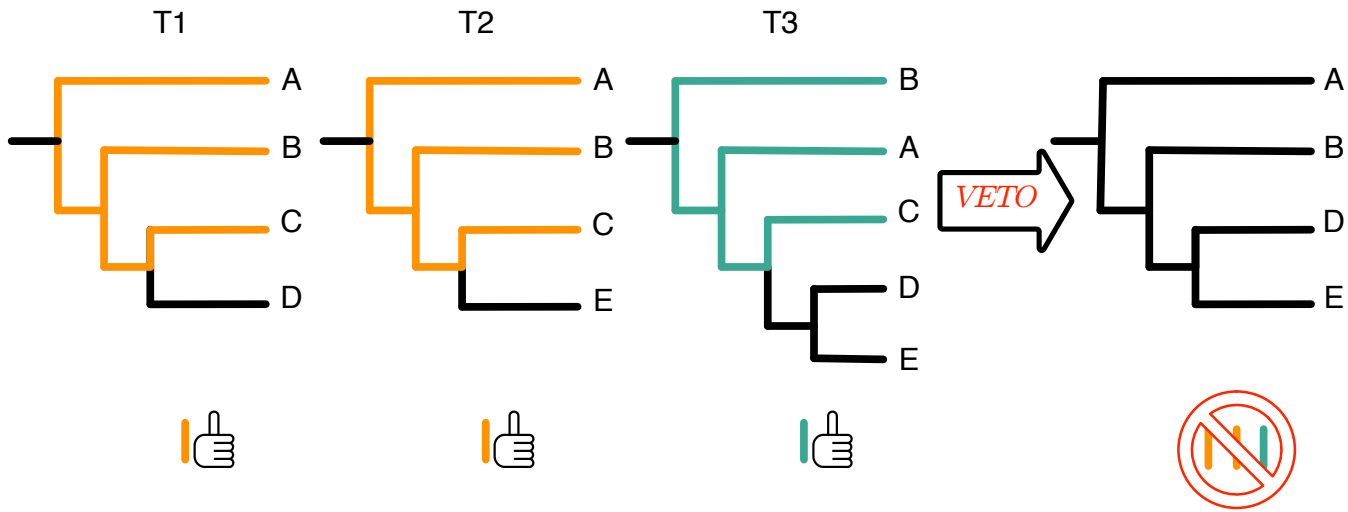
Supertree methods – VETO methods

3. Veto methods do not allow the resulting supertree to contain clades that a source tree would vote against.
   - proposing multifurcations



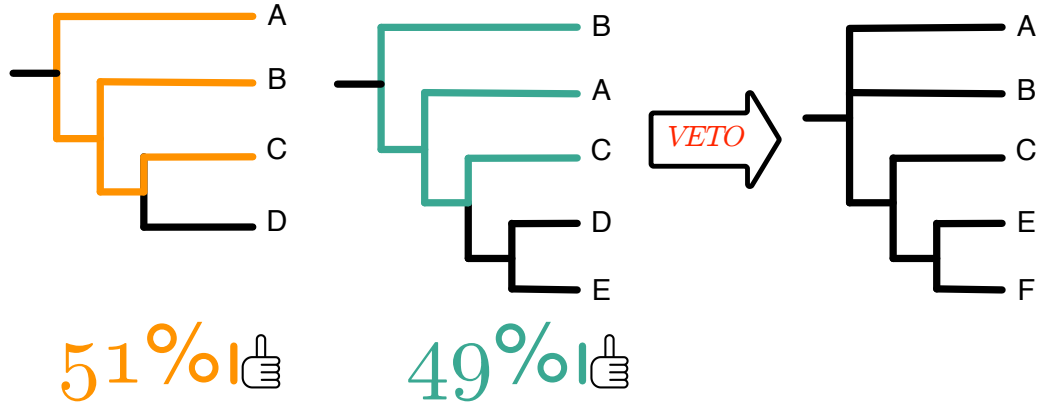
Supertree methods – VETO methods

3. Veto methods do not allow the resulting supertree to contain clades that a source tree would vote against.
   - proposing multifurcations
     OR
   - pruning some taxa



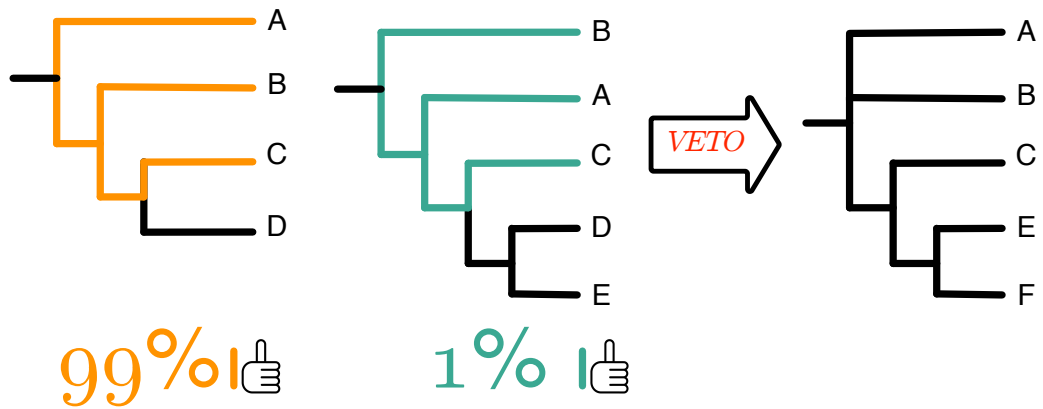
Supertree methods – VETO methods

3. Veto methods do not allow the resulting supertree to contain clades that a source tree would vote against.
   - proposing multifurcations
     OR
   - pruning some taxa
   - pros: more reliable supertrees



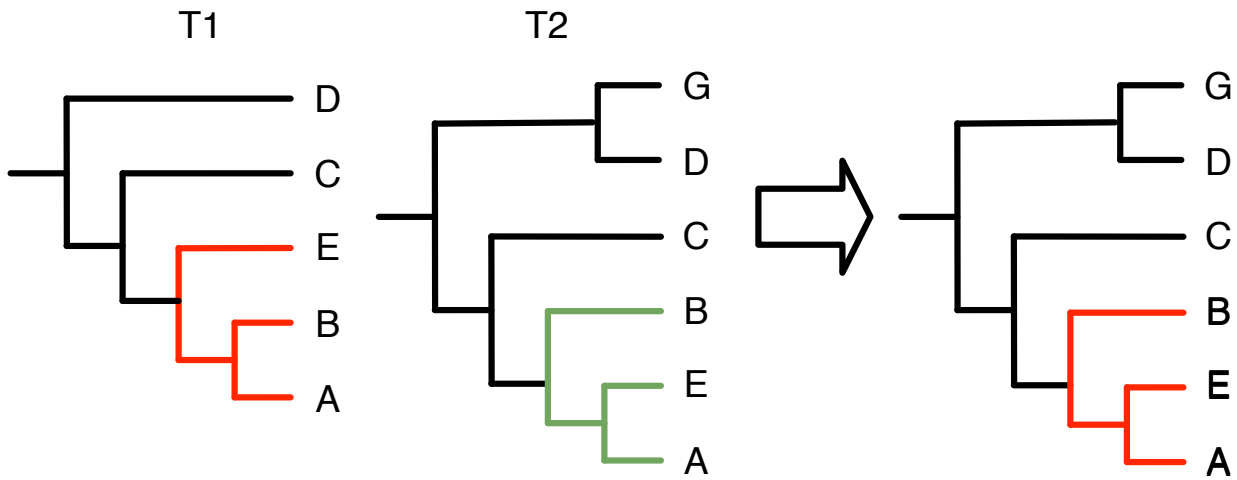
Supertree methods – VETO methods

3. Veto methods do not allow the resulting supertree to contain clades that a source tree would vote against.
   - proposing multifurcations
     OR
   - pruning some taxa
   - pros: more reliable supertrees
   - cons: unresolved supertrees.



PhySIC_IST – A VETO method with desirable proprieties

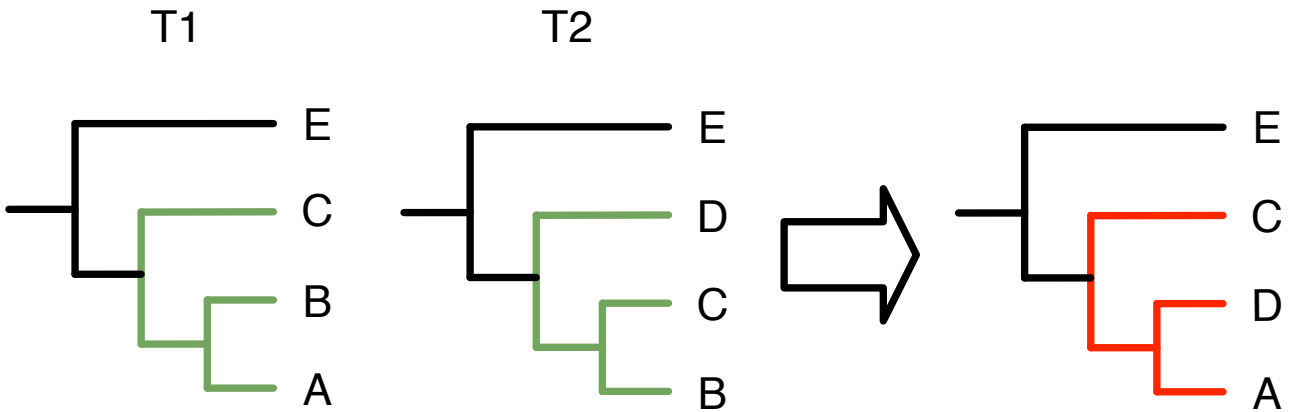
- The resulting supertree does not contain relationships that conflict either directly with a source tree or indirectly with a combination of them.



non-contradiction property, denoted by PC

PhySIC_IST – A VETO method with desirable proprieties

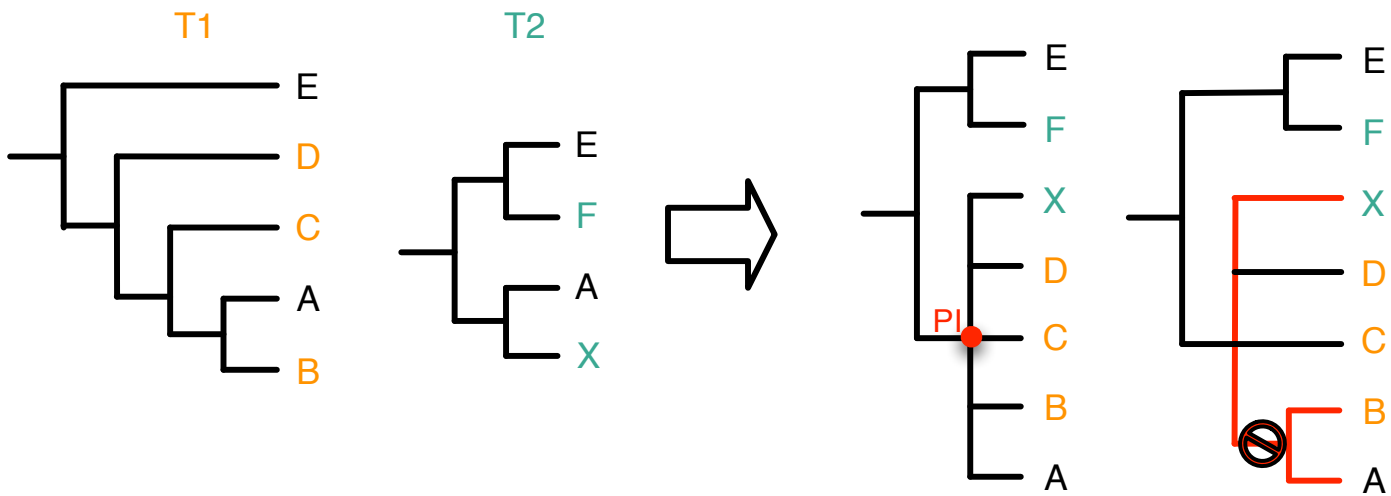
- The resulting supertree does not contain relationships that conflict either directly with a source tree or indirectly with a combination of them.



non-contradiction property, denoted by PC

PhySIC_IST – A VETO method with desirable proprieties

- The resulting supertree only contains relationships that are present in a source tree or collectively induced by several source trees.



induction property, denoted by PI

PhySIC_IST – A VETO method with desirable proprieties

- PhySIC_IST is a method that:

PhySIC_IST – A VETO method with desirable proprieties

- PhySIC_IST is a method that:
  - maximizes the information contained in the produced supertree;

PhySIC_IST – A VETO method with desirable proprieties

- PhySIC_IST is a method that:
  - maximizes the information contained in the produced supertree;
  - returns a supertree T that still respects PC and PI by:

PhySIC_IST – A VETO method with desirable proprieties

- PhySIC_IST is a method that:
  - maximizes the information contained in the produced supertree;
  - returns a supertree T that still respects PC and PI by:
    - allowing multifurcations;

PhySIC_IST – A VETO method with desirable proprieties

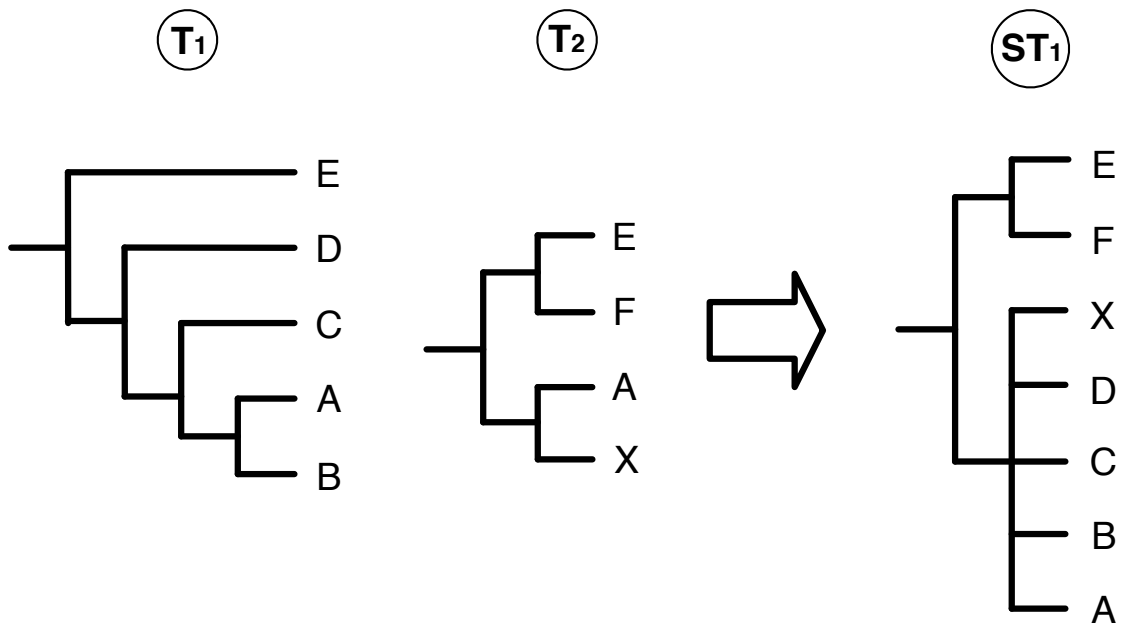
- PhySIC_IST is a method that:
  - maximizes the information contained in the produced supertree;
  - returns a supertree T that still respects PC and PI by:
    - allowing multifurcations;

      AND
    - pruning rogue taxa:

PhySIC_IST – A VETO method with desirable proprieties

- PhySIC_IST is a method that:
  - maximizes the information contained in the produced supertree;
  - returns a supertree T that still respects PC and PI by:
    - allowing multifurcations;

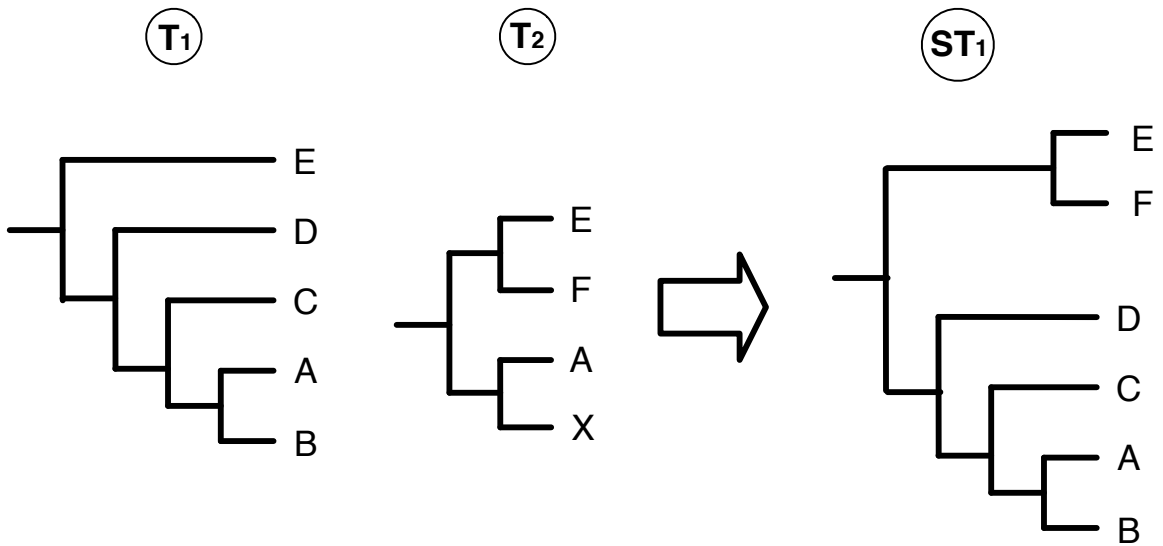
      AND
    - pruning rogue taxa:



PhySIC_IST – A VETO method with desirable proprieties

- PhySIC_IST is a method that:
  - maximizes the information contained in the produced supertree;
  - returns a supertree T that still respects PC and PI by:
    - allowing multifurcations;

      AND
    - pruning rogue taxa:

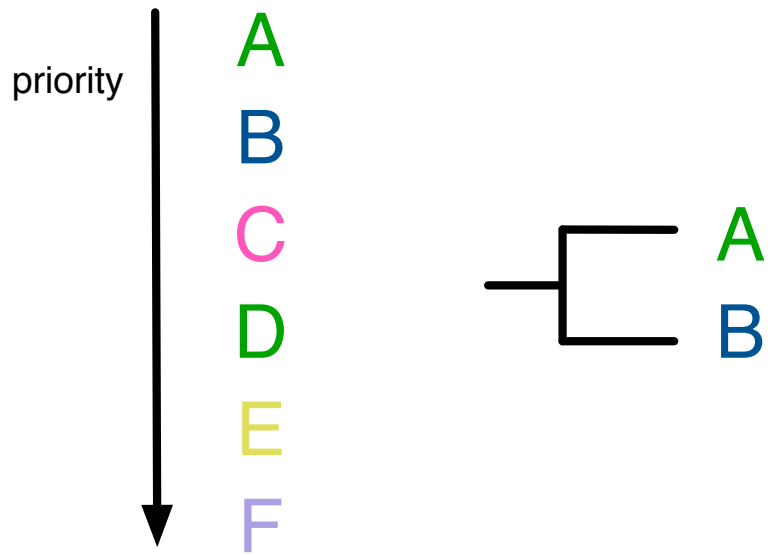


PhySIC_IST

- PhySIC_IST (*PHYlogenetic Signal with Induction and non-Contradiction Inserting a Subset of Taxa*) is an algorithm that operates successive insertions of taxa on a backbone tree.

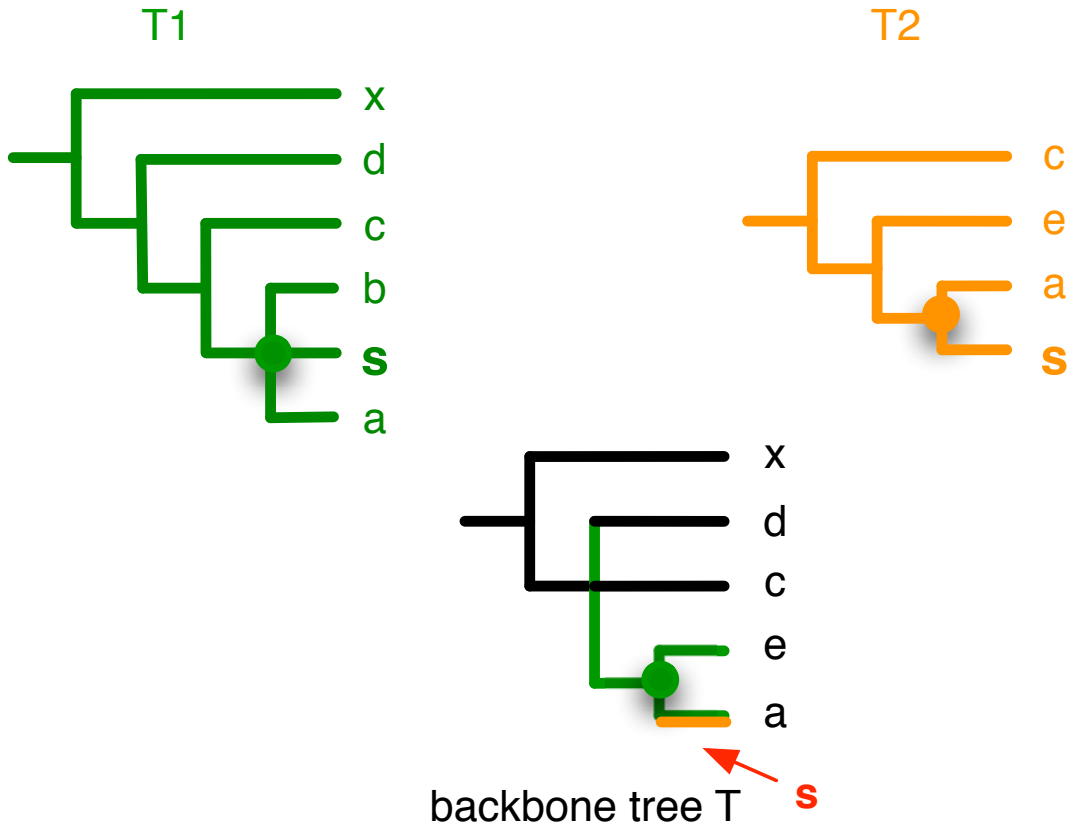
PhySIC_IST

- We order taxa in decreasing priority order
- The first taxa to be inserted are those present in as much source trees as possible and involved in as few contradictions as possible
- We build the backbone tree



PhySIC_IST

- Within which region of the backbone tree can a taxon s be inserted without contradicting T_1 and T_2?



PhySIC_IST

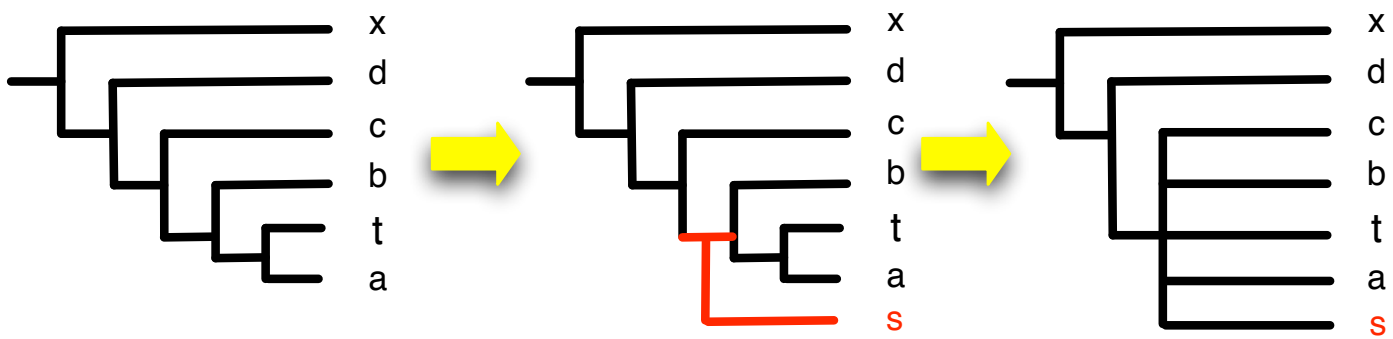
- one best supported position (PI) and all trees agree (PC)

PhySIC_IST

- one best supported position (PI) and all trees agree (PC)
- more than one best supported position and/not all trees agree (PI and PC???)

PhySIC_IST

- one best supported position (PI) and all trees agree (PC)
- more than one best supported position and/not all trees agree (PI and PC???)



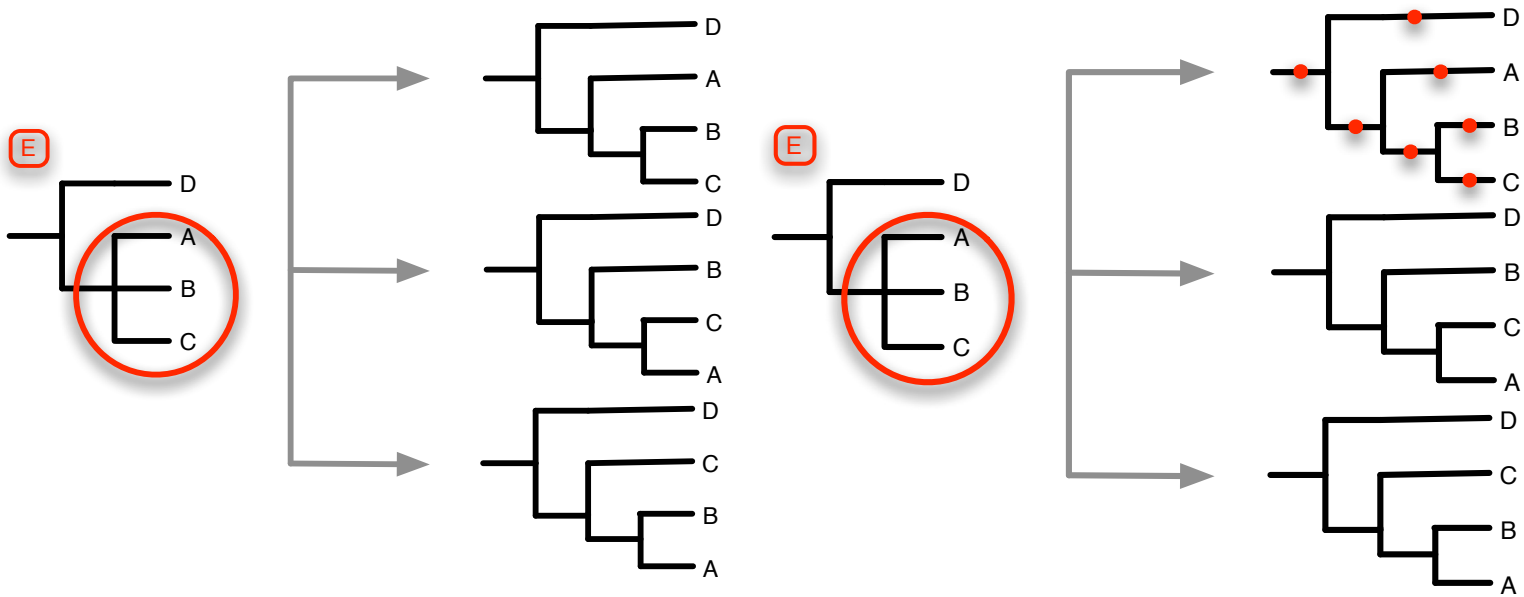
PhySIC_IST

- one best supported position (PI) and all trees agree (PC)
- more than one best supported position and/not all trees agree (PI and PC???)

PhySIC_IST

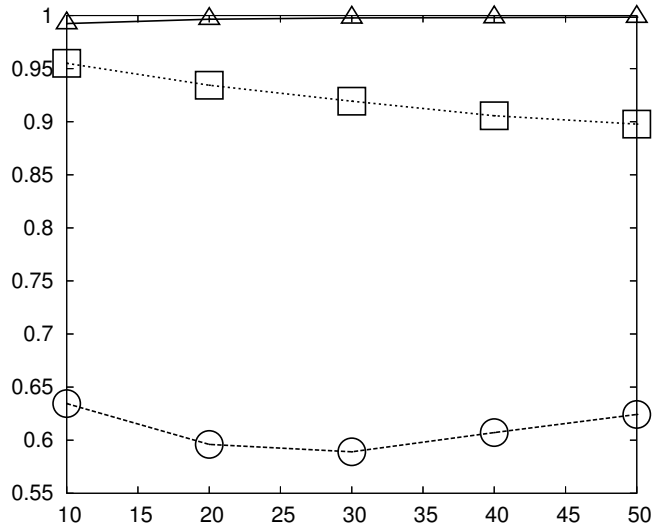
- We need to compare trees with different taxa sets.
- We use a variant of the CIC criterion (Thorley, Wilkinson, Charleston 1998) that also takes into account missing taxa and we define it as:

$$CIC(T, n) = -\lg \frac{\text{number of permitted binary trees with } n \text{ taxa}}{\text{number of possible binary trees with } n \text{ taxa}}$$

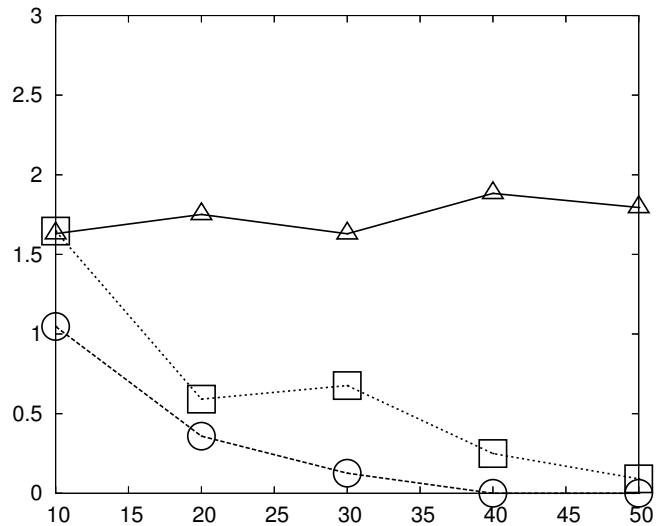


Limits of veto methods

- As the amount of available information continues to increase, the number of conflicts between source trees increases
  MRP △, PhySIC ○, PhySIC_IST □

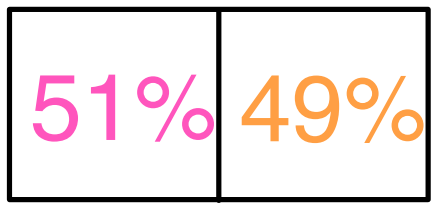


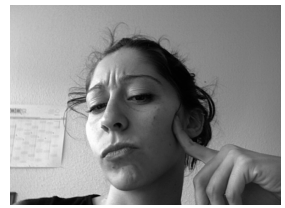
informativeness

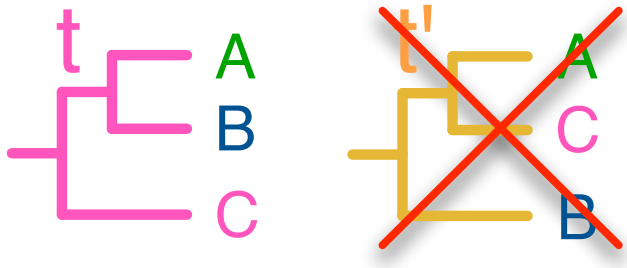


inaccuracy

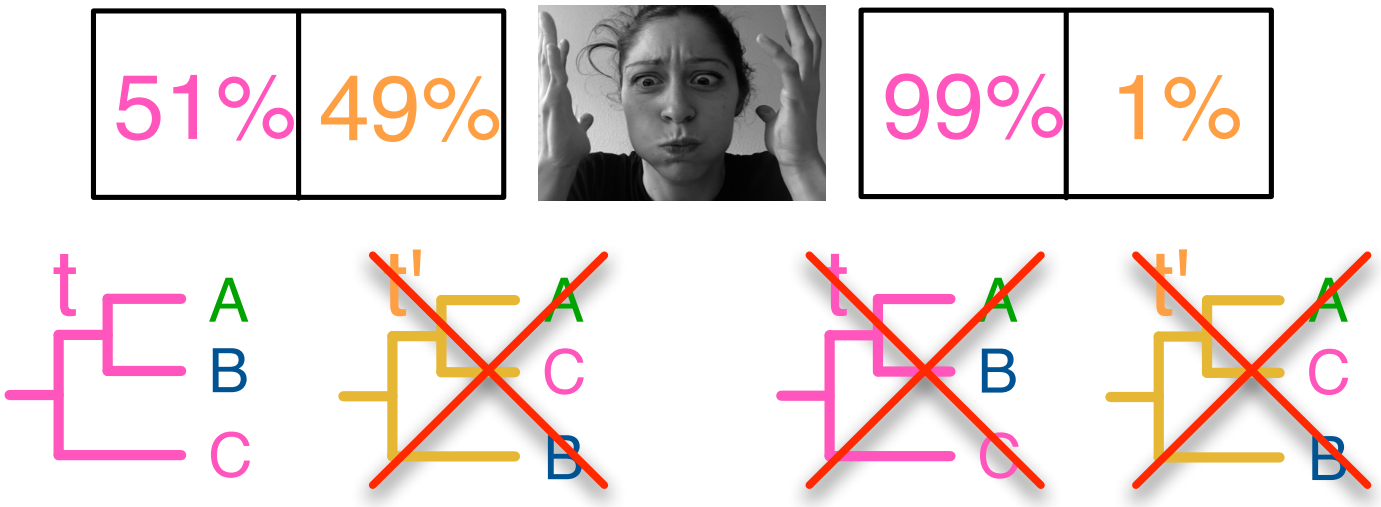
Vote VS veto methods?



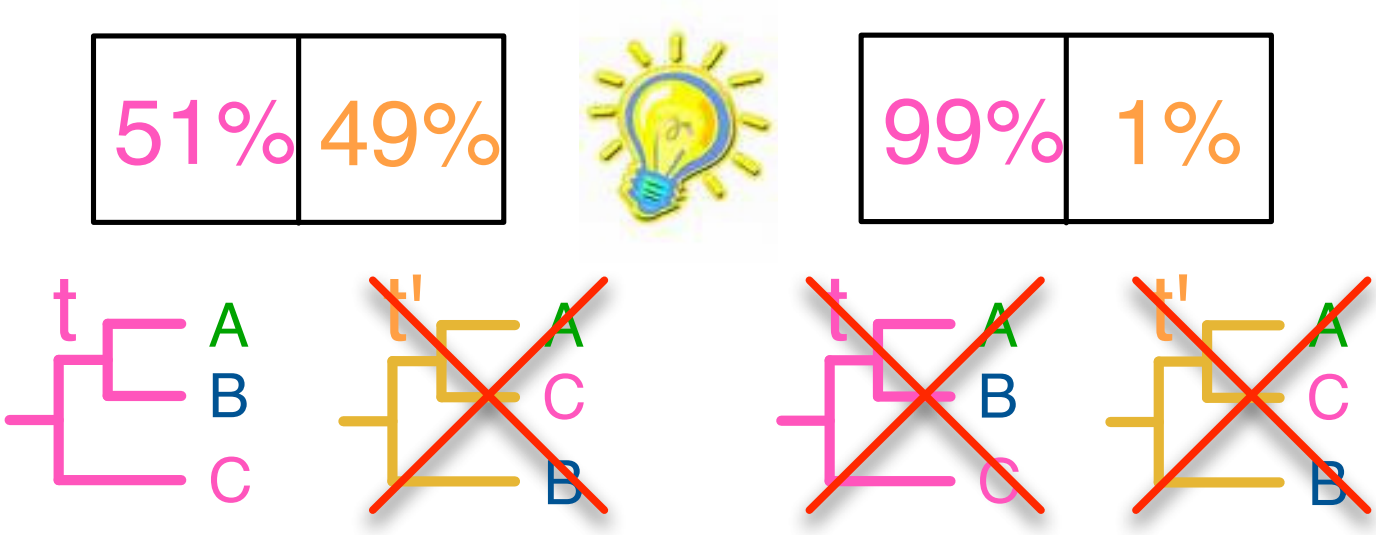




Vote VS veto methods?



Vote VS veto methods?

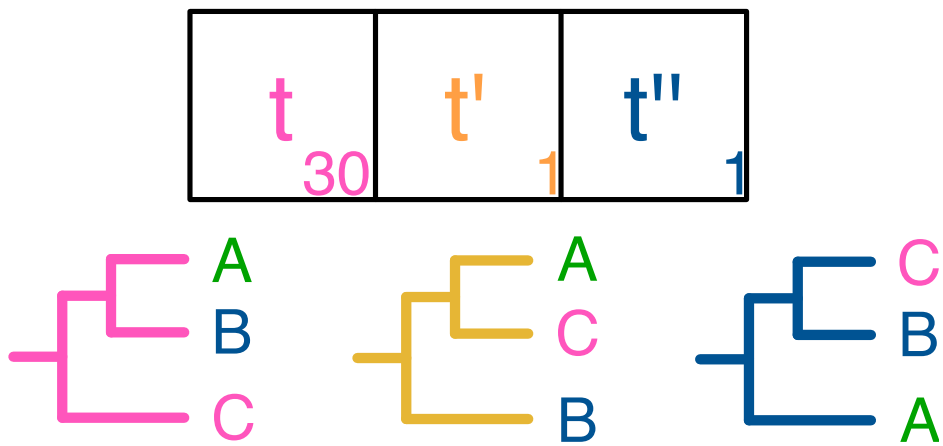


- IDEA: a flexible liberal(voting) preprocessing of the input trees before applying a veto approach.

Source Tree Correction (STC) preprocess

We want to drop the statistically less supported alterative(s), if any exists.

(Chi-square based test)



STC preprocess

- After that, the STC preprocess modifies the source trees, forcing them not to contain the dropped resolutions.

STC preprocess

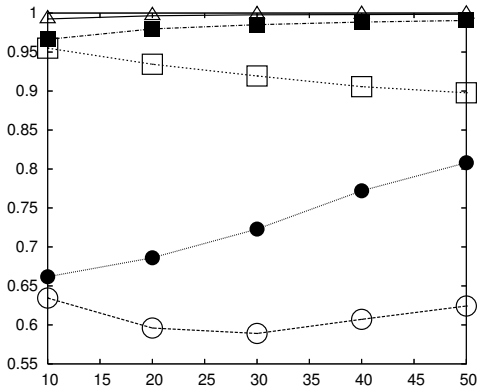
- After that, the STC preprocess modifies the source trees, forcing them not to contain the dropped resolutions.
- Each modified tree may contain either new multifurcations, or lack some of its former taxa.

STC preprocess

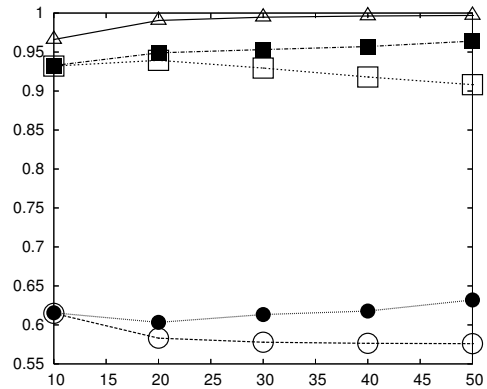
- After that, the STC preprocess modifies the source trees, forcing them not to contain the dropped resolutions.
- Each modified tree may contain either new multifurcations, or lack some of its former taxa.
- The reliability threshold α is chosen by the user.

Large-scale simulations ($\alpha = 0.95$)

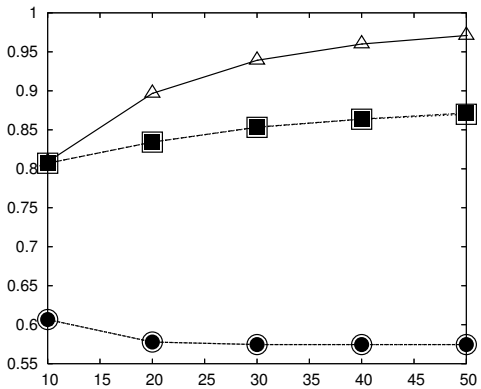
MRP △, PhySIC ○, PhySIC_IST □, STC + PhySIC ● and STC + PhySIC_IST ■



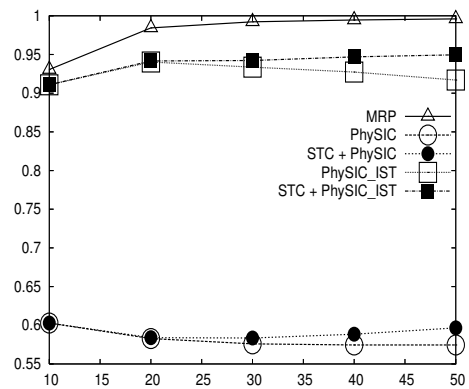
$d = 25\%$



$d = 50\%$



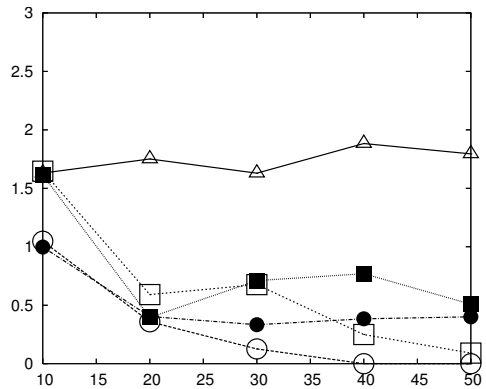
$d = 75\%$



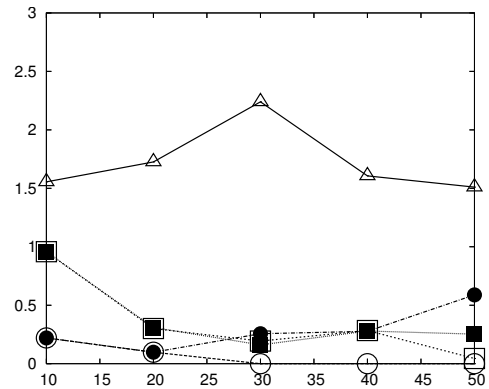
mixed d

Large-scale simulations ($\alpha = 0.95$)

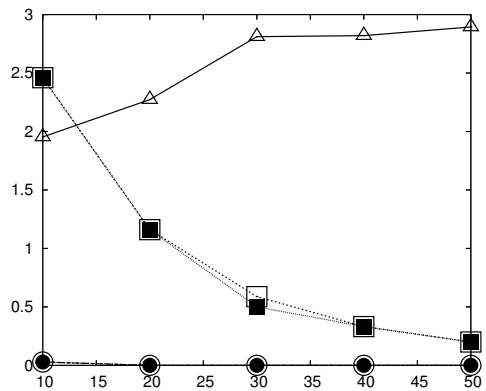
MRP △, PhySIC ○, PhySIC_IST □, STC + PhySIC ● and STC + PhySIC_IST ■



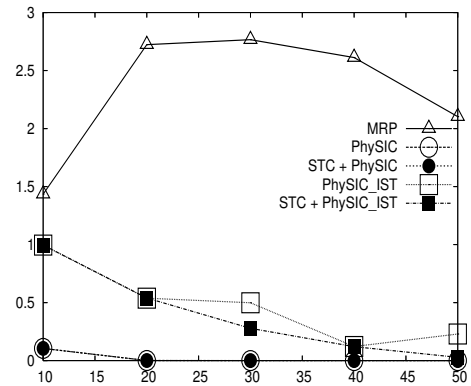
$d = 25\%$



$d = 50\%$



$d = 75\%$



mixed d

Other VETO methods

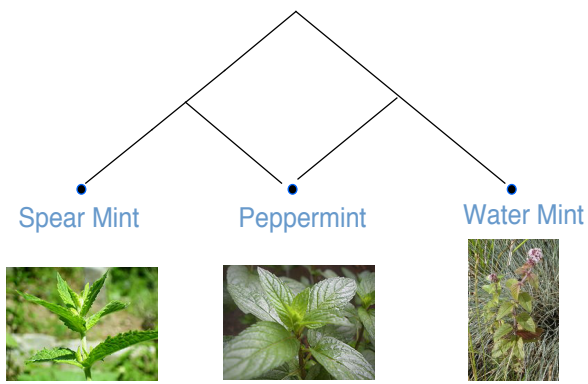
- The semi-strict supertree
- PhySIC supertree
- The SMAST supertrees
- ...

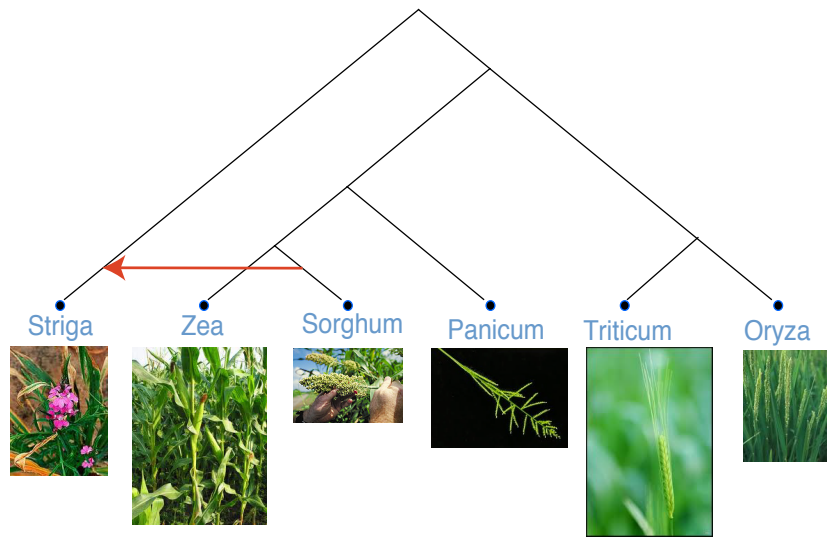
THANKS!

Merci à V Berry et S Berard pour leur transparents, source d'inspiration de certains de miens :)

A *new* approach: building phylogenic networks

Why do we need them? Due to reticulate evolutionary phenomena (hybridization, recombination, horizontal gene transfer) the evolution of a set of species sometimes cannot be described using a single phylogenetic tree.





Networks...

IF TIME...!