Tübingen - 24/09/2008

Decomposition and reconstruction of level-k phylogenetic networks from triplets

Philippe Gambette

UM2 UNIVERSITÉ MONTPELLIER 2 SCIENCES ET TECHNIQUES

LIRMM

CNRS

Outline

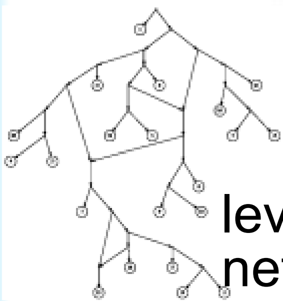
- Phylogenetic networks
- Decomposition of level-k networks
- Reconstruction of networks from triplets
- The obstruction approach to reconstruction
- Triplet identifiability of galled trees

Phylogenetic networks

Phylogenetic network

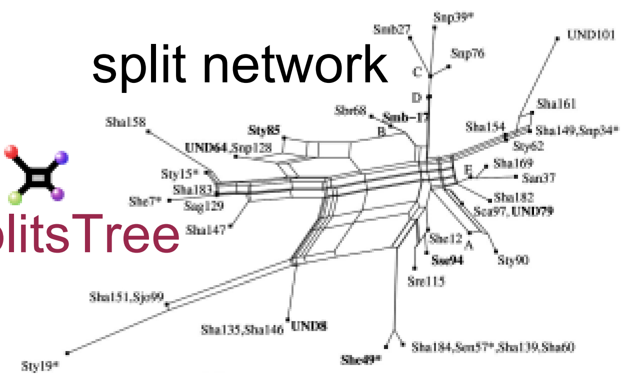
From Wikipedia, the free encyclopedia

A **phylogenetic network** is *any* graph used to visualize evolutionary relationships between species or organisms. It is employed when reticulate events such as hybridization, horizontal gene transfer, recombination, or gene duplication and loss are believed to be involved. Phylogenetic trees are a subset of phylogenetic networks.



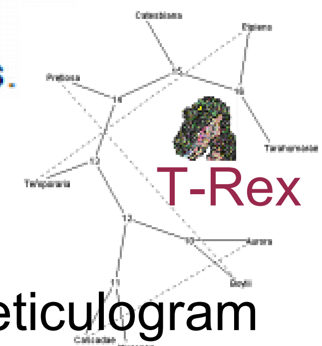
level-2 network

Level-2



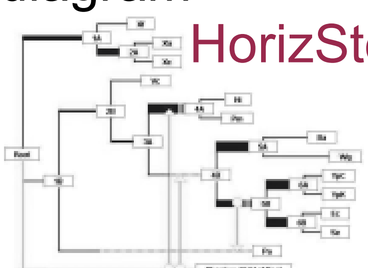
split network

SplitsTree



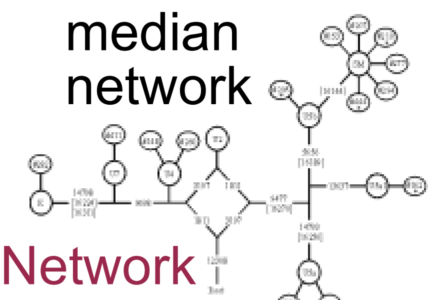
T-Rex

reticulogram



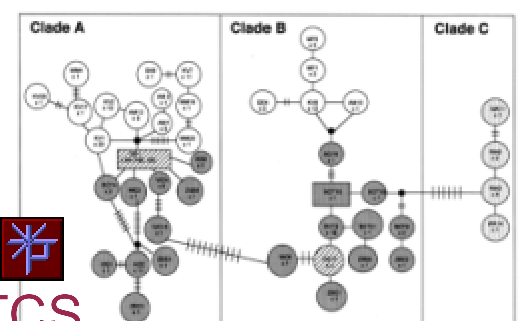
synthesis diagram

HorizStory



median network

Network



minimum spanning network

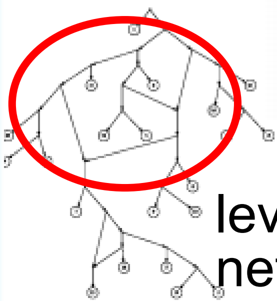
TCS

Phylogenetic networks

Phylogenetic network

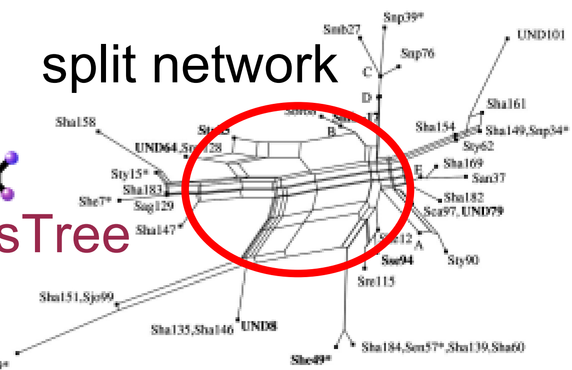
From Wikipedia, the free encyclopedia

A **phylogenetic network** is *any* graph used to visualize evolutionary relationships between species or organisms. It is employed when reticulate events such as hybridization, horizontal gene transfer, recombination, or gene duplication and loss are believed to be involved. Phylogenetic trees are a subset of phylogenetic networks.



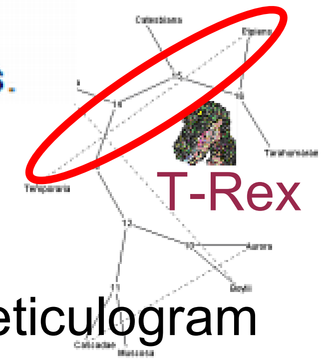
level-2 network

Level-2



split network

SplitsTree

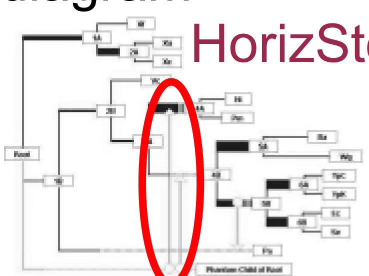


T-Rex

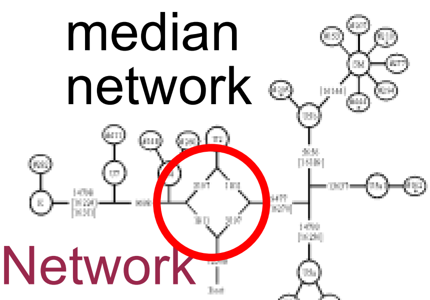
reticulogram

synthesis diagram

HorizStory

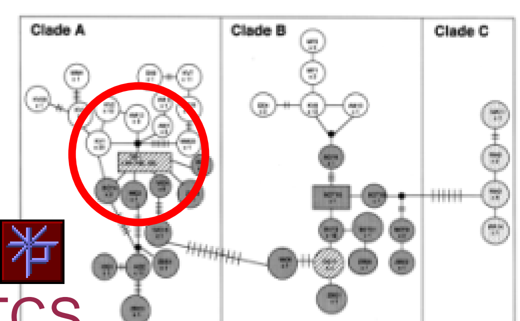


median network



Network

minimum spanning network



TCS

Abstract or explicit networks

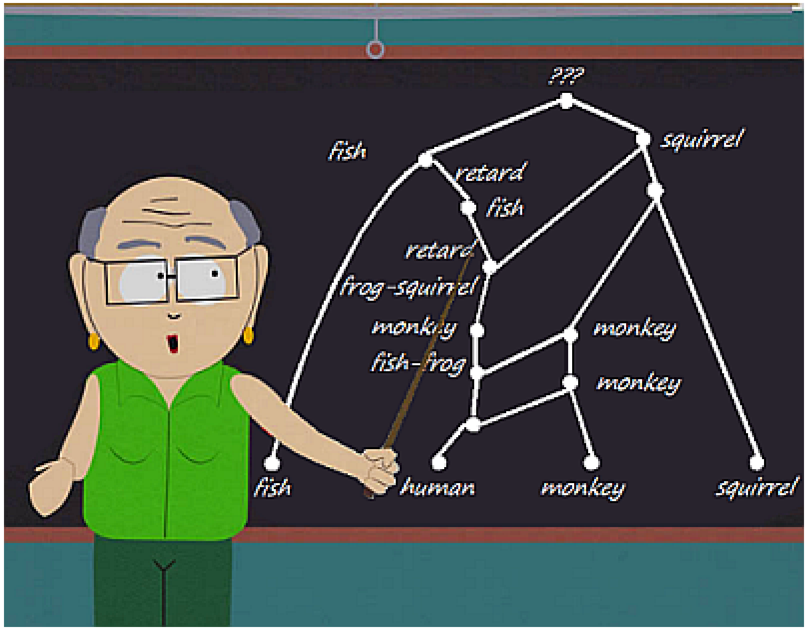
An **explicit phylogenetic network** is a phyogenetic network where all reticulations can be interpreted as precise biological events.

An **abstract network** reflects some phylogenetic signals rather than explicitly displaying biological reticulation events.

Abstract or explicit networks

An **explicit phylogenetic network** is a phyogenetic network where all reticulations can be interpreted as precise biological events.

An **abstract network** reflects some phylogenetic signals rather than explicitly displaying biological reticulation events.

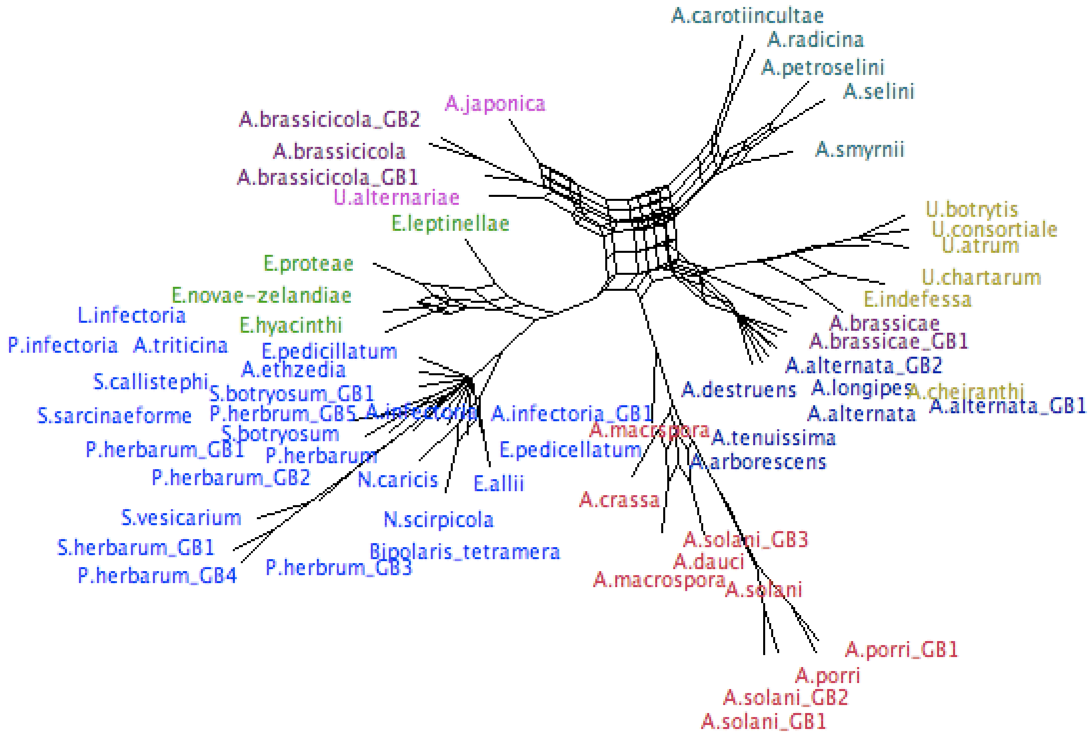


Explicit phylogenetic network representing the Theory of Evolution as described by Mrs Garrison in South Park S10E12.

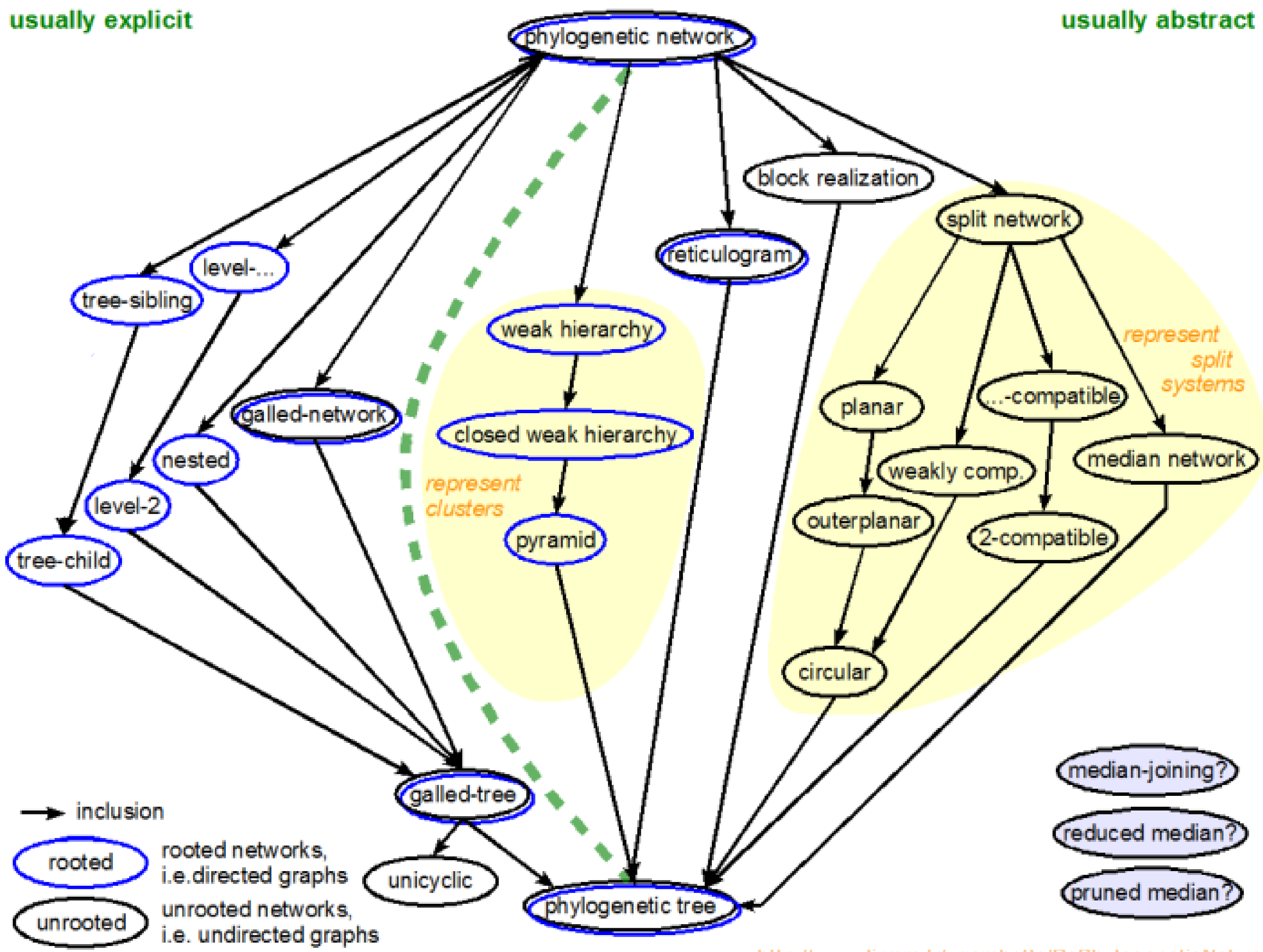
Abstract or explicit networks

An **explicit phylogenetic network** is a phyogenetic network where all reticulations can be interpreted as precise biological events.

An **abstract network** reflects some phylogenetic signals rather than explicitly displaying biological reticulation events.



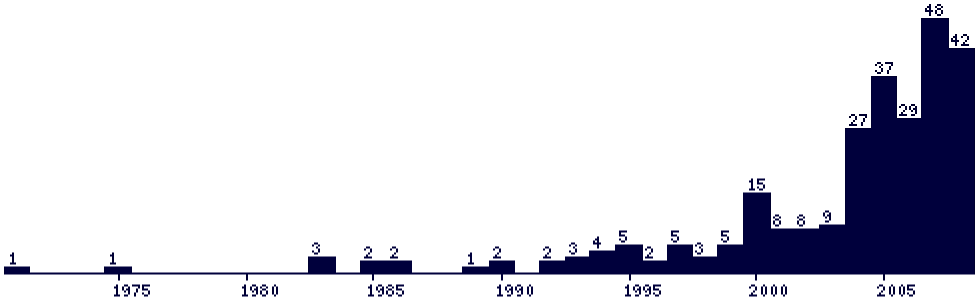
Hierarchy of network subclasses



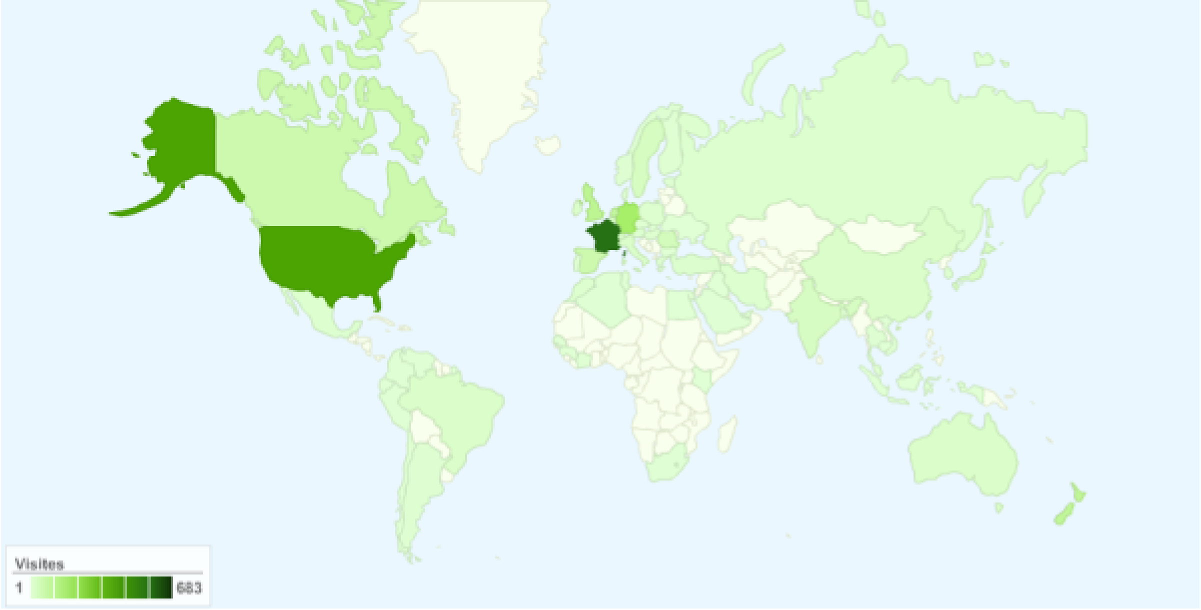
http://www.lirmm.fr/~gambette/RePhylogeneticNetworks.php

Phylogenetic networks: a current topic

Publications about phylogenetic networks:



Visits on the *Who's who in Phylogenetic Networks*:



France,
USA,
Germany,
United Kingdom,
New Zealand,
Netherlands,
Spain,
Canada,
Romania,
Israel...

Phylogenetic networks: a clustered topic

Tag cloud of authors on phylogenetic networks:

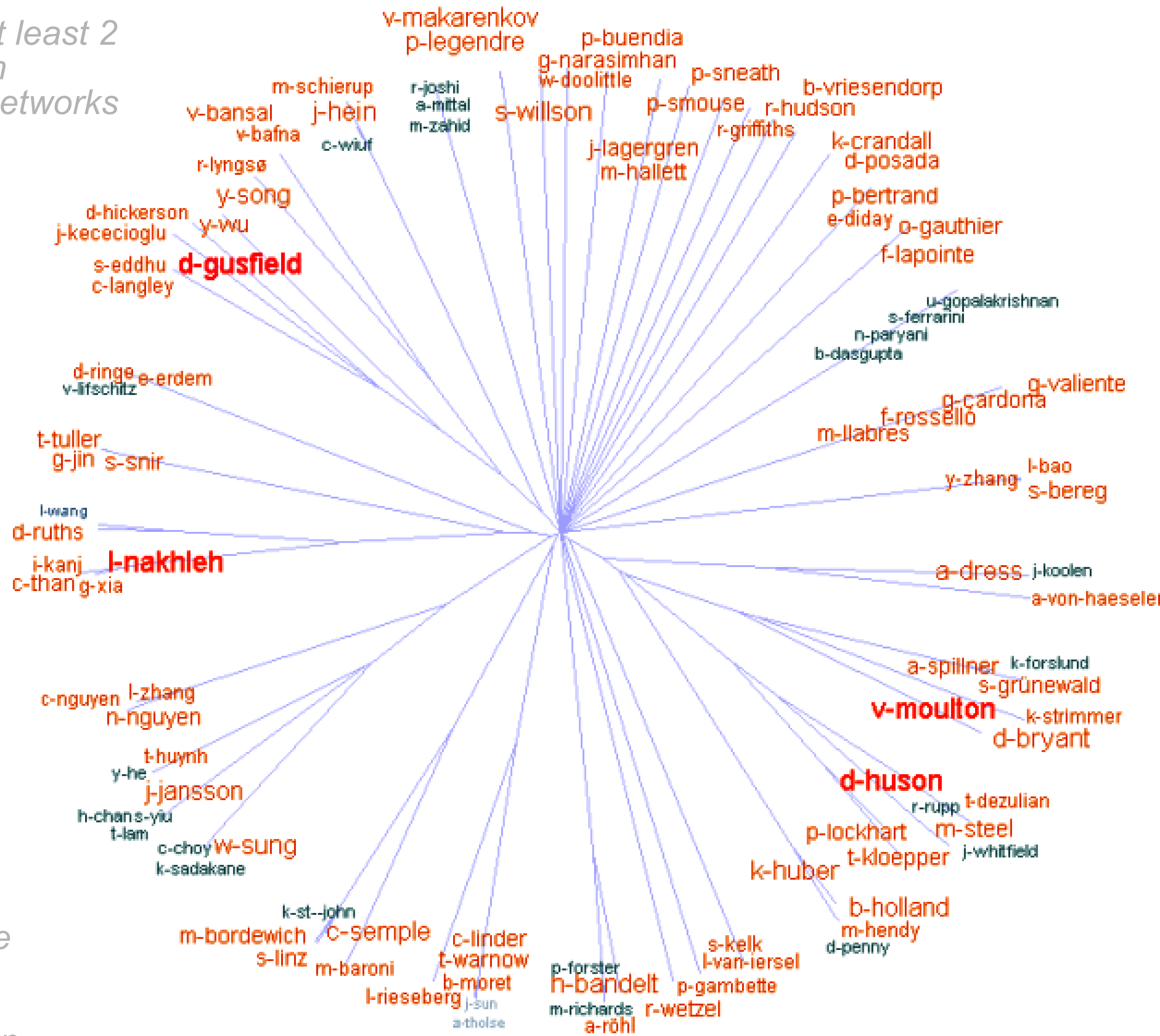
a-boc **a-dress** a-gupta a-lesser a-mittal a-padolina a-rambaut a-röhl a-spillner a-templeton a-tholse a-tofigh a-von-haeseler b-dasgupta b-gemeinholzer b-holland b-leclerc b-moret b-oxelman b-sykes b-vriesendorp c-choy c-cotta c-langley c-linder c-nguyen c-ouzounis c-paul c-rausch **c-semple** c-sing c-than c-wiuf d-brooks **d-bryant** **d-gusfield** d-havell d-hickerson **d-huson** d-kevorkov d-levy d-macleod d-mallick d-morrison d-penny d-posada d-rao d-richter d-ringe d-ruths e-bapteste e-diday e-dubrova e-erdem e-holmes e-watson f-bakker f-delsuc f-hagen f-lapointe f-rohlf f-rosselló f-simancík f-zhao g-bourque g-cardona g-churchill g-conner g-della-vedova g-jin g-kumar g-mauri g-narasimhan g-nelson g-valiente g-xia **h-bandelt** h-chan h-innan h-spencer h-wanntorp i-althöfer i-cassens i-kanj j-aude j-byrka j-codani j-davies j-dopazo j-gogarten **j-hein** **j-jansson** j-kececioglu j-keijsper j-koolen j-lagergren j-manuch j-mellor-crummey j-minett j-morefield j-risler j-sun j-trujillo j-whitfield k-bean k-crandall k-forslund **k-huber** k-kryukov k-mcbreen k-sadakane k-st--john k-strimmer k-zhang l-addario-berry l-bao l-excoffier l-goldovsky l-jermiin **l-nakhleh** l-pachter l-rieseberg l-stacho l-stougie l-van-iersel l-wang l-wang l-zhang m-baroni m-bordewich m-clement m-franz m-hallett m-hendy m-kennedy m-langton m-llabrés m-lott m-milinkovitch m-mnich m-morin m-paul m-poptsova m-richards m-schierup m-steel m-zahid n-darzentas n-grassly n-hamilton n-kaplan n-nguyen n-paryani n-saitou n-vyahhi o-gauthier p-bertrand p-bonizzoni p-buendia p-evans p-forster p-gambette p-gawrychowski p-jana **p-legendre** p-lockhart p-mardulyn p-smouse p-sneath q-wu r-beiko r-charlebois r-dondi r-gray r-griffiths r-hudson r-joshi r-lyngsø r-rupp r-timme r-uehara r-wetzel r-winkworth s-benthin s-bereg s-cameron s-cornelsen s-eddhu s-erdogan s-ferrarini s-grünewald s-kelk s-linz s-moran s-myers s-snir **s-willson** s-woolley s-xu s-yiu t-asano t-boekhout t-dezulian t-huynh t-kloepper t-lam t-mailund t-nguyen t-tuller t-warnow u-brandes u-gopalakrishnan v-bafna v-bansal v-berry v-grant v-kunin v-kusherbaeva v-lifschitz v-macaulay **v-makarenkov** **v-moulton** w-doolittle w-fitch w-maddison **w-sung** x-zhao y-desdevises y-diaz-lazcoz y-he **y-song** y-wu y-zhang z-ding

The size represents the number of publications about phylogenetic networks, weighted by the number of coauthors on each publication.

Phylogenetic networks: a clustered topic

Tree cloud of the main authors on phylogenetic networks:

Authors with at least 2 publications on phylogenetic networks



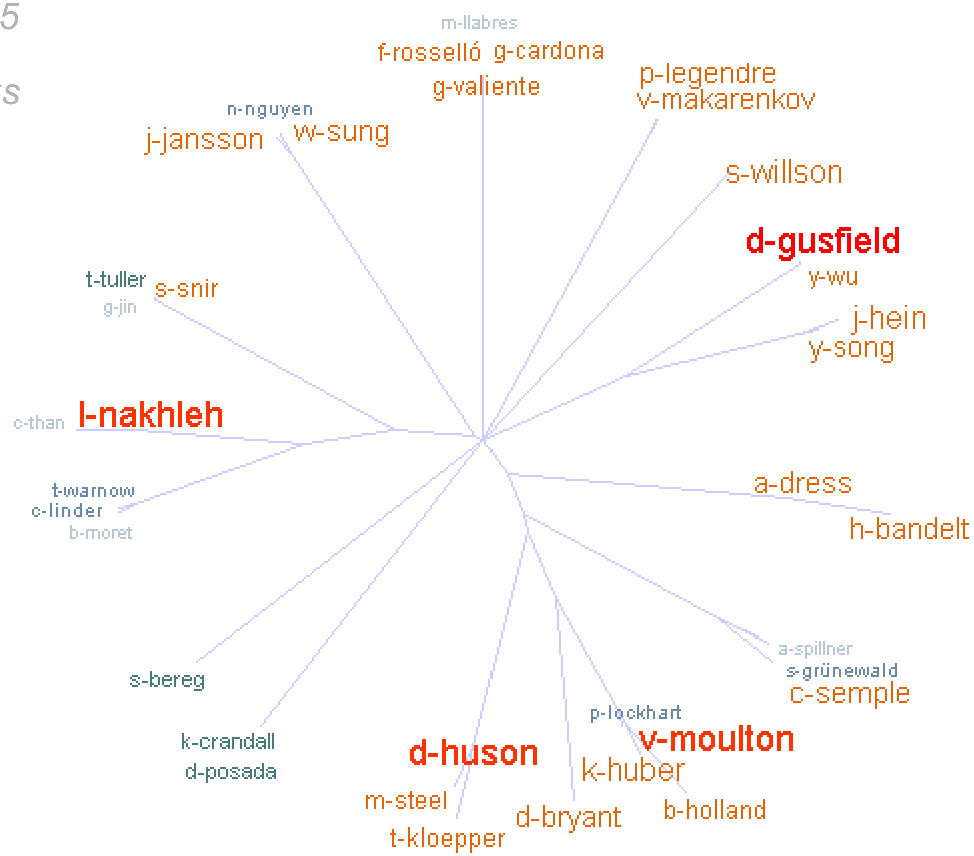
The size represents the number of publications about phylogenetic networks, weighted by the number of coauthors on each publication.

made with Tree Cloud and SplitsTree - http://www.lirmm.fr/~gambette/ProgTreeCloud.php

Phylogenetic networks: a clustered topic

Tree cloud of the main authors on phylogenetic networks:

Authors with at least 5 publications on phylogenetic networks



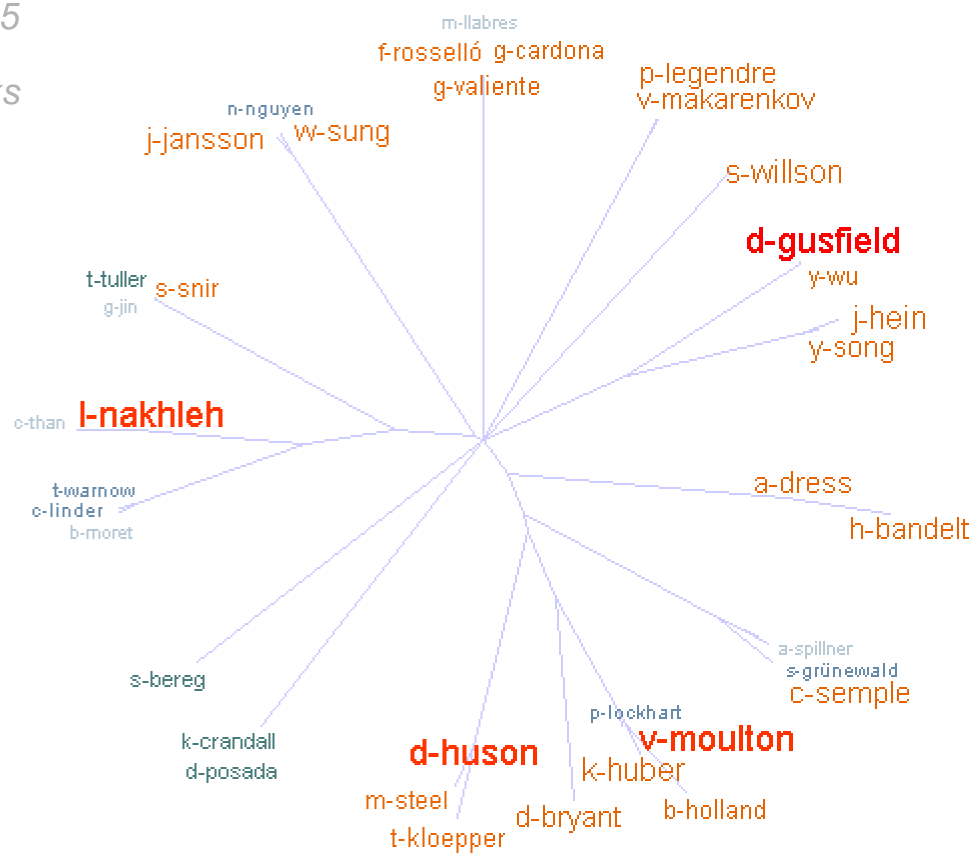
The size represents the number of publications about phylogenetic networks, weighted by the number of coauthors on each publication.

made with Tree Cloud and SplitsTree - http://www.lirmm.fr/~gambette/ProgTreeCloud.php

Phylogenetic networks: a clustered topic

Tree cloud of the main authors on phylogenetic networks:

Authors with at least 5 publications on phylogenetic networks



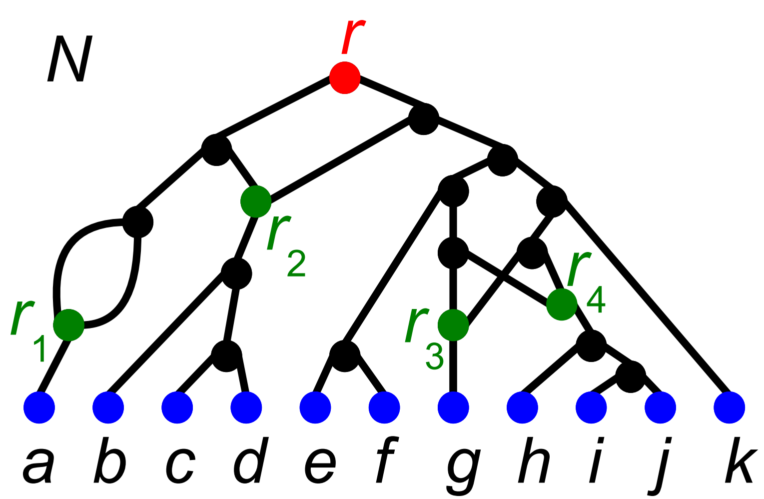
How to build **robust** tree clouds?
- which **cooccurrence distance**?
- which **tree reconstruction method**?

Ongoing work with Jean Véronis (Aix-en-Provence, Computational Linguistics) and Delphine Amstutz (Paris, XVIIth Century Literature Analysis)

Level-*k* phylogenetic networks

An **level-*k* phylogenetic network *N*** on a set *X* of *n* taxa is a multidigraph in which:

- exactly one vertex has indegree 0 and outdegree 2: the **root**,
- all other vertices have either:
    - indegree 1 and outdegree 2: **split vertices**,
    - indegree 2 and outdegree ≤ 1: **reticulation vertices**,
    - or indegree 1 and outdegree 0: **leaves** labeled by *X*,
- any **blob** has at most *k* reticulation vertices.

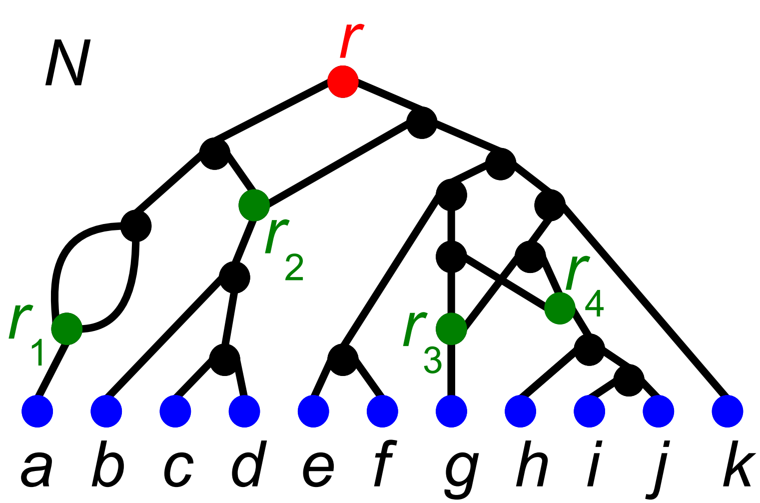


All arcs are oriented downwards

In collaboration with Vincent Berry (Montpellier, Bioinformatics) and Christophe Paul (Montpellier, Graph Theory)

Level-*k* phylogenetic networks

An **level-*k* phylogenetic network *N*** on a set *X* of *n* taxa is a multidigraph in which:
- exactly one vertex has indegree 0 and outdegree 2: the **root**,
- all other vertices have either:
    - indegree 1 and outdegree 2: **split vertices**,
    - indegree 2 and outdegree ≤ 1: **reticulation vertices**,
    - or indegree 1 and outdegree 0: **leaves** labeled by *X*,
- any **blob** has at most *k* reticulation vertices.

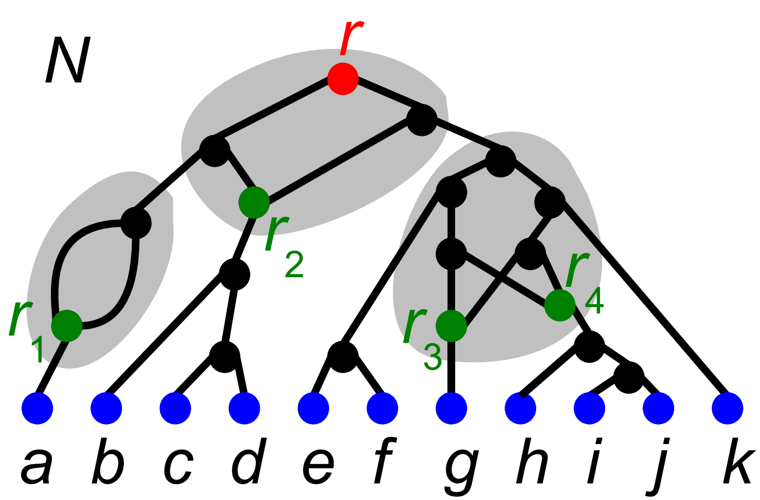


A **blob** is a maximal **biconnected** component of the underlying undirected graph, that is a maximal subgraph which remains connected after the removal of one of its vertices.

Level-*k* phylogenetic networks

An **level-*k* phylogenetic network *N*** on a set *X* of *n* taxa is a multidigraph in which:

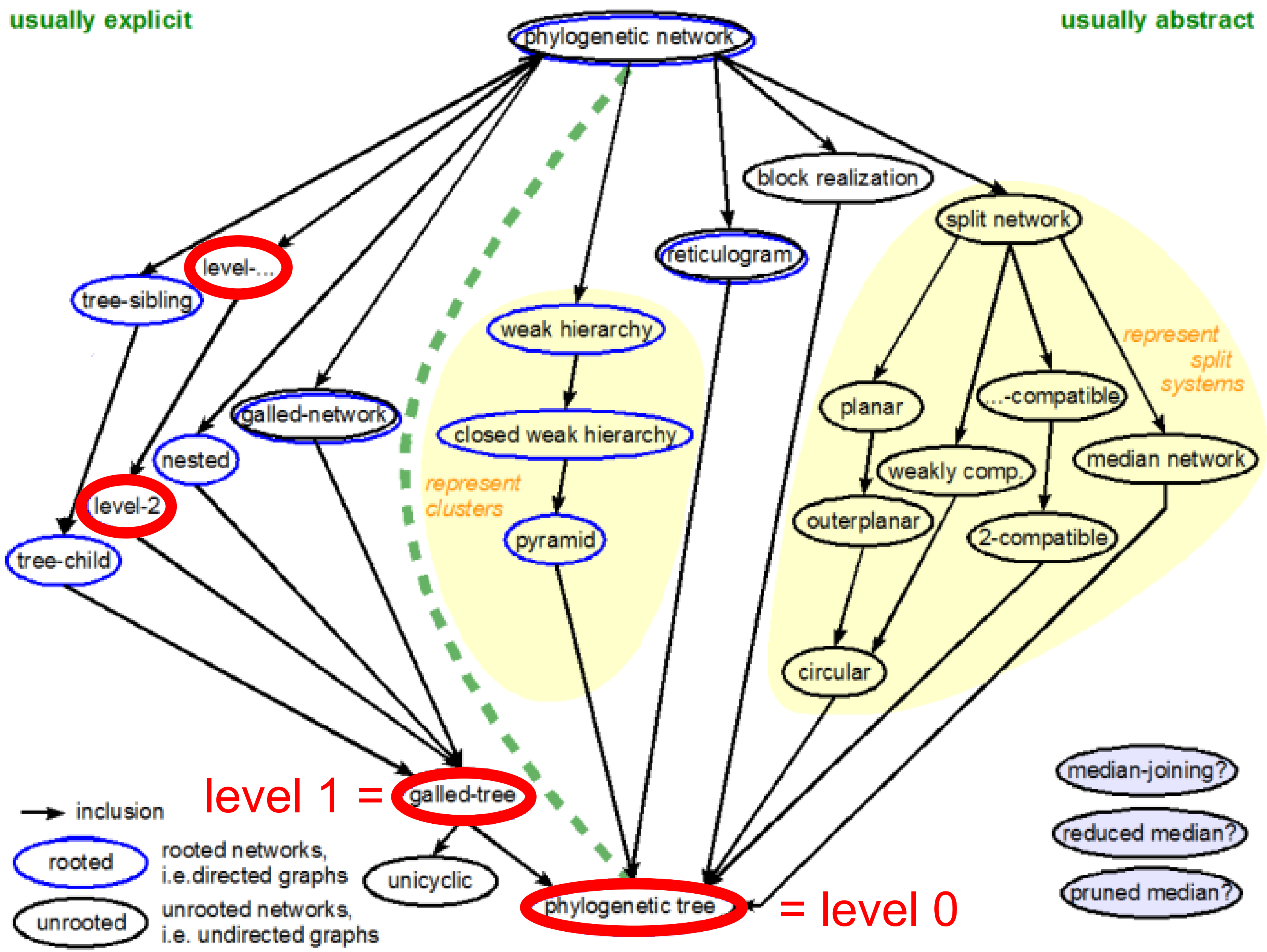
- exactly one vertex has indegree 0 and outdegree 2: the **root**,
- all other vertices have either:
    - indegree 1 and outdegree 2: **split vertices**,
    - indegree 2 and outdegree ≤ 1: **reticulation vertices**,
    - or indegree 1 and outdegree 0: **leaves** labeled by *X*,
- any **blob** has at most *k* reticulation vertices.



N has level 2.

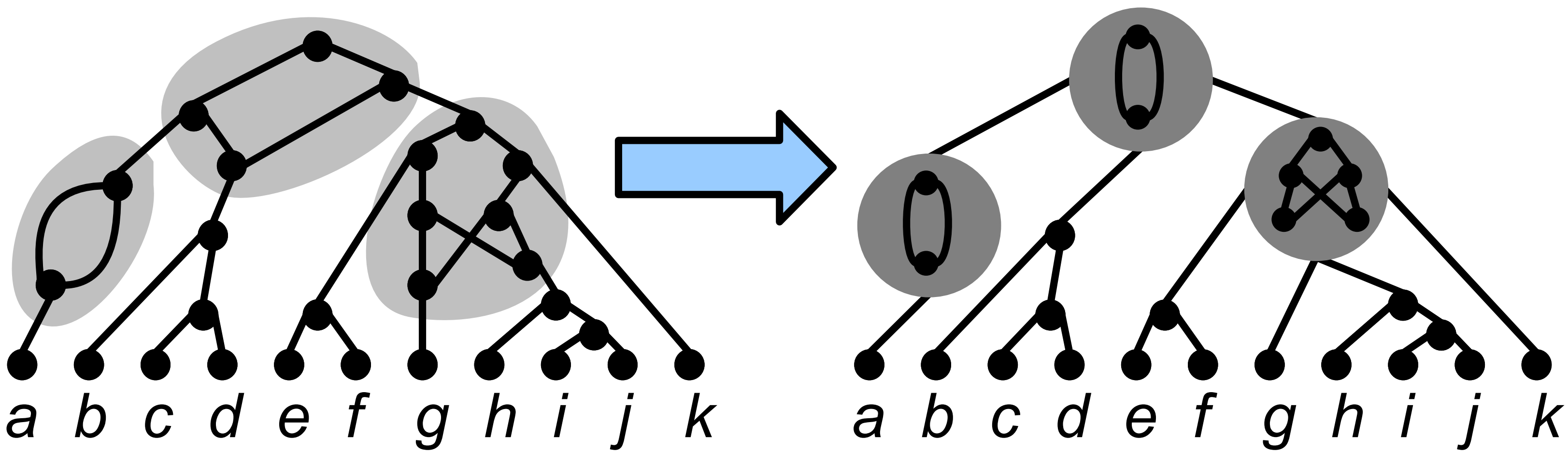
A **blob** is a maximal **biconnected** component of the underlying undirected graph, that is a maximal subgraph which remains connected after the removal of one of its vertices.

Level-*k* phylogenetic networks



Decomposition of level-*k* networks

We formalize the decomposition into biconnected components:



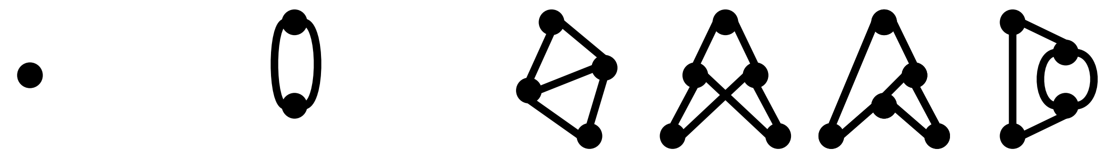
N, a level-*k* network.

N decomposed as a **tree of generators**.

Generators were introduced by van Iersel & al (Recomb 2008) for a restricted class of level-*k* networks.

Level-*k* generators

A **level-*k* generator** is a biconnected level-k network.



$\mathcal{G}^0$ $\mathcal{G}^1$ 2a 2b 2c 2d

The **sides** of the generator are:
- its arcs
- its reticulation vertices of outdegree 0

$\boldsymbol{S_k}$ is the set of generators of level at most *k*.

Decomposition theorem of level-*k* networks

N is a level-*k* network

iff

there exists a sequence $(l_j)_{j\in[1,r]}$ of r locations
(arcs or reticulation vertices of outdegree 0)
and a sequence $(G_j)_{j\in[0,r]}$ of generators of level at most k, such that:

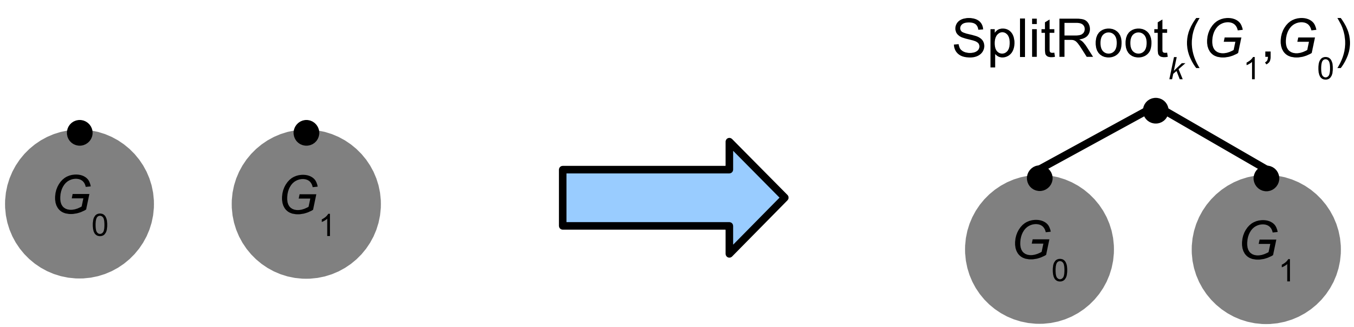
- $N = \text{Attach}_k(l_r, G_r, \text{Attach}_k(\dots \text{Attach}_k(l_2, G_2, \text{Attach}_k(l_1, G_1, G_0))\dots))$,
- or $N = \text{Attach}_k(l_r, G_r, \text{Attach}_k(\dots \text{Attach}_k(l_2, G_2, \text{SplitRoot}_k(G_1, G_0))\dots))$.

Decomposition theorem of level-k networks

N is a level-k network

iff

there exists a sequence $(l_j)_{j \in [1,r]}$ of r locations
(arcs or reticulation vertices of outdegree 0)
and a sequence $(G_j)_{j \in [0,r]}$ of generators of level at most k, such that:
- $N = \mathrm{Attach}_k(l_r, G_r, \mathrm{Attach}_k(\ldots \mathrm{Attach}_k(l_2, G_2, \mathrm{Attach}_k(l_1, G_1, G_0))\ldots))$,
- or $N = \mathrm{Attach}_k(l_r, G_r, \mathrm{Attach}_k(\ldots \mathrm{Attach}_k(l_2, G_2, \mathrm{SplitRoot}_k(G_1, G_0))\ldots))$.



Decomposition theorem of level-k networks

N is a level-k network

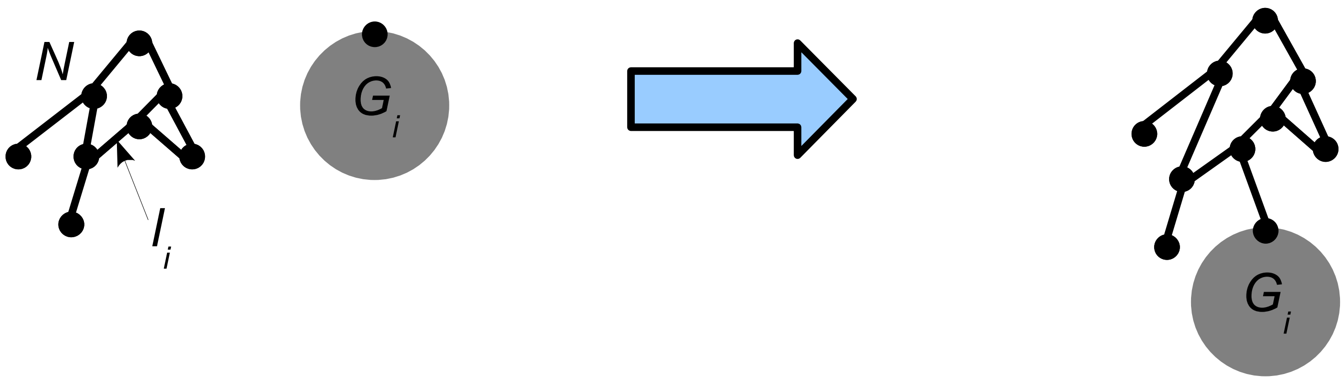
iff

there exists a sequence $(l_j)_{j\in[1,r]}$ of r locations
(arcs or reticulation vertices of outdegree 0)
and a sequence $(G_j)_{j\in[0,r]}$ of generators of level at most k, such that:

- $N = \text{Attach}_k(l_r,G_r,\text{Attach}_k(\dots \text{Attach}_k(l_2,G_2,\text{Attach}_k(l_1,G_1,G_0))\dots))$,
- or $N = \text{Attach}_k(l_r,G_r,\text{Attach}_k(\dots \text{Attach}_k(l_2,G_2,\text{SplitRoot}_k(G_1,G_0))\dots))$.

l_i is an arc of N

$\text{Attach}_k(l_i,G_i,N)$



Decomposition theorem of level-*k* networks

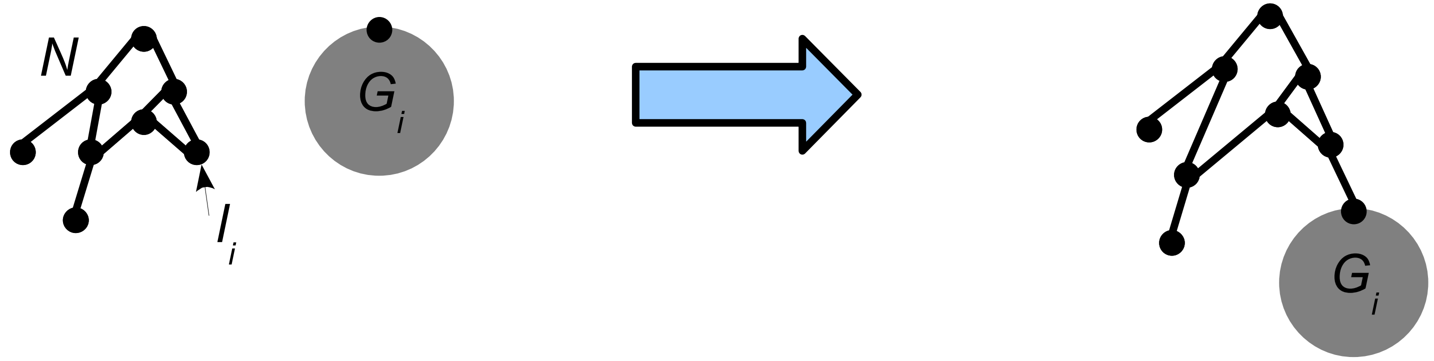
N is a level-*k* network

iff

there exists a sequence $(l_j)_{j\in[1,r]}$ of *r* locations
(arcs or reticulation vertices of outdegree 0)
and a sequence $(G_j)_{j\in[0,r]}$ of generators of level at most *k*, such that:
- $N = Attach_k(l_r, G_r, Attach_k(\ldots Attach_k(l_2, G_2, Attach_k(l_1, G_1, G_0))\ldots))$,
- or $N = Attach_k(l_r, G_r, Attach_k(\ldots Attach_k(l_2, G_2, SplitRoot_k(G_1, G_0))\ldots))$.

l_i is a reticulation vertex of *N*

$Attach_k(l_i, G_i, N)$



Construction of the generators

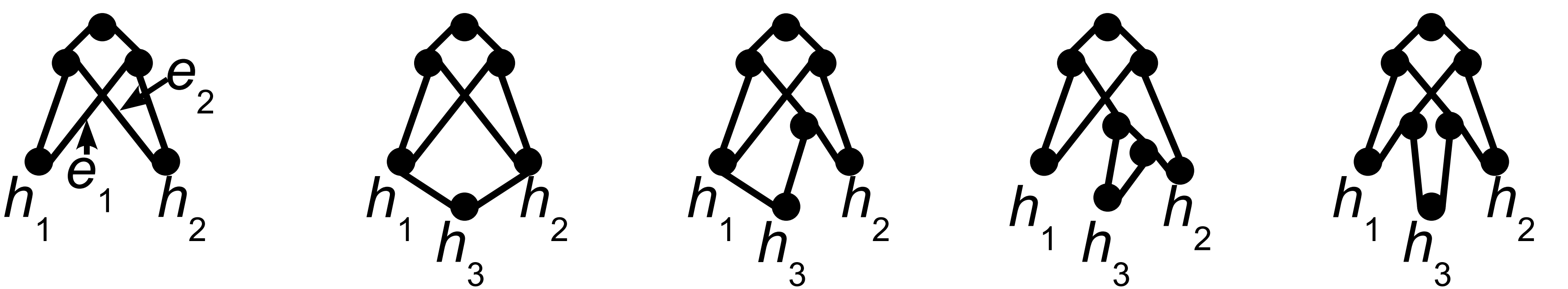
Van Iersel & al give a simple case analysis for level-2.

We give **rules to build level-(k+1) from level-k generators**.

Construction of the generators

We give **rules to build level-(k+1) from level-k generators**.

N is a level-*k* generator.
$R_1(N,X,Y)$ is obtained by:
- choosing two sides *X* and *Y* of *N*, such that if *X* = *Y* then *X* is not a reticulation node (i.e. it is an arc),
- hanging a new reticulation node under *X* and *Y*.

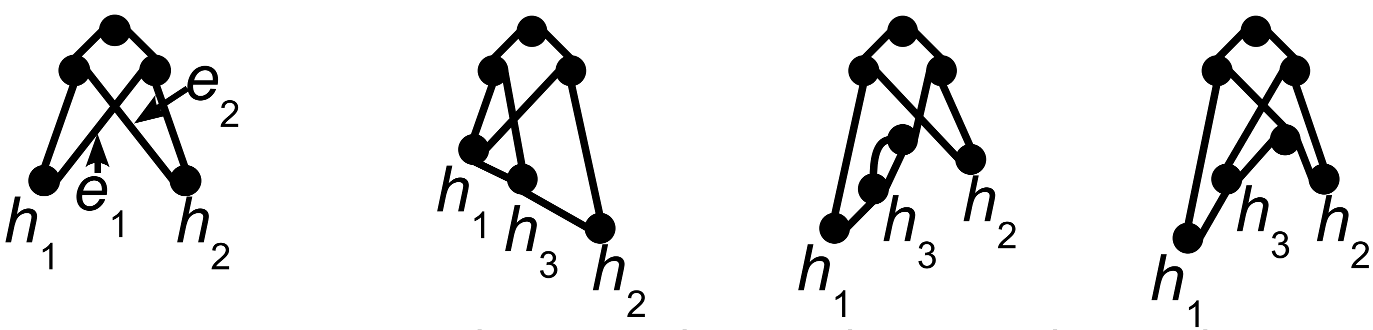


N $R_1(N,h_1,h_2)$ $R_1(N,h_1,e_2)$ $R_1(N,e_2,e_2)$ $R_1(N,e_1,e_2)$

Construction of the generators

We give **rules to build level-(k+1) from level-k generators**.

N is a level-*k* generator.
$R_2(N,X,Y)$ is obtained by:
- choosing a side *X* of *N*, and an arc *Y* of *N*,
- adding an arc from *X* to *Y* (which creates a reticulation node inside arc *Y*).



N $R_2(N,h_1,e_2)$ $R_2(N,e_1,e_1)$ $R_2(N,e_2,e_1)$

Construction of the generators

We give **rules to build level-(k+1) from level-k generators**.

For any level-*k* generator *N*, and any two sides X and Y of N, if $R_1(N,X,Y)$ (resp. $R_2(N,X,Y)$) exists,
then $\boldsymbol{R_1(N,X,Y)}$ (resp. $\boldsymbol{R_2(N,X,Y)}$) **is a level-(k+1) generator**.

For any level-(*k*+1) generator *N*, there **exists a level-*k* generator $\boldsymbol{N_0}$**, and some sides X and Y of N_0 such that
$\boldsymbol{N = R_1(N_0,X,Y)}$ **or N =** $\boldsymbol{R_2(N_0,X,Y)}$.

Construction of the generators

We give **rules to build level-(k+1) from level-k generators**.

For any level-*k* generator *N*, and any two sides X and Y of N, if $R_1(N,X,Y)$ (resp. $R_2(N,X,Y)$) exists,
then $\boldsymbol{R_1(N,X,Y)}$ (resp. $\boldsymbol{R_2(N,X,Y)}$) **is a level-(k+1) generator**.

For any level-(*k*+1) generator *N*, there **exists a level-*k* generator $\boldsymbol{N_0}$**, and some sides X and Y of N_0 such that
$\boldsymbol{N = R_1(N_0,X,Y)}$ **or** $\boldsymbol{N = R_2(N_0,X,Y)}$.

Corollary:
g_k: number of level-k generators.
$g_{k+1} \leq 50\, k^2\, g_k$

Construction of the generators

For any level-k generator N, and any two sides X and Y of N, if $R_1(N,X,Y)$ (resp. $R_2(N,X,Y)$) exists,
then $\mathbf{R_1(N,X,Y)}$ (resp. $\mathbf{R_2(N,X,Y)}$) **is a level-(k+1) generator**.

For any level-(k+1) generator N, there **exists a level-k generator N_0**, and some sides X and Y of N_0 such that
$\mathbf{N = R_1(N_0,X,Y)}$ **or** $\mathbf{N = R_2(N_0,X,Y)}$.

Corollary:
g_k: number of level-k generators.
$g_{k+1} \leq 50\, k^2\, g_k$

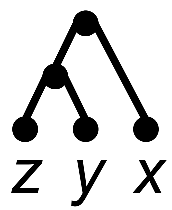
Some of the level-(k+1) generators obtained from level-k generators are **isomorphic**!

→ find lower and upper bounds for g_k

→ generate level-4 generators (65 level-3 generators)

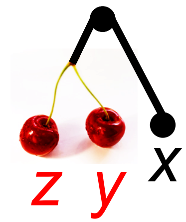
Reconstruction from triplets

A **triplet** $x|yz$ is a rooted phylogenetic tree on 3 taxa $\{x,y,z\}$ such that x, and the father of y and z, are sons of the root.



Reconstruction from triplets

A **triplet** $x|yz$ is a rooted phylogenetic tree on 3 taxa $\{x,y,z\}$ such that x, and the father of y and z, are sons of the root.

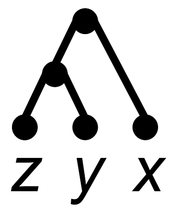


$\{y,z\}$ is then called the **cherry** of $x|yz$.

Restriction of $\mathcal{T}$ to X: $\mathcal{T}_{|X} = \{\ t \in \mathcal{T}\ |\ t$ on taxa x,y,z $\in X\ \}$

Reconstruction from triplets

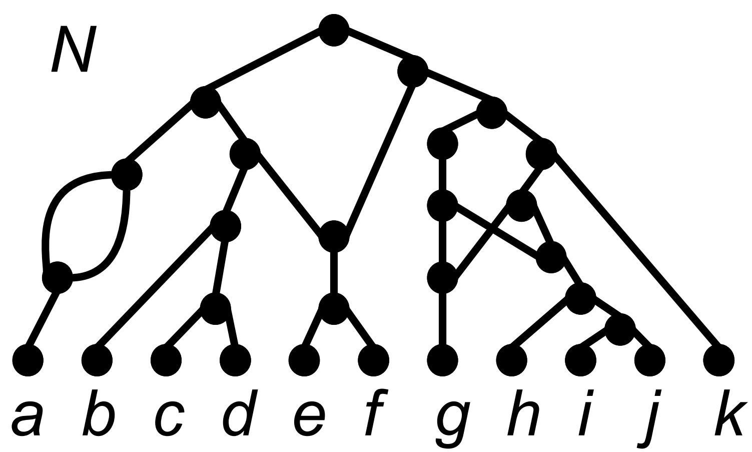
A **triplet** *x*|*yz* is a rooted phylogenetic tree on 3 taxa {*x*,*y*,*z*} such that *x*, and the father of *y* and *z*, are sons of the root.



A triplet *x*|*yz* is **compatible** with a level-*k* phylogenetic network *N* if:

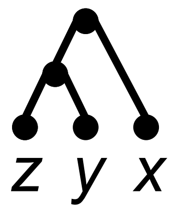
- *N* contains two nodes *u* and *v*
- and pairwise internally vertex-disjoint paths:
    - from *u* to *y*,
    - from *u* to *z*,
    - from v to *u*,
    - and from *v* to *x.*

N

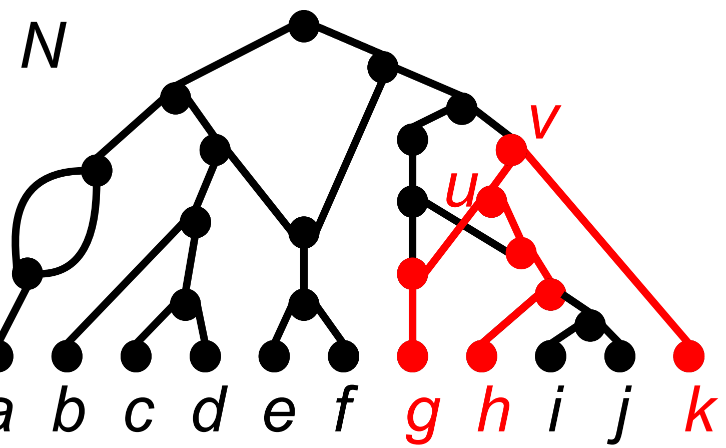


Reconstruction from triplets

A **triplet** *x*|*yz* is a rooted phylogenetic tree on 3 taxa {*x*,*y*,*z*} such that *x*, and the father of *y* and *z*, are sons of the root.



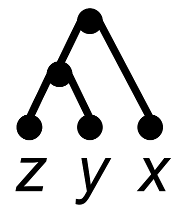
A triplet *x*|*yz* is **compatible** with a level-*k* phylogenetic network *N* if:
- *N* contains two nodes *u* and *v*
- and pairwise internally vertex-disjoint paths:
    - from *u* to *y*,
    - from *u* to *z*,
    - from v to *u*,
    - and from *v* to *x.*



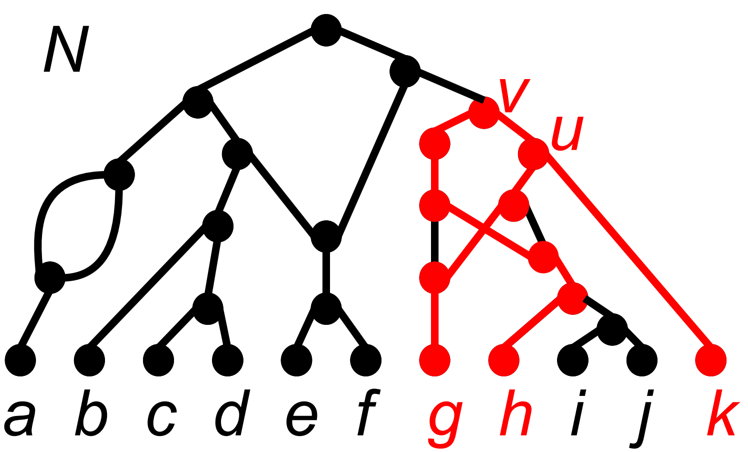
k|*gh* **compatible** with *N*.

Reconstruction from triplets

A **triplet** *x*|*yz* is a rooted phylogenetic tree on 3 taxa {*x*,*y*,*z*} such that *x*, and the father of *y* and *z*, are sons of the root.



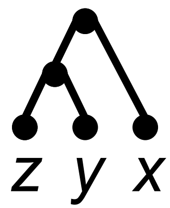
A triplet *x*|*yz* is **compatible** with a level-*k* phylogenetic network *N* if:
- *N* contains two nodes *u* and *v*
- and pairwise internally vertex-disjoint paths:
    - from *u* to *y*,
    - from *u* to *z*,
    - from v to *u*,
    - and from *v* to *x*.



h|*gk* **compatible** with *N*.

Reconstruction from triplets

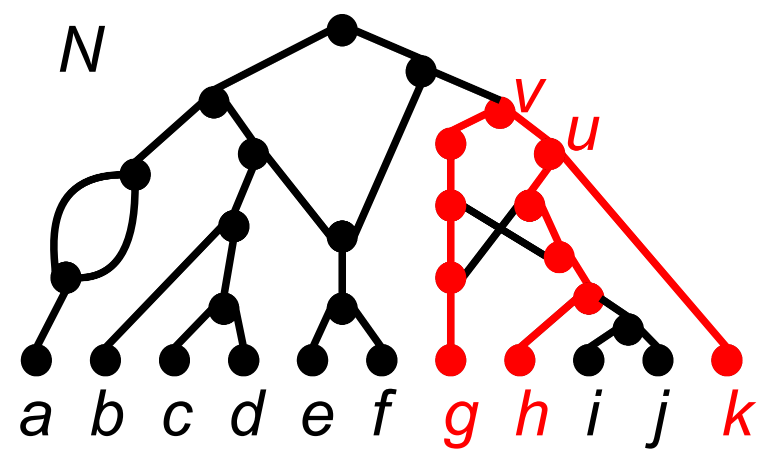
A **triplet** *x*|*yz* is a rooted phylogenetic tree on 3 taxa {*x*,*y*,*z*} such that *x*, and the father of *y* and *z*, are sons of the root.



A triplet *x*|*yz* is **compatible** with a level-*k* phylogenetic network *N* if:

- *N* contains two nodes *u* and *v*
- and pairwise internally vertex-disjoint paths:
    - from *u* to *y*,
    - from *u* to *z*,
    - from v to *u*,
    - and from *v* to *x*.

g|*hk* **compatible** with *N.*



Reconstruction from triplets

Why use triplets as input of a tree or network reconstruction algorithm?

Under a coalescent model, an inferred tree on strictly more than three taxa, is most likely to be wrong because of discrepancies in gene and species tree.

(Degnan & Rosenberg, 2006)

Who uses triplets as input of a tree or network reconstruction algorithms?
- ???
- people who use trees as input?
- people will when it is implemented in Dendroscope or SplitsTree?

Reconstruction from triplets: BUILD

The set $\mathcal{T}(N)$ of **all triplets compatible** with a level-k network N can be computed in $O(|\mathcal{T}(N)|) = O(n^3)$

(dynamic programming, Byrka, Gawrychowski, Huber, Kelk, 2008)

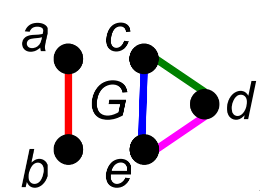
A **tree** T compatible with a set $\mathcal{T}$ of triplets can be **reconstructed** in $O(|\mathcal{T}|+n^2 \log n)$.

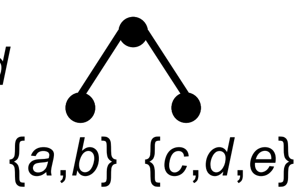
(BUILD, top-down algorithm, Aho, Sagiv, Szymanski, Ullman, 1981)
(efficient implementation by Henzinger, King, Warnow, 1999)

Recursive algorithm on X:
- build the following graph G:
    - taxa as vertices,
    - edge $\{x,y\}$ if $\exists t \in \mathcal{T}_{|X}$ such that $\{x,y\}$ is a cherry of t.
- vertices in ≠ connected components are in ≠ subtrees
- apply algorithm on each connected component

$X=\{a,b,c,d,e\}$

$\mathcal{T}_{|X}= \{a|cd, a|ce, c|ab, d|ab, c|de\}$





Reconstruction from triplets: BUILD

The set $\mathcal{T}(N)$ of **all triplets compatible** with a level-k network N can be computed in $O(|\mathcal{T}(N)|) = O(n^3)$

(dynamic programming, Byrka, Gawrychowski, Huber, Kelk, 2008)

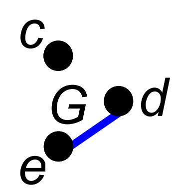
A **tree** T compatible with a set $\mathcal{T}$ of triplets can be **reconstructed** in $O(|\mathcal{T}|+n^2 \log n)$.

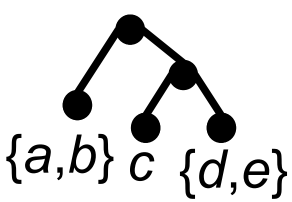
(BUILD, top-down algorithm, Aho, Sagiv, Szymanski, Ullman, 1981)
(efficient implementation by Henzinger, King, Warnow, 1999)

Recursive algorithm on X:
- build the following graph G:
    - taxa as vertices,
    - edge $\{x,y\}$ if $\exists t \in \mathcal{T}_{|X}$ such that $\{x,y\}$ is a cherry of t.
- vertices in ≠ connected components are in ≠ subtrees
- apply algorithm on each connected component

$X=\{c,d,e\}$

$\mathcal{T}_{|X} = \{c|de\}$





Reconstruction from triplets: BUILD

The set $\mathcal{T}(N)$ of **all triplets compatible** with a level-k network N can be computed in $O(|\mathcal{T}(N)|) = O(n^3)$

(dynamic programming, Byrka, Gawrychowski, Huber, Kelk, 2008)

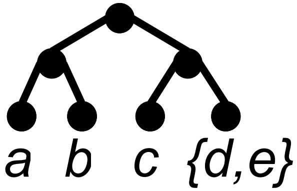
A **tree** T compatible with a set $\mathcal{T}$ of triplets can be **reconstructed** in $O(|\mathcal{T}|+n^2 \log n)$.

(BUILD, top-down algorithm, Aho, Sagiv, Szymanski, Ullman, 1981)
(efficient implementation by Henzinger, King, Warnow, 1999)

Recursive algorithm on X:
- build the following graph G:
    - taxa as vertices,
    - edge $\{x,y\}$ if $\exists t \in \mathcal{T}_{|X}$ such that $\{x,y\}$ is a cherry of t.
- vertices in ≠ connected components are in ≠ subtrees
- apply algorithm on each connected component

$X=\{a,b\}$

$\mathcal{T}_{|X} = \{\}$



Reconstruction from triplets: BUILD

The set $\mathcal{T}(N)$ of **all triplets compatible** with a level-k network N can be computed in $O(|\mathcal{T}(N)|) = O(n^3)$

(dynamic programming, Byrka, Gawrychowski, Huber, Kelk, 2008)

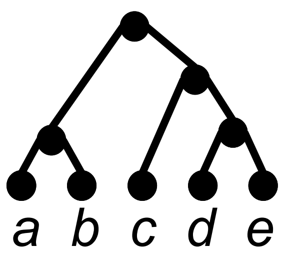
A **tree** T compatible with a set $\mathcal{T}$ of triplets can be **reconstructed** in $O(|\mathcal{T}|+n^2 \log n)$.

(BUILD, top-down algorithm, Aho, Sagiv, Szymanski, Ullman, 1981)
(efficient implementation by Henzinger, King, Warnow, SODA 1996)

Recursive algorithm on X:
- build the following graph G:
    - taxa as vertices,
    - edge $\{x,y\}$ if $\exists t \in \mathcal{T}_{|X}$ such that $\{x,y\}$ is a cherry of t.
- vertices in ≠ connected components are in ≠ subtrees
- apply algorithm on each connected component

$X=\{d,e\}$

$\mathcal{T}_{|X} = \{\}$



Reconstruction from triplets

The problem of **reconstructing a level-1 network** compatible with a set of triplets is **NP-complete**.

(Jansson, Nguyen, Sung, 2004)

A **level-1 network** compatible with a **dense** set of triplets can be **reconstructed** in $O(|\mathcal{T}(N)|) = O(n^3)$.
dense = at least 1 triplet on each set of 3 leaves exists in $\mathcal{T}$.

(Jansson, Nguyen, Sung, 2004)

A **level-2 network** compatible with a **dense** set of triplets can be **reconstructed** in $O(n^8)$.

(van Iersel et al, Recomb 2008)

Based on a decomposition of the triplet set with **SN-sets**, which corresponds to **some decomposition of the network**.

The obstruction approach

Idea:
- reduce **global conflicts** in the triplet set to **local conflicts**,
- if no local conflict, build the **local configuration of the network**.

The obstruction approach

Idea:
- reduce **global conflicts** in the triplet set to **local conflicts**,
- if no local conflict, build the **local configuration of the network**.

Requirements:
- enough information on the triplet set to find the conflicts
$\rightarrow$ **dense** set.
- enough information to build the local configuration
$\rightarrow$ **extremely dense** set, i.e. no triplet is missing (input=$\mathcal{T}(N)$).

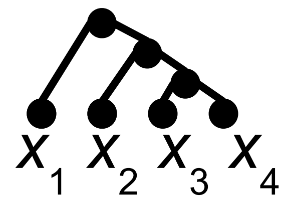
Characterization of trees from triplets

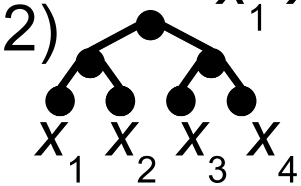
A dense triplet set $\mathcal{T}$ is compatible with a tree T

iff

no set of three leaves is present in two different triplets of $\mathcal{T}$, and all triplet sets on four leaves are isomorphic:

- either to $\{x_1|x_2x_3, x_1|x_2x_4, x_1|x_3x_4, x_2|x_3x_4\}$ (case 1)
- or to $\{x_1|x_3x_4, x_2|x_3x_4, x_3|x_1x_2, x_4|x_1x_2\}$ (case 2)





iff

$\mathcal{T}$ does not contain any triplet set isomorphic to any of the four following obstructions:
$\{a|bc, c|ab\}$, $\{a|bc, c|bd, d|ab\}$, $\{a|bc, c|bd, d|ac\}$, $\{a|bc, a|bd, d|ac\}$.

Similar characterizations were found by Dress (1997), Guillemot & Berry (2007).

Consequences of the characterization

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

Certifying algorithms return, with each output, an **easily checked certificate** that the output has not been compromised by a **bug**.

The certificate we provide is:
- the tree, if it can be reconstructed,

- an obstruction on 4 triplets, otherwise.

Consequences of the characterization

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

Certifying algorithms return, with each output, an **easily checked certificate** that the output has not been compromised by a **bug**.

The certificate we provide is:
- the tree, if it can be reconstructed,
  → checking that all the input triplets are compatible with the tree is easy!
- an obstruction on 4 triplets, otherwise.
  → checking that it is isomorphic to one of the 4 obstructions is easy!

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set $\mathcal{T}$, when possible.

1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $\mathcal{T}_{|X}$

- For any leaf *x*, consider $\mathcal{T}_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.

2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

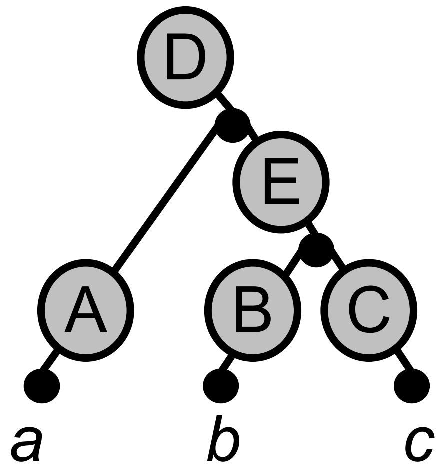
1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.



A:{*a*|*bc*,*b*|*ax*,*c*|*ax*,*x*|*bc*}
B:{*a*|*bc*,*a*|*bx*,*a*|*cx*,*c*|*bx*}
C:{*a*|*bc*,*a*|*bx*,*a*|*cx*,*b*|*cx*}
D:{*a*|*bc*,*x*|*ab*,*x*|*ac*,*x*|*bc*}
E:{*a*|*bc*,*a*|*bx*,*a*|*cx*,*x*|*bc*}

2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

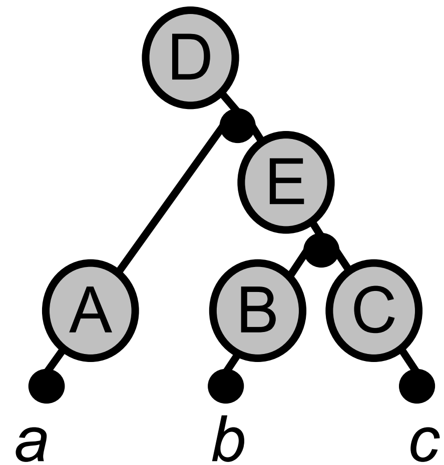
A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.
No zone is correct? Obstruction!

- For any leaf set corresponding to one zone, apply the recursive algorithm.



A:{*a*|*bc*,*b*|*ax*,*c*|*ax*,*x*|*bc*}
B:{*a*|*bc*,*a*|*bx*,*a*|*cx*,*c*|*bx*}
C:{*a*|*bc*,*a*|*bx*,*a*|*cx*,*b*|*cx*}
D:{*a*|*bc*,*x*|*ab*,*x*|*ac*,*x*|*bc*}
E:{*a*|*bc*,*a*|*bx*,*a*|*cx*,*x*|*bc*}

- Connect the recursively obtained trees.

2. Check that all input triplets are in the obtained tree.
A triplet is not compatible? Obstruction!

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

1. Recursive algorithm on *X*: *X*={*a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i*,*j*,*k*}

- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.

2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

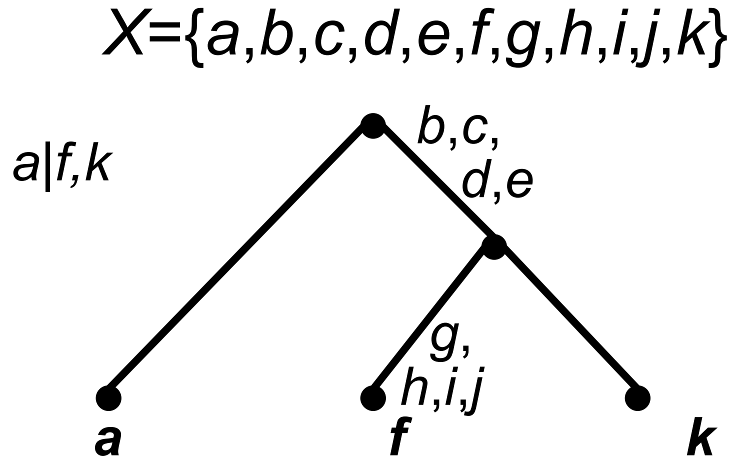
1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.



2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

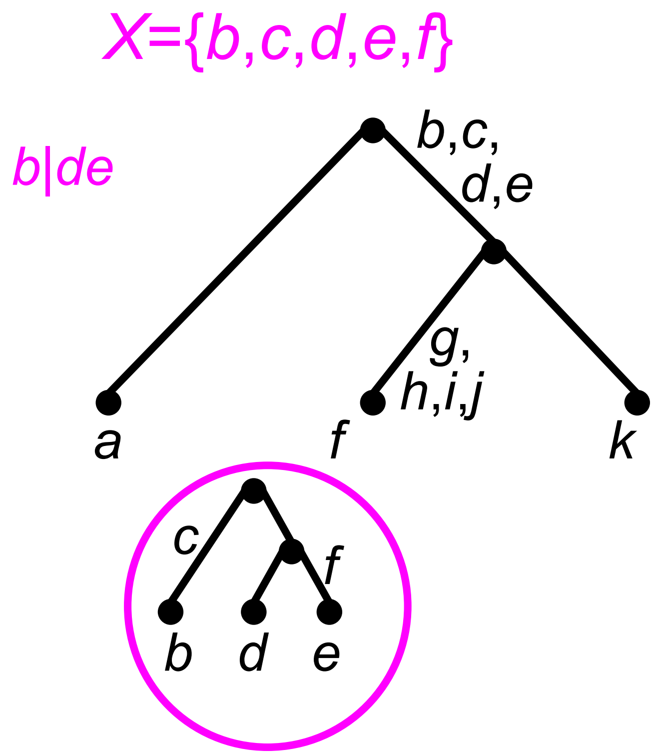
1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.



2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

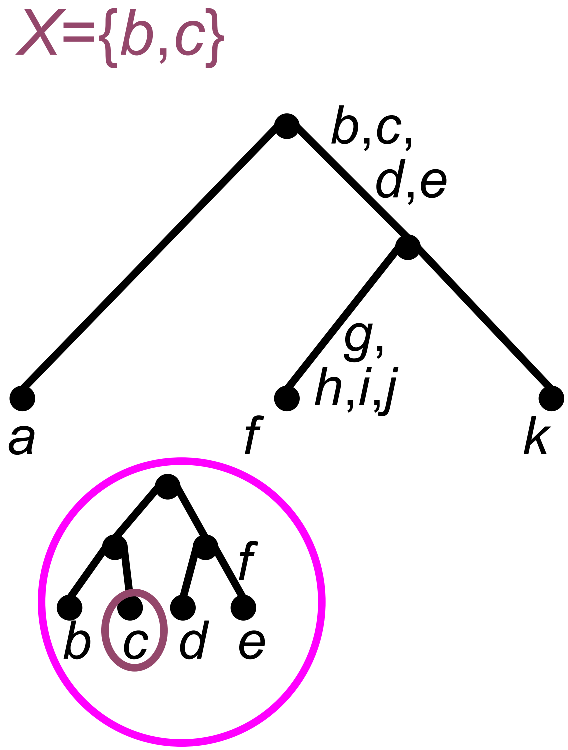
1. Recursive algorithm on *X*:

- Take any triplet $a|bc$ in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.



2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

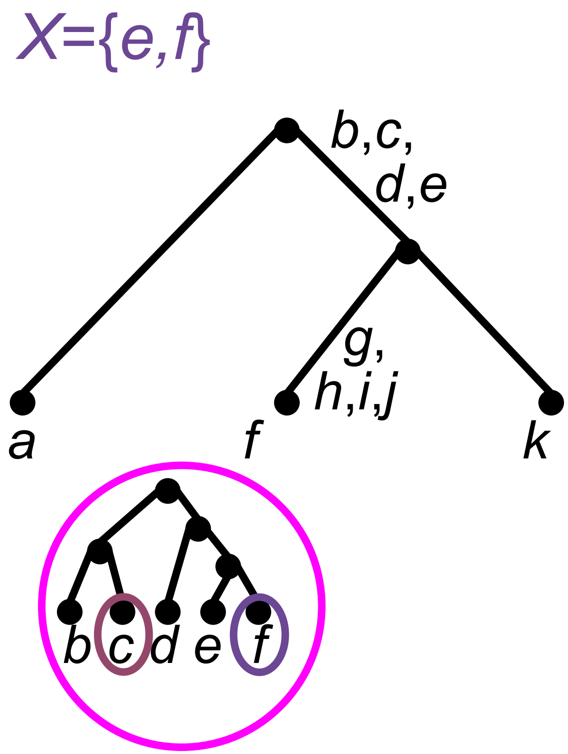
1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.



2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

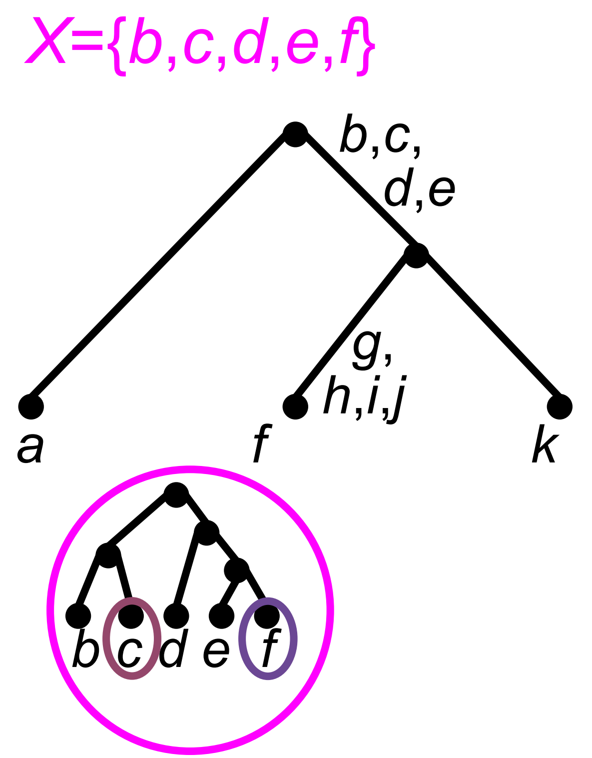
1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.

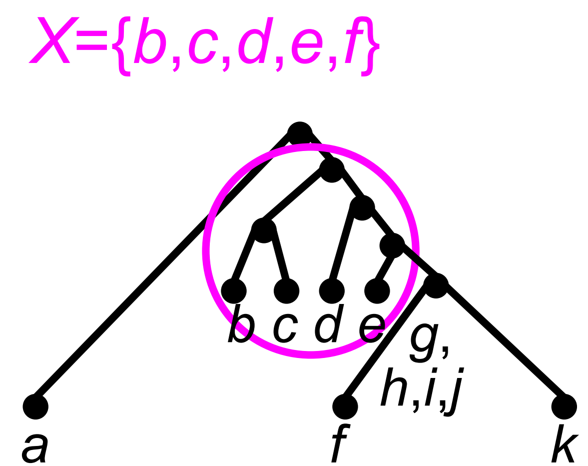


2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

1. Recursive algorithm on *X*:



- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.

2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

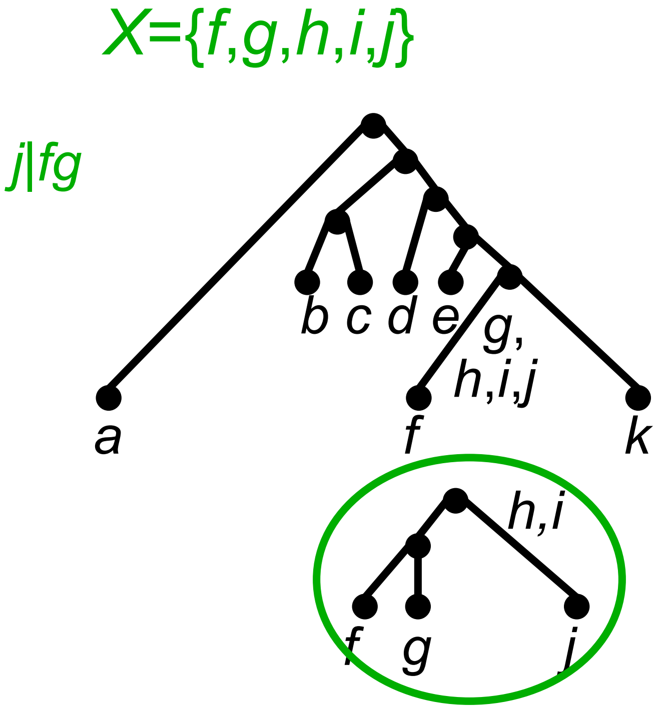
1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.



2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

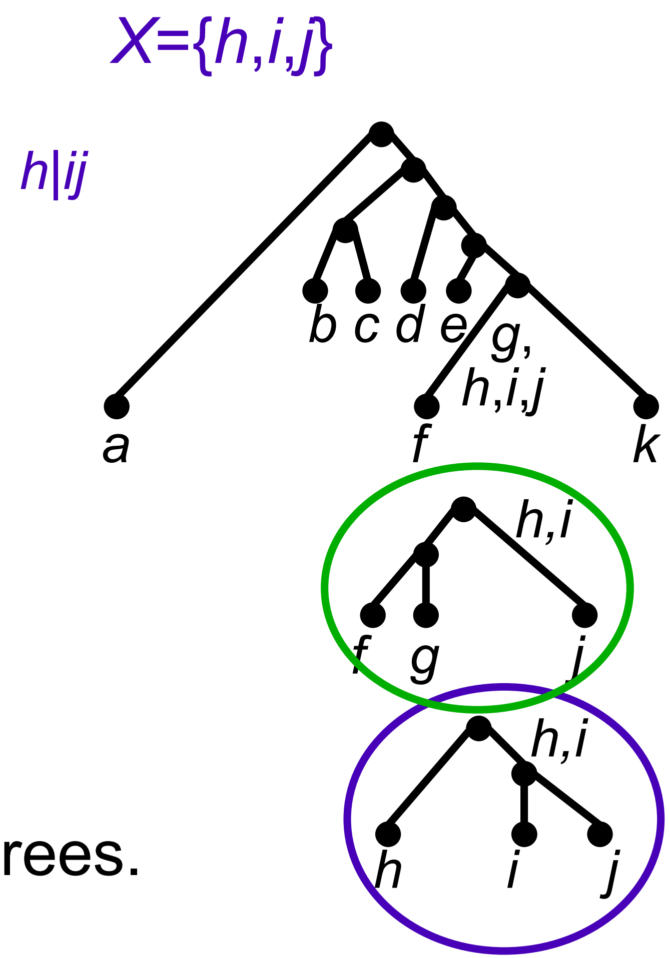
1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.



2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

1. Recursive algorithm on *X*:

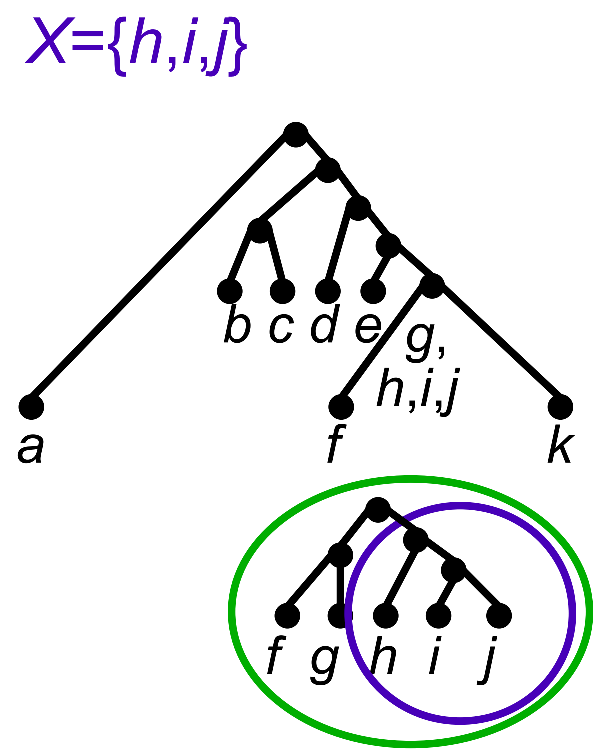
- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.

2. Check that all input triplets are in the obtained tree.



The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

1. Recursive algorithm on *X*:

- Take any triplet *a*|*bc* in $T_{|X}$

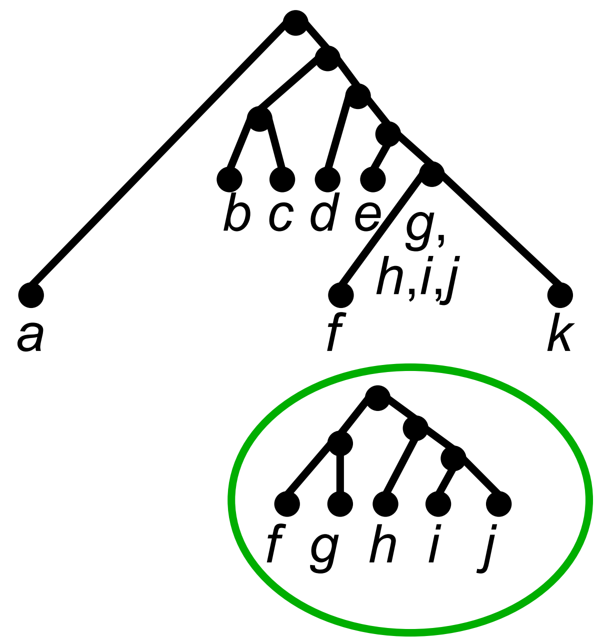
- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.

2. Check that all input triplets are in the obtained tree.

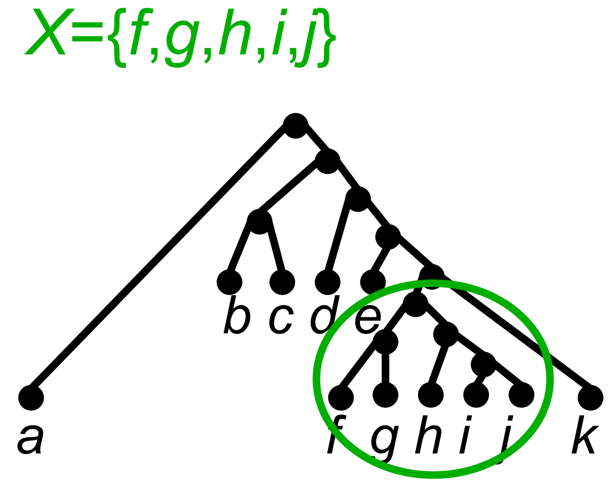
X={*f*,*g*,*h*,*i*,*j*}



The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

1. Recursive algorithm on *X*:



- Take any triplet *a*|*bc* in $T_{|X}$

- For any leaf *x*, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put *x*.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.

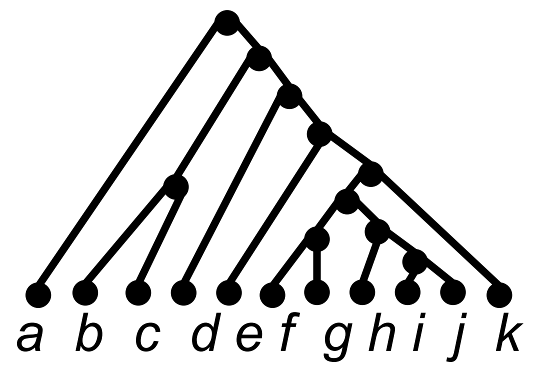
2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

1. Recursive algorithm on *X*:

$X=\{a,b,c,d,e,f,g,h,i,j,k\}$



- Take any triplet $a|bc$ in $T_{|X}$

- For any leaf x, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put x.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.

2. Check that all input triplets are in the obtained tree.

The tree reconstruction algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

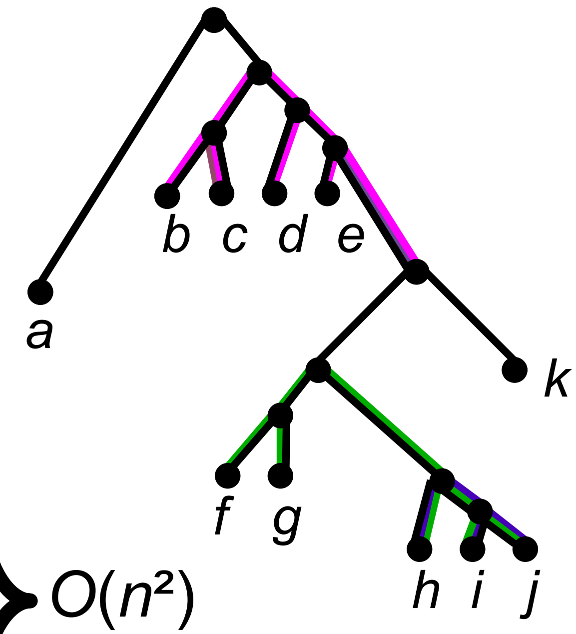
1. Recursive algorithm on X:

- Take any triplet $a|bc$ in $T_{|X}$

- For any leaf x, consider $T_{|\{a,b,c,x\}}$ to know in which of 5 different zones you should put x.

- For any leaf set corresponding to one zone, apply the recursive algorithm.

- Connect the recursively obtained trees.



$O(n^2)$

$O(n)$ leaves
→ $O(n)$ edges
→ $O(n)$ non-overlapping sets of edges
→ **$O(n)$ recursive calls**

2. Check that all input triplets are in the obtained tree. $O(n^3)$

FPT algorithm for minimum triplet edition

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

Corollary:
An obstruction on 4 leaves can be found in $O(n^3)$.

Maximum Compatible Subset of Rooted Triples:
Input: Triplet set T, integer $t \leq |T|$ (nb of bad triplets).
Question: Is there a subset of T of size at least $|T|-t$ compatible with a tree?

FPT algorithm for minimum triplet edition

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

Corollary:
An obstruction on 4 leaves can be found in $O(n^3)$.

Maximum Compatible Subset of Rooted Triples:
Input: Triplet set T, integer $t \leq |T|$ (nb of bad triplets).
Question: Is there a subset of T of size at least $|T|-t$ compatible with a tree?

NP-complete.

(proofs by Bryant 1997, Jansson 2001, Wu 2004)

$O((|T|+n^2)3^n)$ algorithm.

(Wu, 2004)

FPT algorithm for minimum triplet edition

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

Corollary:
An obstruction on 4 leaves can be found in $O(n^3)$.

Maximum Compatible Subset of Rooted Triples:
Input: Triplet set T, integer $t \leq |T|$ (nb of bad triplets).
Question: Is there a subset of T of size at least $|T|-t$ compatible with a tree?

FPT-algorithm for dense sets:
- find an obstruction $O(n^3)$
- edit one of its triplets:
  → 2 possibilities for each triplet
  → total of 6 possibilities

Total complexity: $O(n^3 6^t)$

FPT algorithm for minimum triplet edition

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

Corollary:
An obstruction on 4 leaves can be found in $O(n^3)$.

Maximum Compatible Subset of Rooted Triples:
Input: Triplet set T, integer $t \leq |T|$ (nb of bad triplets).
Question: Is there a subset of T of size at least $|T|$-t compatible with a tree?

FPT-algorithm for dense sets:
- find an obstruction $O(n^3)$
- edit one of its triplets:
  → 2 possibilities for each triplet
  → total of 6 possibilities

Total complexity: $O(n^3 6^t)$

- Get a better complexity?
- Program a faster implementation?

Ongoing work with Vincent Berry (Montpellier, Bioinformatics) and Christophe Paul (Montpellier, Graph Theory)

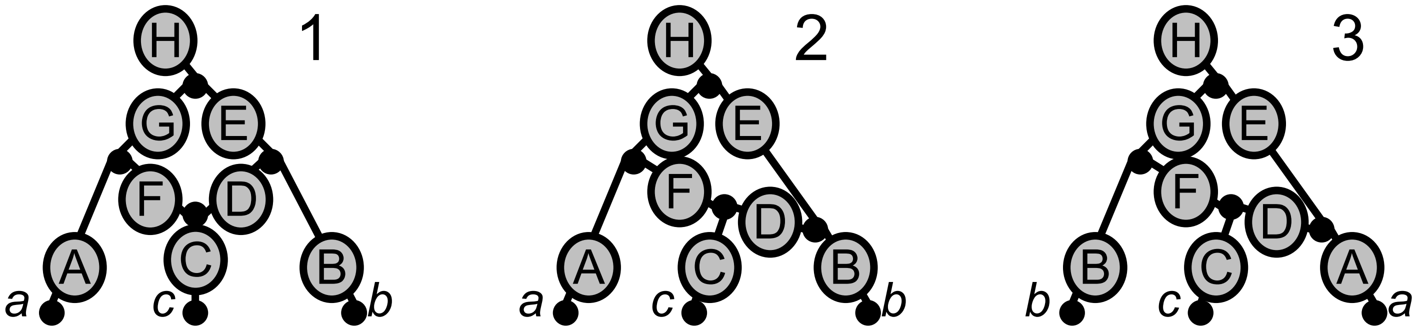
Extremely dense level-1 decision algorithm

A simple **certifying** algorithm to reconstruct a tree from a dense triplet set T, when possible.

Corollary:
Similar $O(n^3)$ algorithm to decide whether a triplet set T is extremely dense for some level-1 network.

Differences:
- different starting point:
if T is not compatible with a tree and is extremely dense, then T contains a triplet subset isomorphic to $\{a|bc, b|ac\}$.
$\rightarrow$ start with $\{a|bc, b|ac\}$

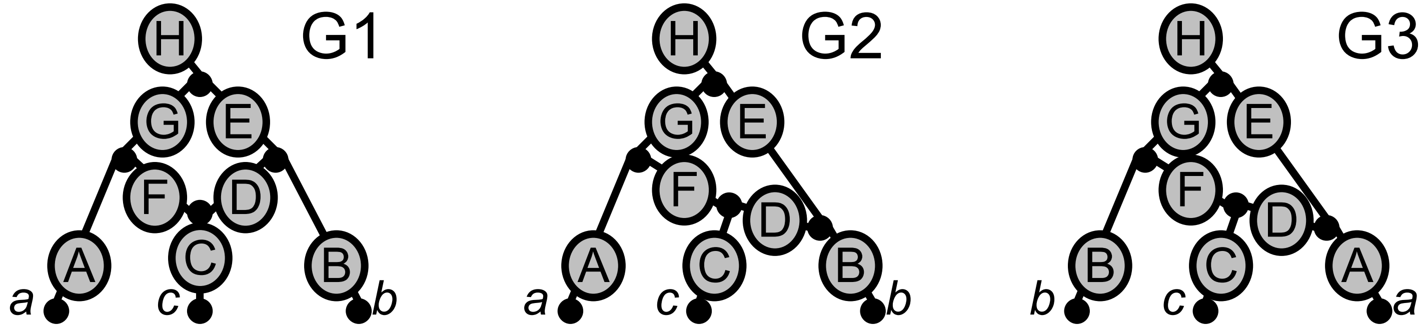
- different zones:



Extremely dense level-1 decision algorithm

$O(n^3)$ algorithm to decide whether a triplet set T is **extremely dense** for some level-1 network.

Different zones:



Different triplet sets define zones D, E, F, G in configurations G1, G2, G3

→ detect some incompatibilities,
→ other incompatibilities found when inserting a subnetwork.
→ other incompatibilities found by a final triplet check.

The same triplet sets define each zone A, B, C, H in every configurations G1, G2, G3

→ ambiguity between two forms,
→ which is the real reticulation node?

Identifiability of galled trees

A strict level-1 network (galled tree) is **identifiable** by its set of triplets if it is the only strict level-1 network which is compatible with exactly this set of triplets.

Characterization of triplet-identifiable galled trees:

A strict level-1 network is **identifiable** by its set of triplets iff each blob contains at least 5 vertices

→ Any **softwired** galled tree is identifiable by its set of clusters?
→ Characterize cluster / triplet sets compatible with a **unique** level-1 network?
→ Characterize cluster / triplet sets compatible with a level-1 network?
(find obstructions like for trees)

Ongoing work with Katharina Huber (Norwich UK, Mathematics and Computational Biology)

Questions?

Thank you for attention!