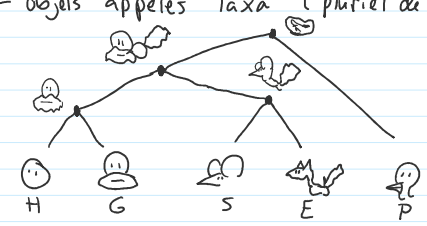
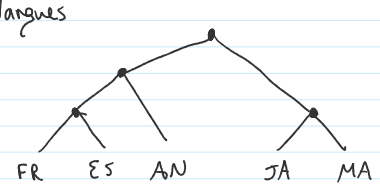
Phylogénétique

- Étude de relations évolutives entre objets
  - objets appelés taxa (pluriel de taxon)

ex:



ex: langues



Suppositions:
- évolution en arbre
- chaque taxon ancestral donne lieu à 2 taxa enfants (arbres binaire)
- feuilles = taxa contemporains (d'aujourd'hui)
  noeuds internes = taxa ancestraux inconnus

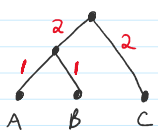
Entrée: feuilles (séquences, distances, ...)
Sortie: arbre
- étiquette sur les arêtes qui représente un temps d'évol.
- état des ancêtres

Approches:
1. par distances
2. par caractères
3. statistiques

1. Approches par distances

ex: 3 taxa A, B, C $D_{x,y}$ = distance entre X et Y

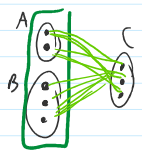
D	A	B	C
A	0	2	5
B	2	0	5
C	5	5	0



- Algorithme UPGMA
  - unweighted pair group with arithmetic mean

Entrée: matrice D (rangées = taxa, colonnes = taxa)
$D_{x,y}$ = dist. X et Y symétrique

1) choisir A, B qui minimisent $D_{A,B}$
2) ajouter un nouveau taxon qu'on va appeler (A∪B)
   → ajouter une rangée + colonne à D pour (A∪B)
3) retirer les rangées A et B
4) pour chaque taxon C restant calculer $D_{(A\cup B),C}$



$D_{(A\cup B),C}$ = moyenne des dists entre les taxa (A∪B) et C

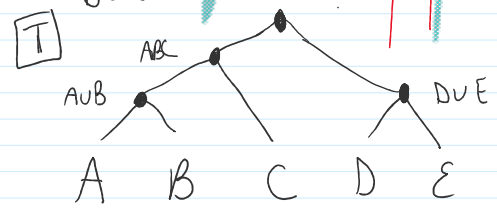
$$= \frac{1}{\#dists} \cdot \sum_{x\in(A\cup B)} \sum_{y\in C} D_{x,y}$$

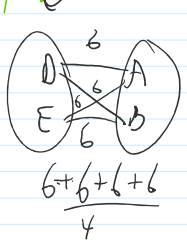
$$= \frac{1}{|A\cup B|\cdot|C|} \cdot \sum_{x\in(A\cup B)} \cdot \sum_{y\in C} D_{x,y}$$

5) goto 1) jusqu'à ce qu'il reste 1 taxon

ex: D

	A	B	C	D	E	A∪B	D∪E	A∪B ∪C
A	0	2	4	6	6			
B		0	4	6	6			
C			0	6	6	4	6	
D				0	4	6		
E					0	6		
A∪B						0	6	
D∪E							0	

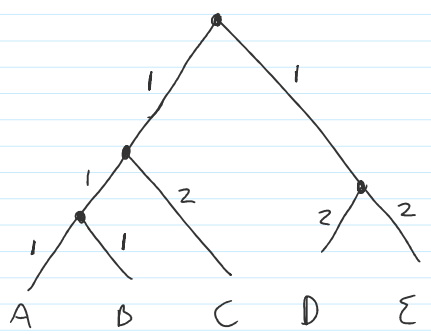




Q: peut-on étiqueter les branches de façon à ce que dans l'arbre T,

$dist_T(X,Y) = D_{x,y}$ $dist_T(X,Y)$ = somme des poids des branches sur le chemin X, Y dans T

	A	B	C	D	E
A	0	2	4	6	6
B		0	4	6	6
C			0	6	6
D				0	4
E					0



Un tel étiquetage fonctionne si la distance D (matrice) est ultramétrique.

- Une distance D est ultramétrique s'il existe un arbre

Un clustering fonctionne si la distance D (matrice) est ultramétrique.

- Une distance D est ultramétrique s'il existe un arbre T qui satisfait les conditions:
  ① Pour tous taxa u,v, $D_{u,v} = dist_T(u,v)$
  ② Soit r la racine de T. Alors ∀ taxa u,v,
  $dist_T(r,u) = dist_T(r,v)$

Théorème: si D est ultramétrique, alors UPGMA reconstruit un arbre capable de satisfaire ① et ②

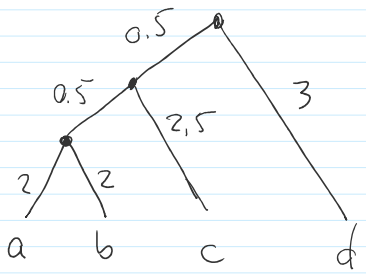
Dans les données réelles, les distances ne sont jamais ultramétrique.

Une autre approche, Neighbor-Joining, permet d'alléger ces conditions. Cette approche se concentre sur les distances additives.

- Une distance D est additive s'il existe un arbre T tel que
① $\forall u,v$, $D_{uv} = dist_T(u,v)$.

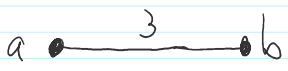
Q: comment vérifier qu'une distance est additive?

	a	b	c	d
a	0	4	5	6
b		0	5	6
c			0	6
d				0

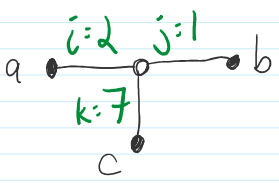


∃? arbre additif? Cet arbre n'a pas besoin d'être enraciné.

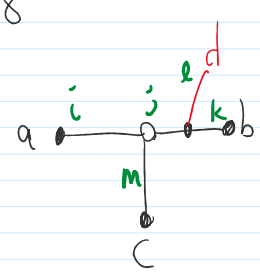
	a	b	c	d
a	0	3	9	6
b		0	8	8
c			0	5
d				0

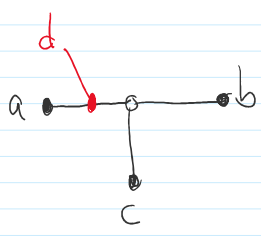


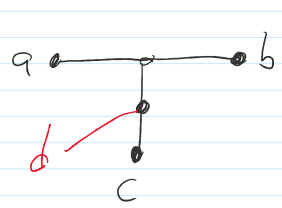
$i+j = 3$
$i+k = 9$
$j+k = 8$



$i+j+l = 6$
$i+j+k = 3$
⋮



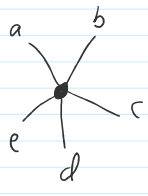


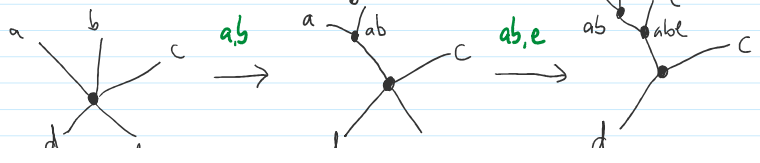


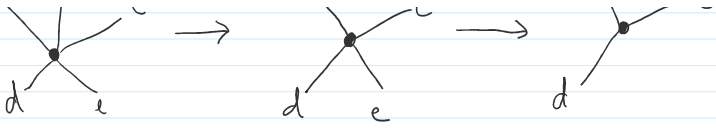
- Remarque: si une distance D est additive, alors il existe un seul arbre unique qui peut représenter les distances.

- L'algo Neighbor-Joining trouve un arbre additif s'il existe.
S'il n'existe pas, l'algo approxime "bien" les distances.

Idée: • démarrer d'un arbre étoile
• raffiner cet arbre une étape à la fois, en joignant successivement deux voisins du centre.



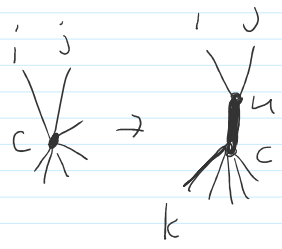




- On arrête quand le centre a 3 voisins.
  ↳ arbre non-enraciné

Q: comment choisir les voisins du centre à joindre? (Compliqué)

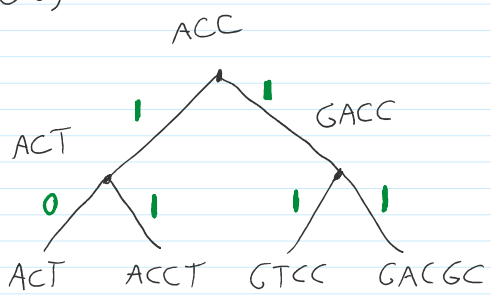
algoNJ(D)
  soit T l'arbre étoile avec 1 feuille par taxon
  c ← centre de T
  tant que c a plus que 3 voisins
    parmi les m voisins de c, choisir
    les deux voisins i,j qui minimisent
    $(m-2)D_{i,j} - \left(\sum_{k=1}^{m} D_{i,k} + \sum_{k=1}^{m} D_{j,k}\right)$
    (i,j proches ; i et j loins des autres)
    Joindre i,j en créant un nouveau noeud u
    Mettre à jour $D_{u,k}$ $\forall k \neq i,j$
      $D_{u,k} = \frac{1}{2}\left[D_{i,k} + D_{j,k} - D_{i,j}\right]$
  x



Approches par caractères

Entrée: séquences $S_1, S_2, \ldots, S_m$
Sortie: arbre T avec
- feuilles étiquetées par les S_i
- tout noeud interne v soit étiqueté par une séquence s(v)
- minimiser la somme des coûts sur les branches (maximiser un score)

ex:
S_1	ACT
S_2	ACCT
S_3	GTCC
S_4	GACGC



rechercheArbre($S_1, S_2, \ldots, S_m$)
  minT = null, minCout = ∞
  pour chaque arbre T avec les S_i aux feuilles
    • trouver le meilleur étiquetage des noeuds internes de T, qui minimise cout(T)
    • si cout(T) < minCout
      minT = T
      minCout = cout(T)
  x
  return T

Complexité: $\Omega((2n-3)!!) = \Omega((2n-3)\cdot(2n-5)\cdot(2n-7)\cdot\ldots\cdot 5\cdot 3\cdot 1)$
(au moins)
$= \Omega(\text{absurde})$

Problème de la petite parcimonie

- Parcimonie: l'hypothèse + simple = préférée
  = demande # min de changements

- Parcimonie : l'hypothèse + simple = préférée
  = demande # min de changements

- "Petite" : sous-problème de la parcimonie générale
  général : T est inconnu
  petite : T est connu

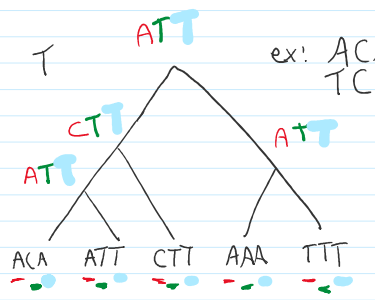
Entrée : séquences $S_1, S_2, \dots, S_m$, arbre T avec S_i aux feuilles
Sortie : étiquetage $s(v)$ des noeuds de T qui minimise

$$\sum_{uv \in E(T)} d(s(u), s(v))$$

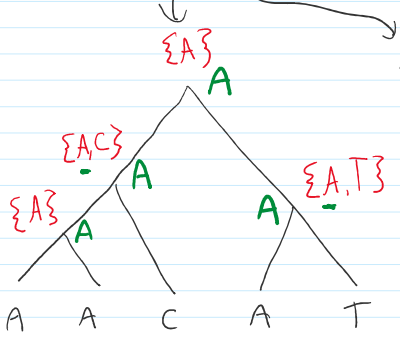
arêtes

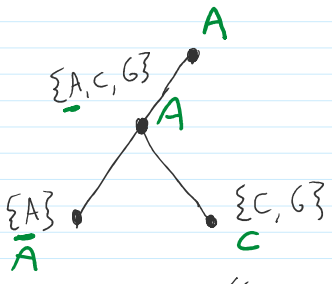
où d est une distance.

Version simple : • d = distance de Hamming
• $S_1, S_2, \dots, S_m$ sont de la même longueur
• $d(S_i, S_j)$ = nombre de positions avec caractère différents
$= |\{ k : 1 \le k \le |S_i| \text{ et } S_i[k] \neq S_j[k] \}|$



On peut traiter chaque pos. une à la fois





// si parent rien en commun, choix arbitraire

```
C = dict()  // C[v] = caractères candidats au noeud v
getCandidats(v)
    si v = feuille
        C[v] = caractère assigné à v
    sinon
        Soit v1, v2 les enfants de v
        getCandidats(v1)
        getCandidats(v2)
        si C[v1] ∩ C[v2] ≠ ∅
            C[v] = C[v1] ∩ C[v2]
        sinon
            C[v] = C[v1] ∪ C[v2]
getCandidat(racine)
```

Complexité :
- chaque noeud v est visité une fois dans l'exécution de cet algo
- temps pour traiter un noeud $O(|\Sigma|)$
Σ = alphabet
pour calculer $\cap, \cup$
$\Rightarrow O(n|\Sigma|)$
n = nb de noeuds

// phase 2

```
getEtiquette(v)
    si v = racine
        s(v) = élément arbitraire de C[v]
    sinon si v ≠ feuille
        Soit p le parent de v
        si s(p) ∈ C[v]
```

soit p le parent de v
si $s(p) \in C[v]$
  $s(v) = s(p)$
sinon
  $s(v)$ = élém. arbitraire de $C[v]$

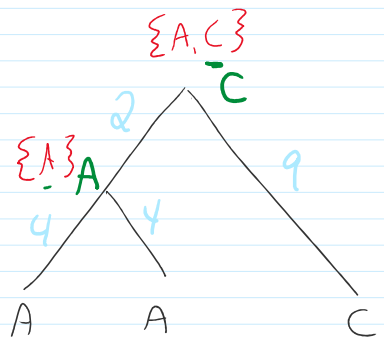
si (v ≠ feuille)
  get Etiquette(v_1) //v_1, v_2 enfants de v
  get Etiquette(v_2)

Version générale: matrice de coûts entre caractères

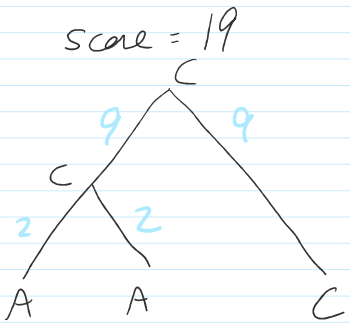
$$d(S_i, S_j) = \sum_{k=1}^{|S_i|} M[S_i[k], S_j[k]]$$

ex: M :

	A	C	G	T
A	4	2	1	2
C	2	9	3	1
G	1	3	5	2
T	2	1	2	4



Si on veut maximiser la somme des scores sur les branches, ceci n'est pas un bon exemple



score = 22

Avec un seul caractère,
pour trouver un étiquetage qui maximise

$$\sum_{uv \in E(T)} M[s(u), s(v)]$$

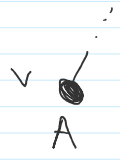
on utilise la prog. dynamique.

On définit, pour chaque noeud v et chaque $a \in \Sigma$,

$V[v, a]$ = score maximum du sous-arbre enraciné en v avec la contrainte que $s(v) = a$

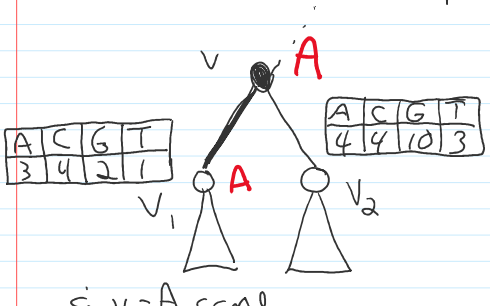
On cherche, comme score optimal, l'entrée $\max_{a \in \Sigma} V[\text{racine}, a]$

Cas de base, v = feuille



$$V[v, a] = \begin{cases} 0 & \text{si } a \text{ est le caractère assigné à la feuille} \\ -\infty & \text{sinon} \end{cases}$$

Récurrence
Soit v_1, v_2 les enfants de v

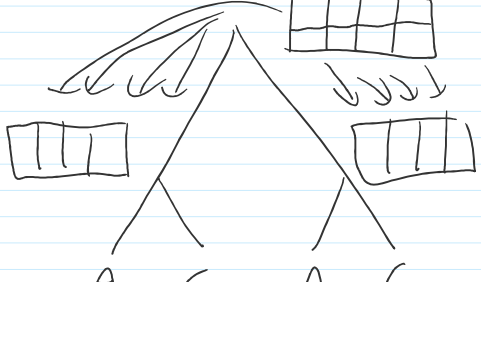


$$V[v, a] = \max_{b \in \Sigma} [V[v_1, b] + M[a, b]] + \max_{b \in \Sigma} [V[v_2, b] + M[a, b]]$$

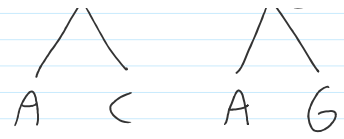
si $v_1 = A$, score $V[v_1, A] + M[A, A]$
si $v_1 = C$, score $V[v_1, C] + M[A, C]$

M :

	A	C	G	T
A	4	2	1	2
C	2	9	3	1
G	1	3	5	2
T	2	1	2	4



$V[v,:,c] + M[A,c]$

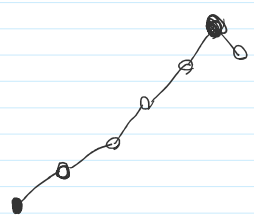


- Calculer toutes les entrées $V[v,a]$, il faut un temps "nb d'entrées dans V" × "temps pour calculer une entrée"
  "nb de noeuds" × "nb de symboles" × "nb de symboles"
  $= O(n \cdot |\Sigma|^2)$

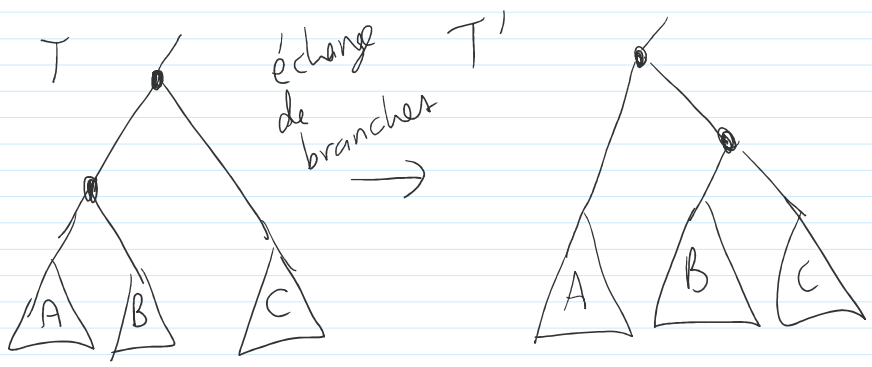
- Algo de $\cap, \cup$: algo de Fitch
- Algo de prog. dyn. : algo de Sankoff-Rousseau

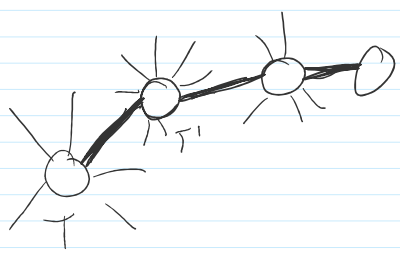
En pratique pour trouver le meilleur arbre T

$T \leftarrow$ arbre Neighbor Joining $(S_1, \dots, S_m)$
fini = false
tant que pas fini
  pour chaque voisin T' de T
    si $cout(T') < cout(T)$
      $T \leftarrow T'$
  si T n'a pas été mis à jour, fini = true



Voisin d'un arbre :





À retenir : • approches par dist. : UPGMA (en détail)
Neighbor Joining (principe, ultramétrique vs additive)

• approches par caractères : algo $\cap, \cup$ min. Hamming
algo prog. dyn. max score M
contexte général (trouver T)