Filogenética molecular (II)

Bioinformática, 13-3-18

Basado en Kevin Yip-CSE-CUHK
(Universidad china de Hong-Kong)

Hoy/Mañana …

1. Distancia evolutiva y modelos de mutación
2. Árboles: Las estructuras jerárquicas relacionando diferentes objetos biológicos
    1. Formatos de archivo
    2. reconstrucción de árboles filogenéticos
    3. **Métodos basados en secuencias**
        - **máxima parsimonia**
        - **Máxima verosimilitud**
    4. **métodos basados en distancias**
        - **UPGMA**
        - **Unión de vecinos**

máxima parsimonia

- Suponemos: Un árbol es probable que sea correcto **si implica pocas mutaciones**
- Razón fundamental:
  - Las mutaciones son poco frecuentes
  - "Navaja de Occam": **La explicación más simple es probablemente la correcta**

máxima parsimonia

- Problema general:
  - Dado un conjunto de secuencias (las hojas)
  - encontrar una **topología** de árbol con raíz y las **secuencias ancestrales** del árbol de forma que el número total de mutaciones en el árbol sea mínimo

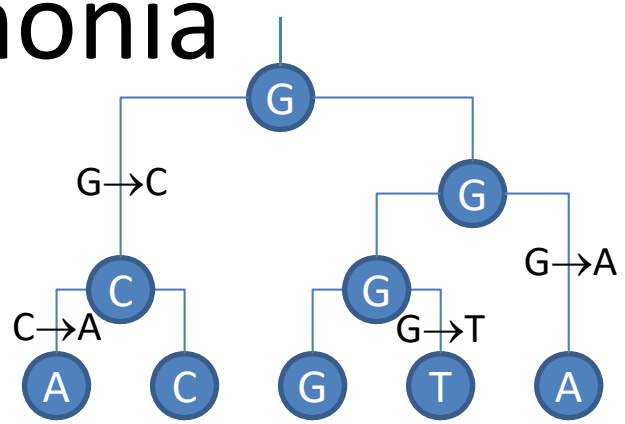
- NP duro: no hay algoritmos en tiempo polinómico

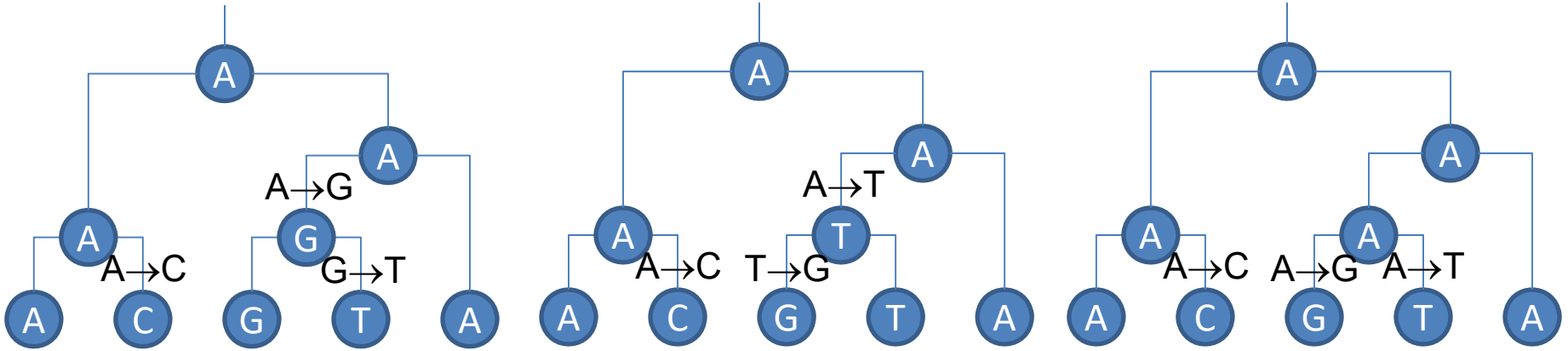
máxima parsimonia

- Problema restringido:
  - **Dado** un conjunto de secuencias y una **topología de árbol** con raíz
  - Encontrar las secuencias ancestrales del árbol de forma que el número total de mutaciones en el árbol sea mínimo
- Ahora nos centraremos en el problema restringido

Ejemplo de parsimonia

- Vamos a considerar una sola posición
  - Asumiendo que las posiciones son independientes, sólo necesitamos un algoritmo para una
  - Veremos un ejemplo con más posiciones
- En el árbol de la derecha, el número de mutaciones es 4
  - ¿Es el mínimo (es decir, la solución más parsimoniosa)?
  - Para esta topología del árbol, el número mínimo de mutaciones es 3. Hay tres conjuntos de estados ancestrales que resultan en este número de mutaciones, que se muestran en los tres árboles de debajo





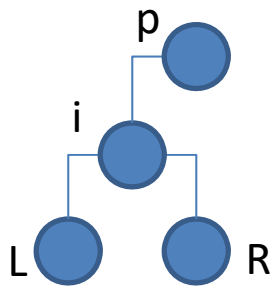
problema de parsimonia

- ¿Cómo asignar estados ancestrales para **minimizar** el número total de mutaciones?
- Ideas: dado un nodo,
  - Si ambos hijos tienen el mismo estado, probablemente es bueno adoptar ese estado
  - Si los hijos tienen dos estados diferentes, probablemente es bueno adoptar uno de ellos
  - **Retrasar la decisión** de la elección exacta hasta que el padre también ha expresado una preferencia

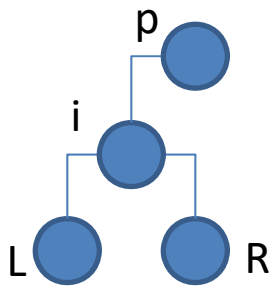
Algoritmo de Fitch

El algoritmo de Fitch: versión simple

- El algoritmo de Fitch: Si sólo se necesita una solución
  - Para cada nodo interno i con los padres y los hijos p, L y R, vamos a determinar su conjunto de preferencias S_i y su carácter final C_i que reduzca al mínimo el número total de mutaciones



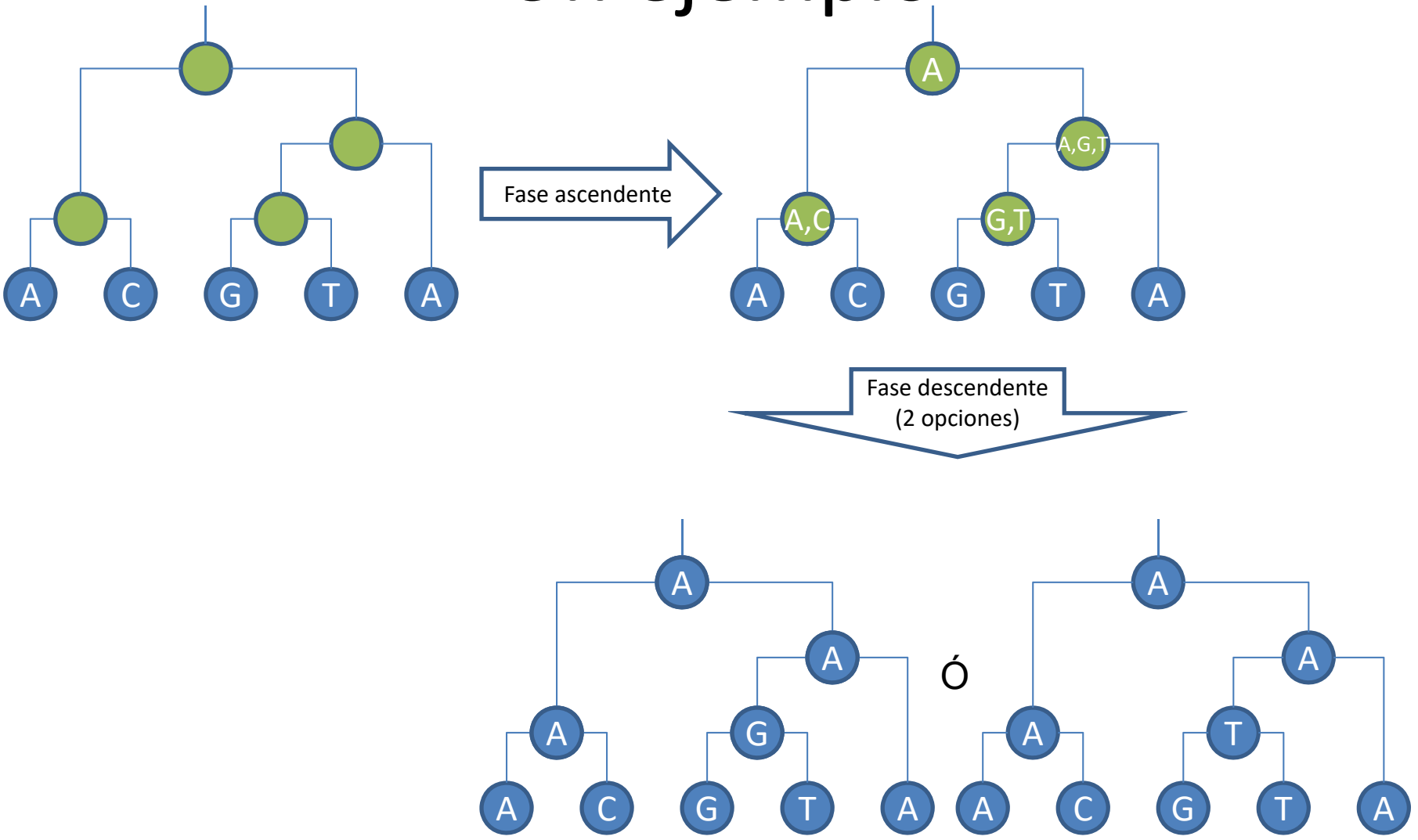
El algoritmo de Fitch: versión simple



– Pasos:

1. Para cada nodo hoja i, S_i es el carácter de la hoja i
2. fase ascendente: Para cada i nodo interno,
   Si $(S_L \cap S_R) = \{\}$ // L y R no están de acuerdo: coger ambos
   $S_i := S_L \cup S_R$
   else // L y R están de acuerdo en algo: tomar el acuerdo
   $S_i := S_L \cap S_R$
3. fase descendente: En primer lugar elegir cualquier $C_{raíz}$ en $S_{raíz}$.
   Luego, para cada i otro nodo interno,
   si $C_p \in S_i$ // p está de acuerdo con i en algo: cogerlo
   $C_i := C_p$
   else // p no está de acuerdo con i: usar las preferencias de i
   $C_i :=$ Elegir uno de S_i

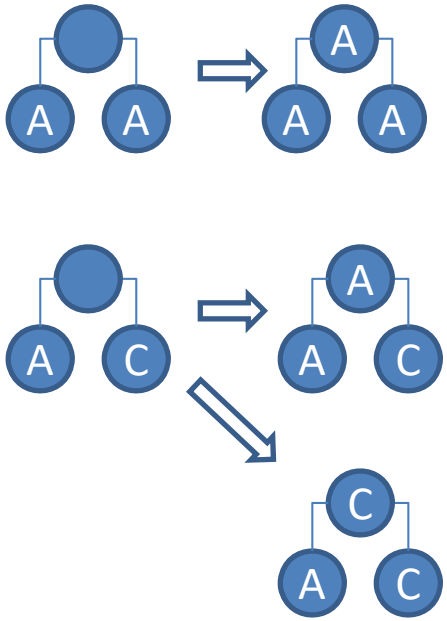
Un ejemplo


conjunto de preferencias
carácter final elegido
A
C
G
T
A
Fase ascendente
A
A,G,T
A,C
G,T
A
C
G
T
A
Fase descendente
(2 opciones)
A
A
A
G
A
C
G
T
A
Ó
A
A
A
T
A
C
G
T
A

¿Por qué funciona?

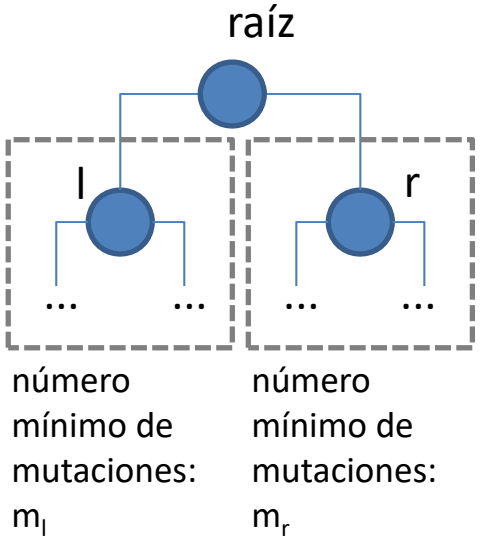
- Demostración por inducción
  - Cuando hay dos hojas, sólo hay dos casos:
    - Tienen el mismo carácter
      - número mínimo real de mutaciones: 0
      - El algoritmo da el mismo número
    - Tienen diferentes caracteres
      - número mínimo de mutaciones en: 1
      - El algoritmo también da el mismo número

    Por lo tanto el algoritmo es óptimo

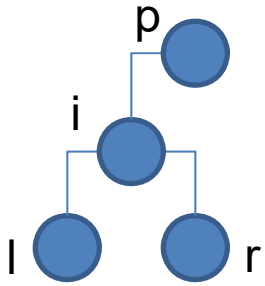


¿Por qué funciona?

- Supongamos que el algoritmo es capaz de minimizar el número de mutaciones para árboles con k o menos hojas
- Ahora, por un árbol con hojas k + 1,
  - Se compone de una raíz conectado a dos sub-árboles con raíces l y r, ambos con k o menos hojas
  - Dos casos:
    - Si $S_l \cap S_r \neq \{\}$, El algoritmo da una solución con $m_l + m_r$ mutaciones, que es óptima debido a la hipótesis de inducción
    - Si $S_l \cap S_r = \{\}$, El algoritmo da una solución con $m_l + m_r + 1$ mutaciones, que también es óptima ya que una mutación adicional debe ser introducida entre la raíz y uno de sus hijos

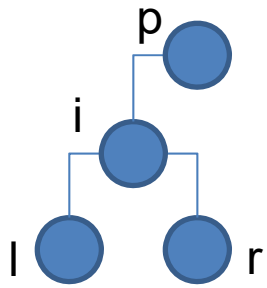


El algoritmo: versión extendida



- Si necesita todas las soluciones de coste mínimo
  - Pasos:
    1. Para cada nodo hoja i, S_i es el carácter de la hoja
    2. fase ascendente (igual que antes): Para cada nodo interno i,
       Si ($S_l \cap S_r$) = {} // L y R no están de acuerdo: hay que tomar ambos conjuntos
       $S_i := S_l \cup S_r$
       else // L y R están de acuerdo en algo: tomarlo
       $S_i := S_l \cap S_r$

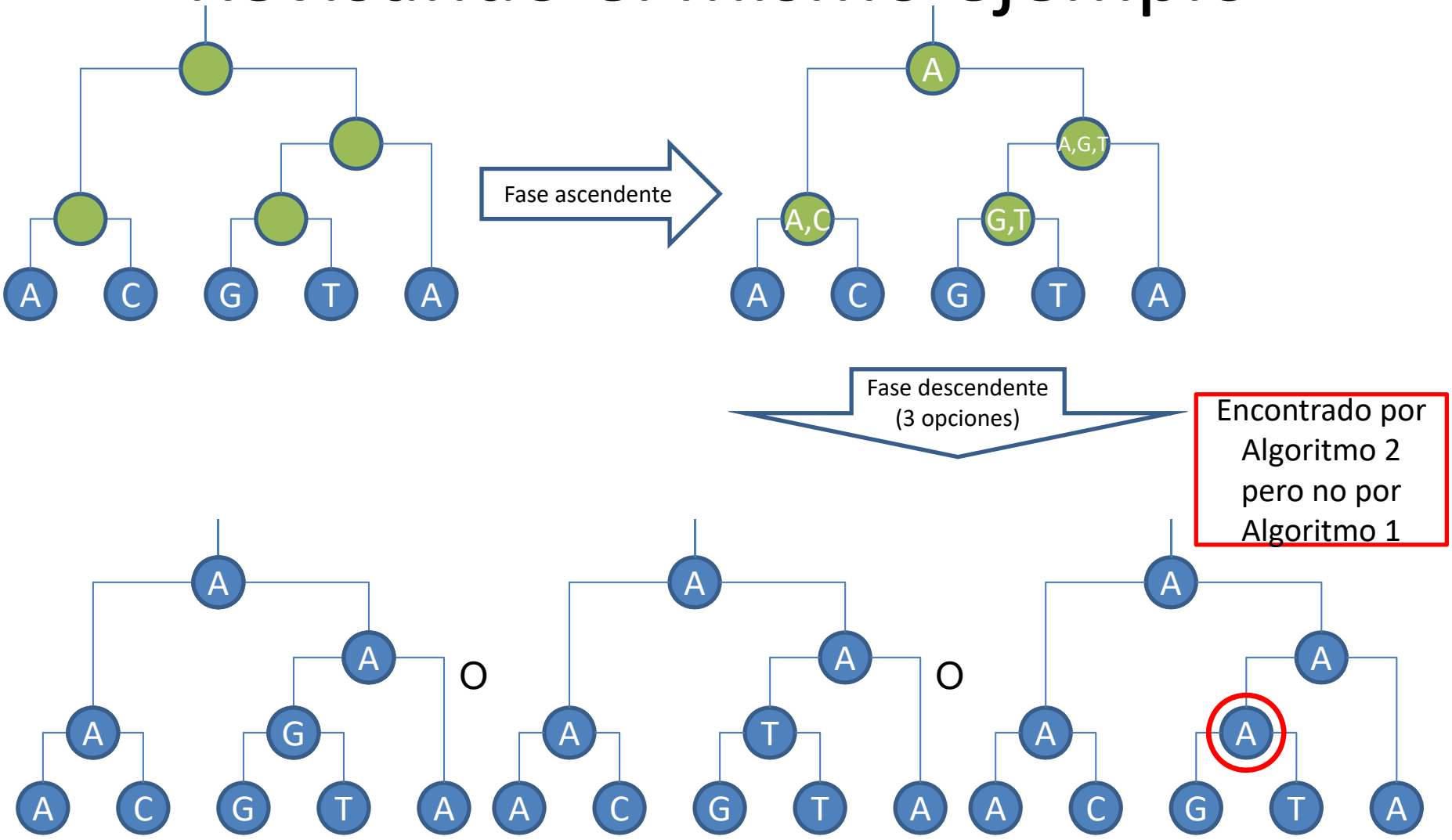
El algoritmo: versión extendida



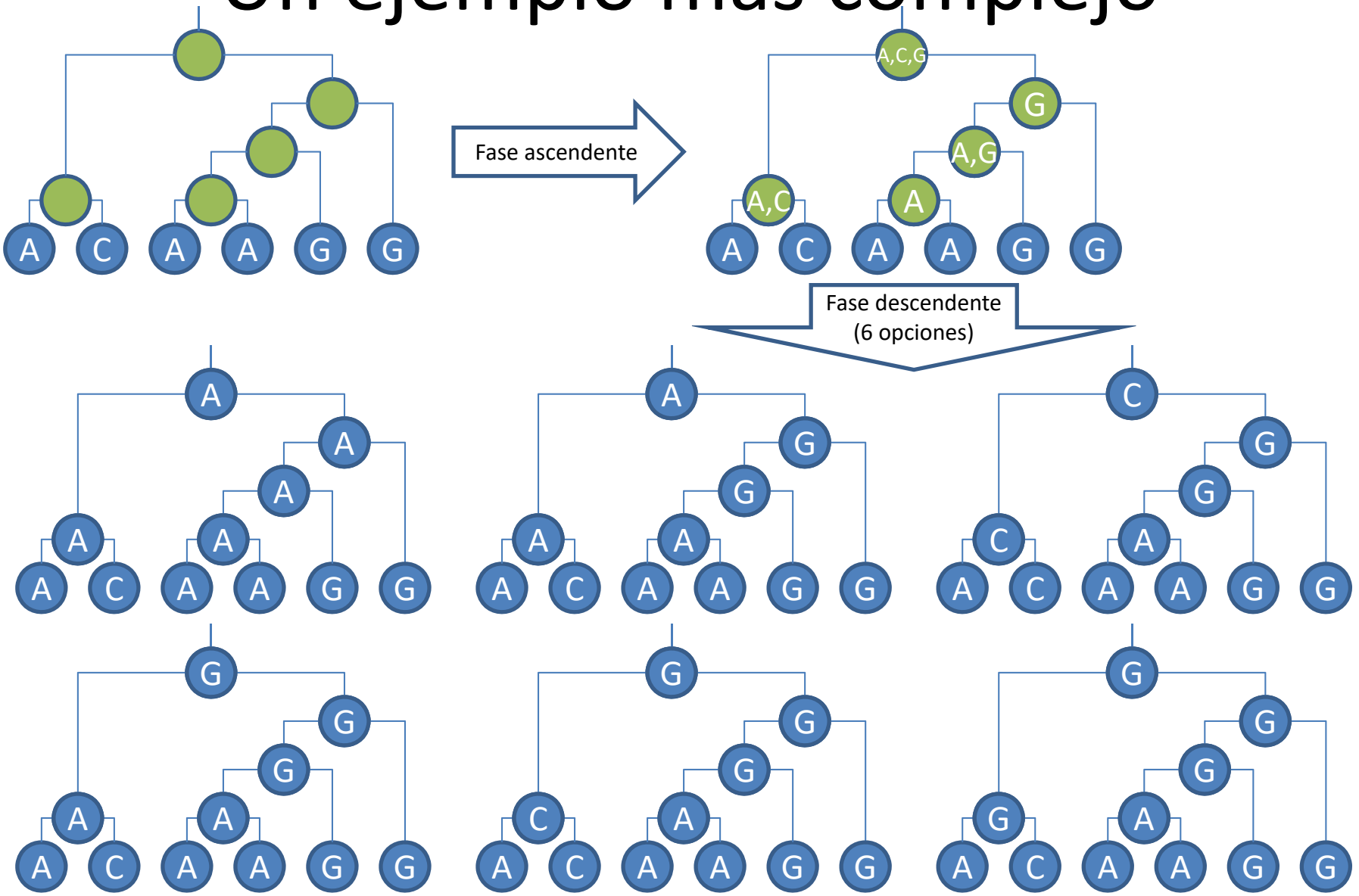
3. fase descendente: Primera selección $C_{raíz}$ desde $S_{raíz}$. Luego, para cada i otro nodo interno **(Diferente estrategia - voto mayoritario): elegiremos C_i a partir de los caracteres que existen en el mayor número de conjuntos entre $\{C_p\}$, S_l y S_r. Además, cada vez que hay múltiples opciones, elegimos una cada vez de para enumerar todas las soluciones óptimas.**

- Podemos demostrar que este algoritmo da todas las soluciones óptimas
- Un caso especial de algoritmo de programación dinámica

Revisando el mismo ejemplo



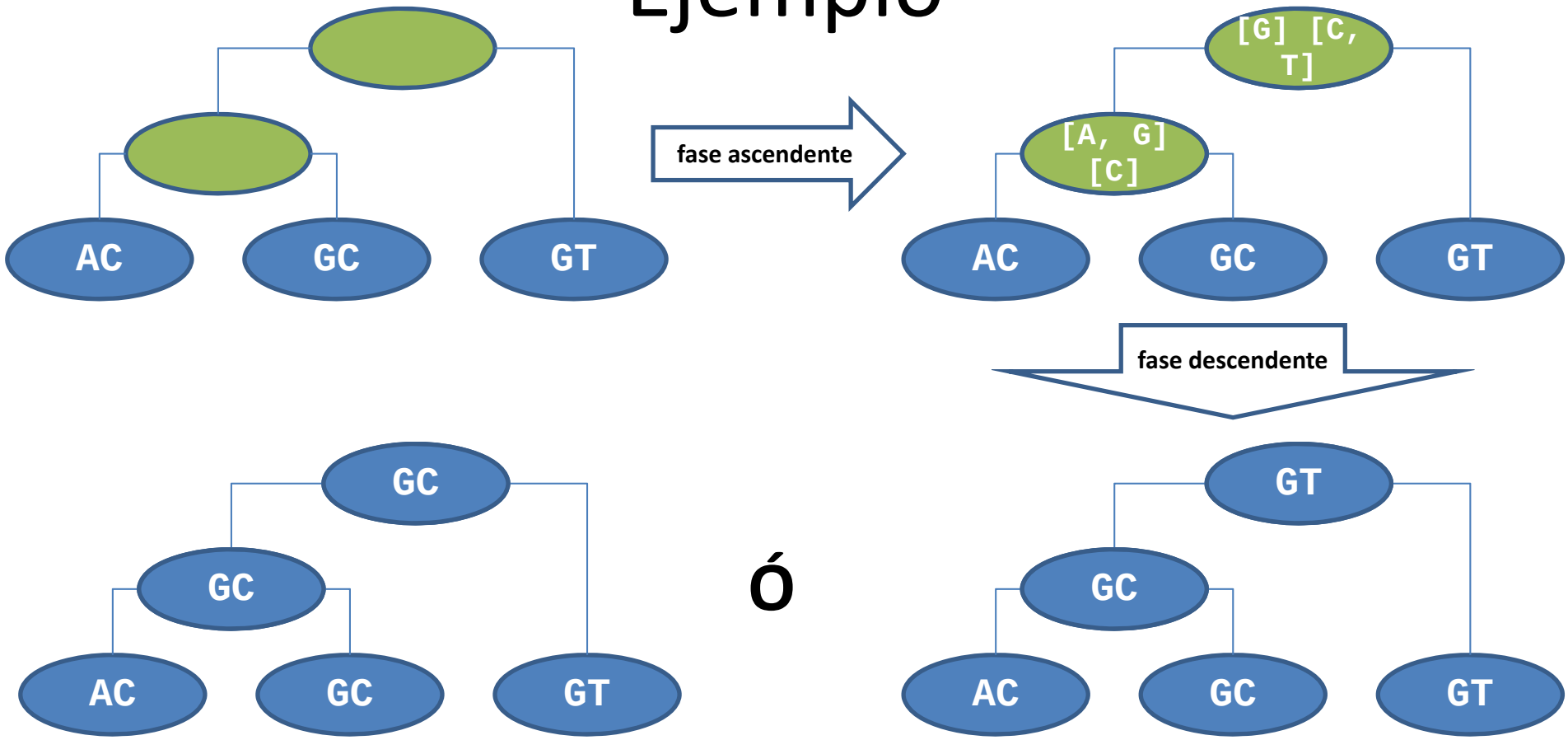
Un ejemplo más complejo



múltiples posiciones

- En una situación real, tenemos que hacer frente a secuencias que contienen más de una posición
- Simplemente aplicamos el algoritmo anterior a las diferentes posiciones de forma independiente
  - Es como suponer que posiciones diferentes mutan de forma independiente

Ejemplo



- Mínimo: 1 sustitución para la posición 1, 1 sustitución de la posición 2
- máxima parsimonia: 2 árboles que pueden alcanzar este mínimo

máxima parsimonia

- Problema restringido:
  - **Dado** un conjunto de secuencias y una **topología de árbol** con raíz
  - Encontrar las secuencias ancestrales del árbol de forma que el número total de mutaciones en el árbol sea mínimo

Resumen parsimonia

- El algoritmo de Fitch es eficiente y resuelve el problema cuando tenemos la topología
- Encontrar la **topología** requiere heurísticas complicadas
- Además para tener un resultado robusto se usa **bootstrapping**:
  - Consiste en reordenar las hojas y volver a aplicar la heurística
- Finalmente se busca un **consenso** de las topologías

HOY …

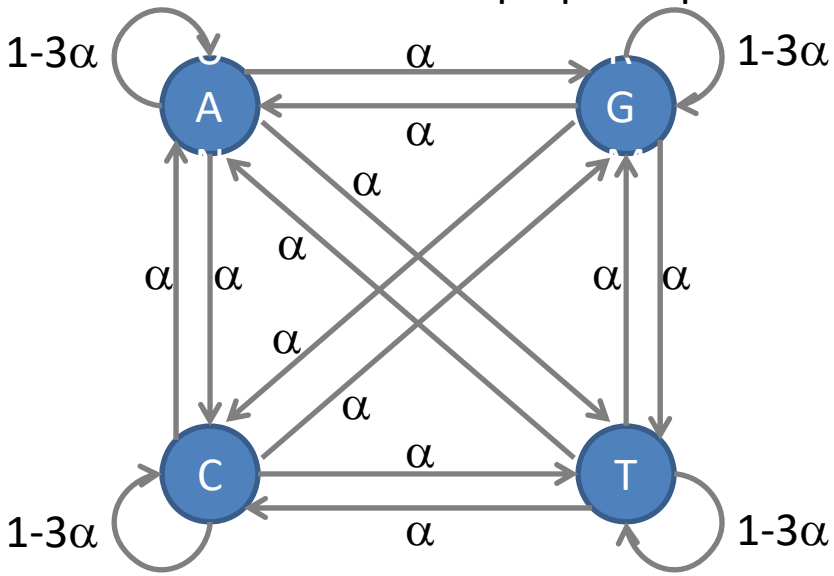
1. Distancia evolutiva y modelos de mutación
2. Árboles: Las estructuras jerárquicas relacionando diferentes objetos biológicos
    1. Formatos de archivo
    2. reconstrucción de árboles filogenéticos
    3. Métodos basados en secuencias
        - máxima parsimonia
        - **Máxima verosimilitud**
    4. métodos basados en distancias
        - UPGMA
        - Unión de vecinos

Máxima verosimilitud

- Probabilidad: Pr ($x|\theta$) es la probabilidad de producir los datos observados por un modelo determinado por sus parámetros,
  - x: Datos observados
    - Las secuencias de entrada, que se suponen alineadas
    - Una vez más, consideramos una sola posición aquí. La probabilidad para el conjunto de las secuencias es el producto de la probabilidad de las posiciones individuales, ya que se suponen independientes
  - θ: Los parámetros del modelo
- máxima verosimilitud: Encontrar el valor de θ tal que Pr ($x|\ \theta$) se maximiza

El modelo Jukes-Cantor

- Propuesto por Jukes y Cantor en 1969
- Tasa de sustitución, α, idéntica para las otras tres bases **en una unidad de tiempo**
  - Suponemos que hay como mucho una mutación por unidad de tiempo - Siempre podemos hacer la unidad más pequeña para asegurar esto



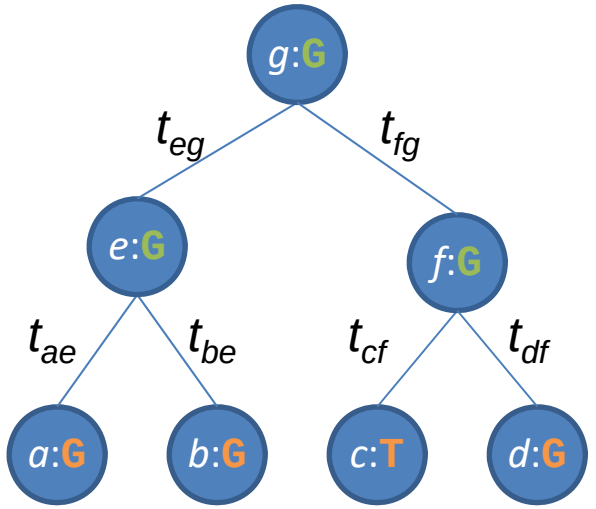
Nota: Una "sustitución" es una mutación puntual que realmente ocurrió, mientras que una falta de coincidencia en una alineación podría ser causado por una o más sustituciones

Los parámetros del modelo

- Existen diferentes posibilidades
  - En todos los casos, *x* son las secuencias de entrada
- Gran problema de verosimilitud
  - Encontrar θ: topología de árbol, las tasas de mutación y los tiempos de divergencia
  - Muy difícil
- Pequeño problema de verosimilitud
  - **Dada la topología de árbol**
  - Encontrar θ: las tasas de mutación y los tiempos de divergencia
  - Hay soluciones heurísticas eficaces que por lo general (pero no siempre) producen resultados óptimos

el cálculo de la verosimilitud

- Supongamos que se nos da lo siguiente, como se muestra en la figura:
  - topología de árbol
  - Los datos observados, x = {a:G,b:G,c:T,d:G}
  - secuencias ancestrales
  - Los parámetros, θ = {<tasas de mutación>, t_{ae},t_{be},t_{cf},t_{df},t_{eg},t_{fg}}
- Probabilidad =Pr(g:G) Pr(e:G|g:G, t_{eg}) Pr(f:G|g:G, t_{fg}) Pr(a:G|e:G, t_{ae}) Pr(b:G|e:G, t_{be}) Pr(c:T|f:G, t_{cf}) Pr(d:G|f:G, t_{df})
  - Hemos aprendido cómo calcular estas probabilidades condicionales para Jukes-Cantor



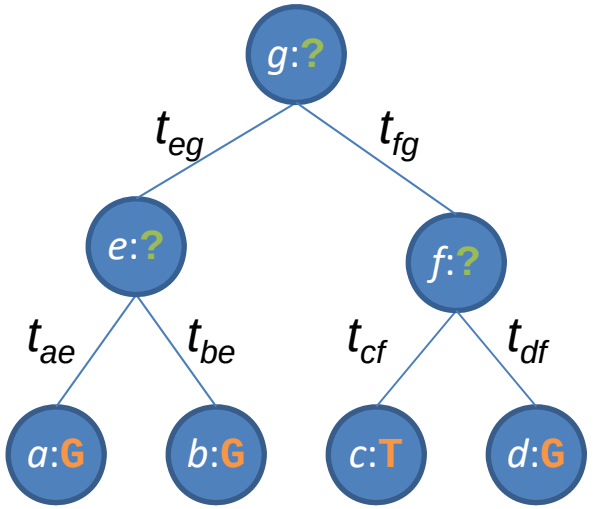
etiquetas de nodo
secuencias observadas
secuencias ancestrales

los tiempos de divergencia

el cálculo de la verosimilitud

- En el pequeño problema de verosimilitud, que sólo da la topología del árbol, pero no los estados ancestrales
- Necesidad de probarlos todos (Suma de $4^3 = 64$ términos) :probabilidad =

Pr(g:A)
Pr(e:A|g:A, t_{eg}) Pr(f:A|g:A, t_{fg})
Pr(a:G|e:A, t_{ae}) Pr(b:G|e:A, t_{be})
Pr(c:T|f:A, t_{cf}) Pr(d:G|f:A, t_{df})
+
Pr(g:C)
Pr(e:A|g:C, t_{eg}) Pr(f:A|g:C, t_{fg})
Pr(a:G|e:A, t_{ae}) Pr(b:G|e:A, t_{be})
Pr(c:T|f:A, t_{cf}) Pr(d:G|f:A, t_{df})
+
...
+
Pr(g:T)
Pr(e:T|g:T, t_{eg}) Pr(f:T|g:T, t_{fg})
Pr(a:G|e:T, t_{ae}) Pr(b:G|e:T, t_{be})
Pr(c:T|f:T, t_{cf}) Pr(d:G|f:T, t_{df})



Posibles estados ancestrales:

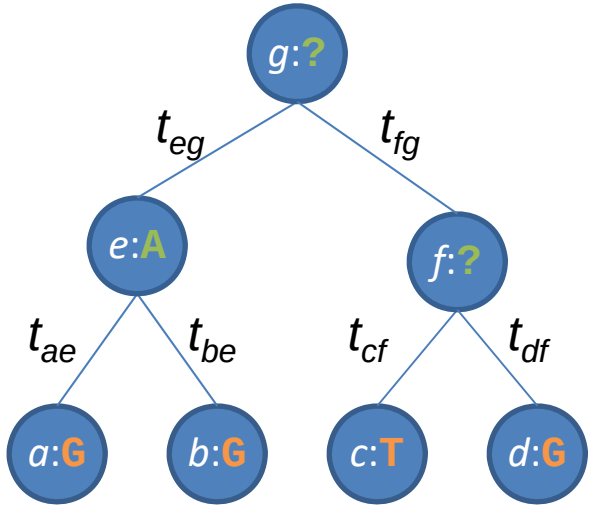
e	A	A	A	A	A		T
f	A	A	A	A	C	...	T
g	A	C	G	T	A		T

eficiencia computacional

- ¿Tiempo de cálculo necesario?
- Para nuestro ejemplo:
  - 3 nodos internos $\Rightarrow 4^3 = 64$ posibles conjuntos de estados ancestrales
  - Para cada conjunto de estados ancestrales, necesitamos multiplicar 7 términos (porque hay 7 nodos en el árbol)
- En general:
  - Si hay n secuencias de entrada, hay n-1 Nodos internos $\Rightarrow 4^{n-1}$ posibles conjuntos de estados ancestrales
  - Para cada conjunto de estados ancestrales, necesitamos multiplicar $n+n-1 = 2n-1$ términos
- Es poco práctico llevar a cabo este número exponencial de operaciones - A continuación, la forma de resolver el problema …
  - **¡Programación dinámica!**

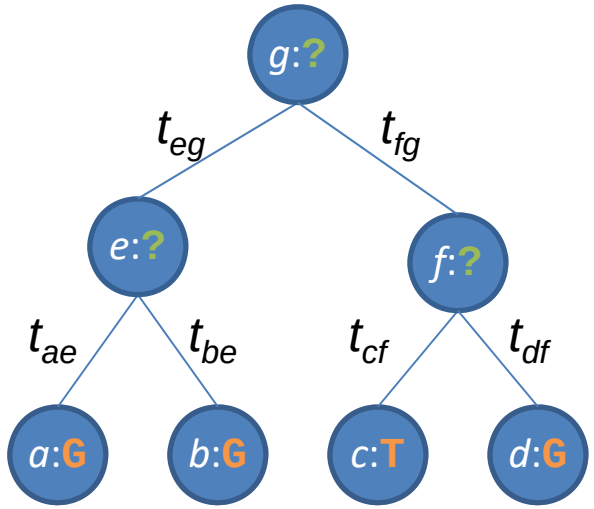
El cálculo de la verosimilitud de manera eficiente

- Una observación importante: una vez que se determina la raíz de un sub-árbol, la probabilidad de este sub-árbol no depende de otros nodos en el árbol entero
- Por ejemplo, una vez que el nodo *e* se decide a tomar carácter A, La verosimilitud de el sub-árbol de nodos *a*,*b* y *e* es $\Pr(e{:}\mathrm{A}|g, t_{eg})$ $\Pr(a{:}\mathrm{G}|e{:}\mathrm{A}, t_{ae})\Pr(b{:}\mathrm{G}|e{:}\mathrm{A}, t_{be})$
  - Si el carácter en el nodo *g* no cambia, el valor de la expresión anterior no cambiará sin importar qué carácter toma *f*
  - Por lo tanto este valor puede ser reutilizado



El cálculo de la verosimilitud de manera eficiente

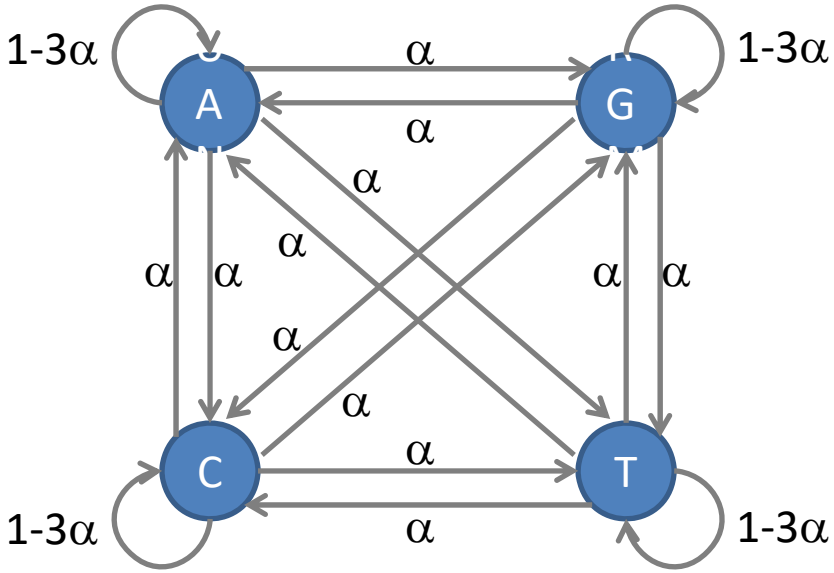
- Definir la tabla V, donde la entrada V (*i*,*c*) es la verosimilitud del subárbol con raíz *i* cuando el **padre** de *i* tiene carácter *c*
  - verosimilitud =
    Pr(*g*:A) V(*e*,A) V(*f*,A) +
    Pr(*g*:C) V(*e*,C) V(*f*,C) +
    Pr(*g*:G) V(*e*,G) V(*f*,G) +
    Pr(*g*:T) V(*e*,T) V(*f*,T)
  - V(*e*, A) =
    Pr(*e*:A|*g*:A,t_{eg}) V(*a*,A) V(*b*,A) +
    Pr(*e*:C|*g*:A,t_{eg}) V(*a*,C) V(*b*,C) +
    Pr(*e*:G|*g*:A,t_{eg}) V(*a*,G) V(*b*,G) +
    Pr(*e*:T|*g*:A,t_{eg}) V(*a*,T) V(*b*,T)
  - V(*a*,A) = Pr(*a*:G|*e*:A, t_{ae})
  - V(*a*,C) = Pr(*a*:G|*e*:C, t_{ae})
  - ...



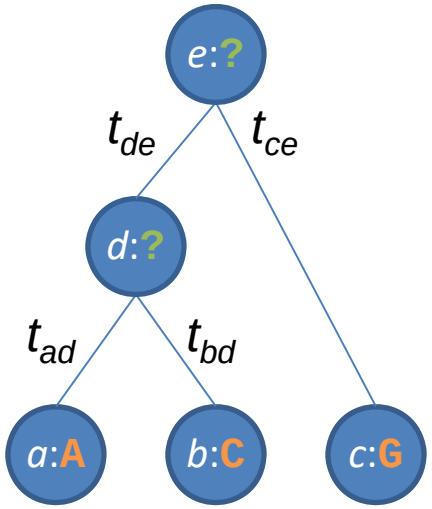
- La tabla V contiene O (*n*) entradas. Calcular el valor de cada entrada requiere un número constante de operaciones ⇒ tiempo lineal

Ejemplo con Jukes-Cantor

- Tasa de sustitución, α, idéntica para las otras tres bases **en una unidad de tiempo**



Ejemplo



- Asumamos:
  - Las cuatro bases son igualmente probables en la raíz
  - La mutación modelo de Jukes-Cantor es correcta
  - tasa de mutación por unidad de tiempo, α = 0,1
  - Cada rama del árbol representa una unidad de tiempo
- V (*i*,*x*) es la verosimilitud del subárbol con raíz *i* cuando el **padre** de *i* tiene carácter *x*

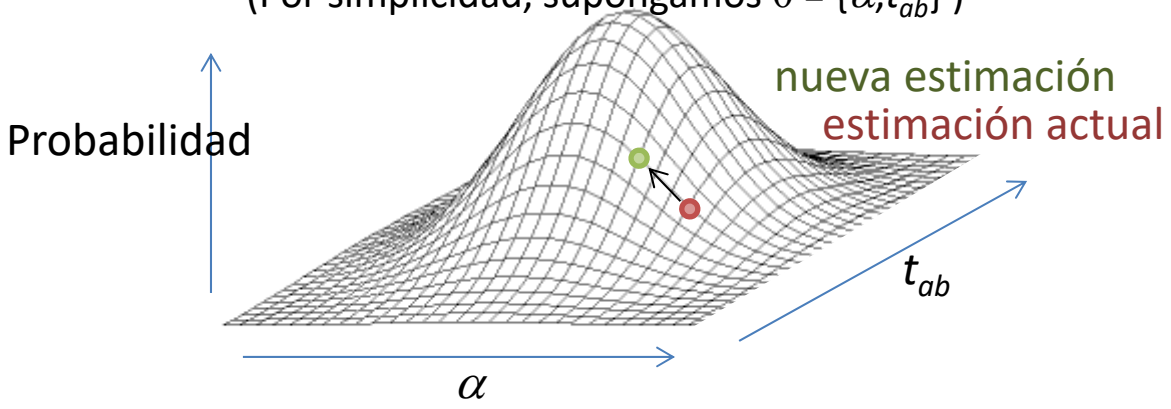
V(*i*, *x*)	*x*=A	*x*=C	*x*=G	*x*=T
i=*a*	0.7	0.1	0.1	0.1
i=*b*	0.1	0.7	0.1	0.1
i=*c*	0.1	0.1	0.7	0.1
i=*d*	0.7(0.7)(0.1) + 0.1(0.1)(0.7) + 0.1(0.1)(0.1) + 0.1(0.1)(0.1) = 0.058	0.1(0.7)(0.1) + 0.7(0.1)(0.7) + 0.1(0.1)(0.1) + 0.1(0.1)(0.1) = 0.058	0.1(0.7)(0.1) + 0.1(0.1)(0.7) + 0.7(0.1)(0.1) + 0.1(0.1)(0.1) = 0.022	0.1(0.7)(0.1) + 0.1(0.1)(0.7) + 0.1(0.1)(0.1) + 0.7(0.1)(0.1) = 0.022

- verosimilitud total: 0,25 (0,058) (0,1) + 0,25 (0,058) (0,1) + 0,25 (0,022) (0,7) + 0,25 (0,022) (0,1) = 0,0073

La solución del problema pequeño

- Entonces, ¿cómo encontrar los valores óptimos de los parámetros?
  - Comience con una estimación al azar θ
  - Aplicar un algoritmo "**hill climbing**"
    - Cambiar el valor de un parámetro de manera que se incremente la probabilidad
    - Repetirlo para cada parámetro, a su vez, por múltiples iteraciones
    - Alcanzará máximo si hay un solo "pico" - Esto es cierto en muchas situaciones reales, aunque se pueden construir casos en los que no es cierto

(Por simplicidad, supongamos $\theta = \{\alpha, t_{ab}\}$)



Fuente de la imagen: http://www.absoluteastronomy.com/topics/Hill_climbing

Resumen de máx. veroximilitud

- Dada la topología del árbol, las tasas de mutación y los tiempos de divergencia
  - Es eficiente calcular la verosimilitud
- Dada sólo la topología:
  - Heurística de hill climbing para encontrar tasas de mutación y tiempos
- Encontrar la topología requiere más heurísticas y bootstrapping