第五章：序列比对与分析

浙江大学　陈飘飘

本章内容

章节结构

- **第一节：序列特征解析**
- **第二节：序列比对和分析**
- **第三节：分子演化树构建**
- **第四节：讨论与展望**

为什么要学习序列比对?

完成一个人的全基因组测序，可获得约 **30 亿**个碱基。若要从中找出基因突变，**第一步必须将测得的片段比对（alignment）到参考基因组上。**

不论是：

- 全基因组测序（WGS）：用于寻找变异
- 外显子测序（WES）：用于寻找致病突变
- 转录组测序（RNA-seq）：用于检测基因表达
- 三维基因组（Hi-C）：用于解析染色体空间结构

都离不开“比对”这一步！

第一节：序列特征解析

生物分子的表示

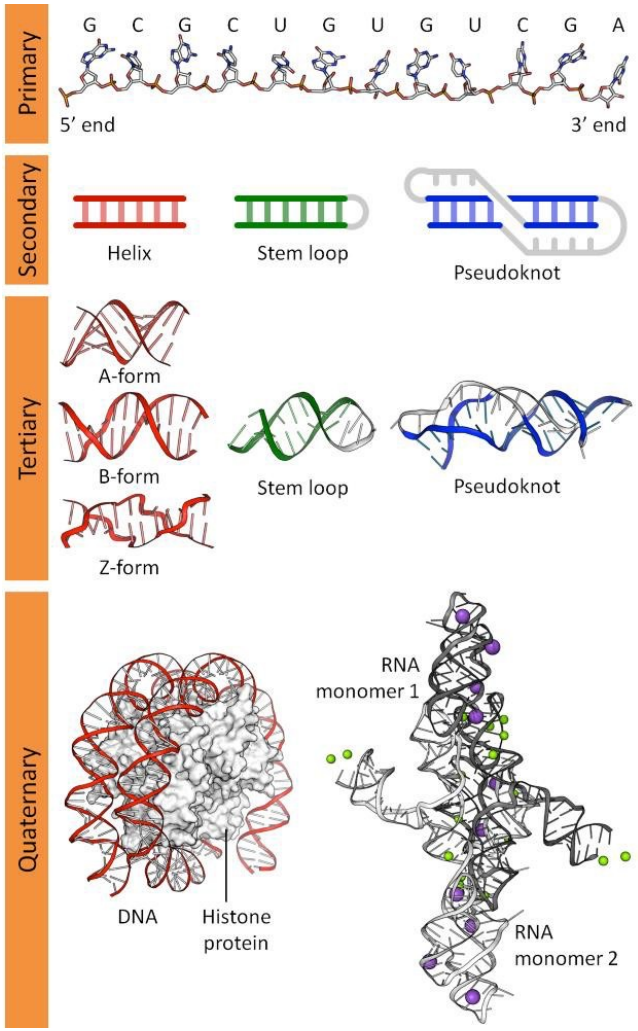
核酸分子

一级结构

二级结构

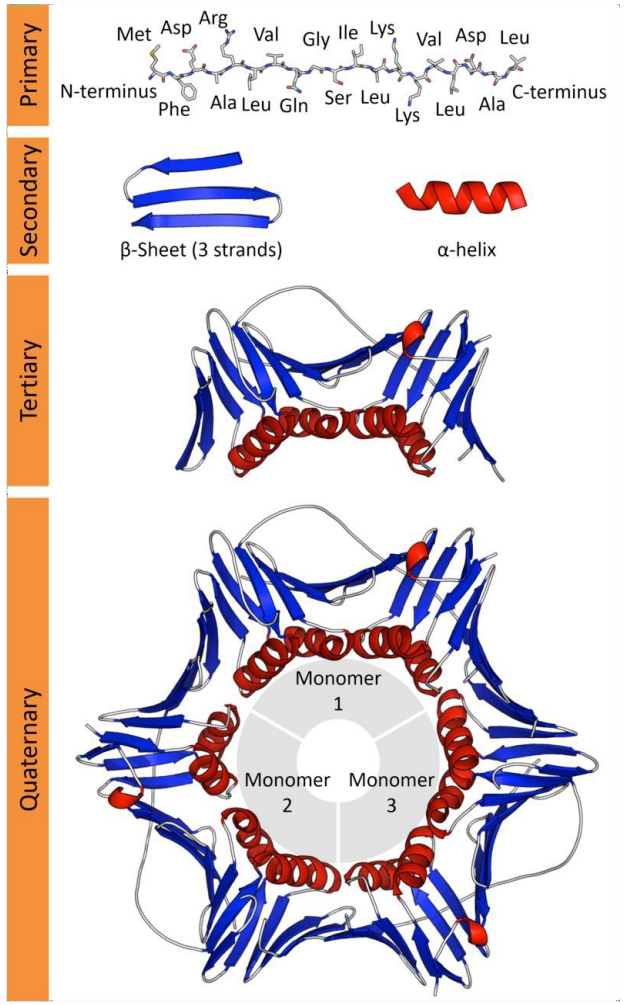
三级结构

四级结构



(Image from: https://upload.wikimedia.org/wikipedia/commons/d/da/DNA_RNA_structure_%28full%29.png)

蛋白质分子



(Modified from: https://commons.wikimedia.org/wiki/Template:Other_versions/Protein_structure_(full)#/media/File:Protein_structure_(full).png)

序列比对与分析/序列特征解析

程序或软件名	描述
整合序列分析工具	
BioEdit	用于分析、编辑和处理 DNA 序列数据的生物信息学软件
EMBOSS	综合在线分析软件包
DNAMAN	LynnonBiosoft 公司开发的高度集成化的 DNA 序列编辑软件
DNASTAR	基于 Windows 和 Macintosh 平台的序列分析软件
序列变换	
REVSEQ	EMBOSS 软件包中的序列变换程序之一
Reverse Complement	Sequence Manipulation Suite(SMS)中的序列变换程序
限制性内切酶位点分析	
REBASE	限制性内切酶数据库
NEBcutter	限制性内酶切位点分析工具，整合 REBASE
WebCutter	限制性内酶切位点分析工具，支持线性和环状 DNA 序列分析以及寻找沉默诱变位点
RestrictionMapper	限制性内切酶切位点分析工具，支持线性和环状 DNA 序列分析
重复序列分析	
RepBase	真核生物转座子和重复序列数据库
STRBase	短串联重复序列(STR)数据库
RepeatMasker	散布重复和低复杂性重复序列分析工具，使用 RepBase 和 Dfam 重复序列数据库
CENSOR	使用 RepBase 查找重复序列
Tandem Repeats Finder	串联重复序列分析工具

序列比对与分析/序列特征解析

我们知道，DNA 序列里其实隐藏着很多“信号”，比如外显子(exon)、内含子(intron)、剪接位点等等。

那问题是——我们怎么在一串看似随机的 A、T、G、C 里找到这些特征呢?

一种常用的方法叫 **隐马尔可夫模型（Hidden Markov Model，HMM）**

“看结果猜过程”

序列比对与分析/序列特征解析

具体来说，根据生物学知识，我们猜测不同位置有不同的统计学特性，

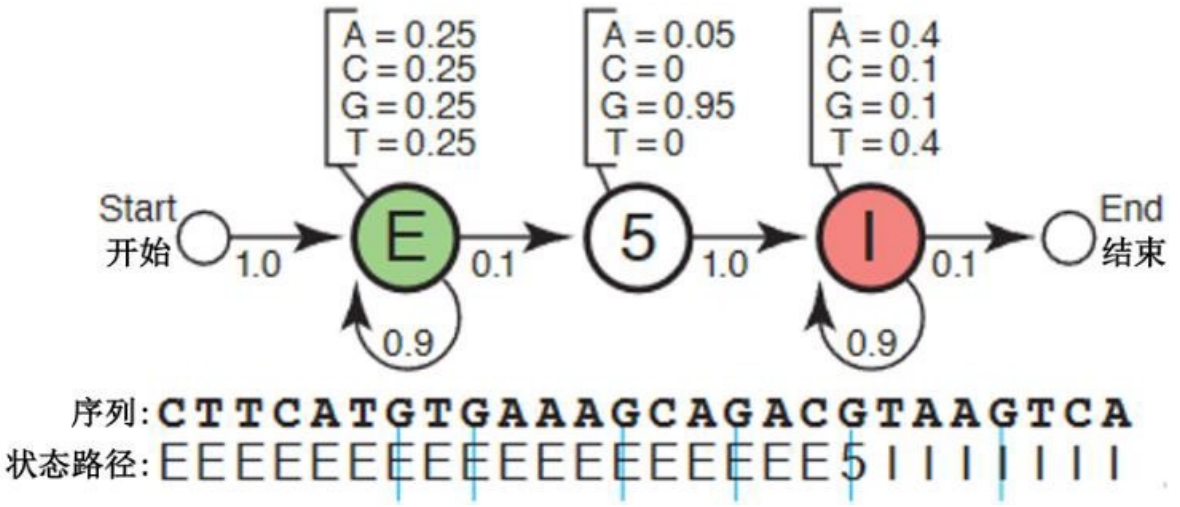
比如:

外显子平均碱基组成较均匀(每个碱基25%)

内含子富含A / T (A / T 各 40%，C / G 各10%)

5′ Spice site 区域核苷酸几乎总是G (95%G 和 5%A)

接下来我们根据上述假设构造HMM:



序列比对与分析/序列特征解析

$$P(S, \pi \mid HMM, \theta) = P(\pi \mid HMM, \theta) \times P(S \mid \pi, HMM, \theta)$$

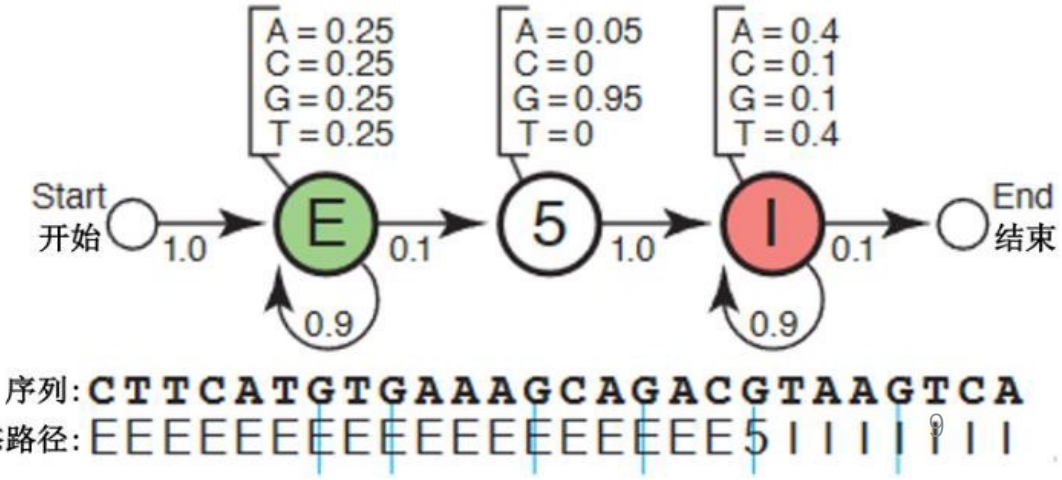
符号含义

- S：观测到的序列（比如 DNA 碱基序列 CTT…）
- π：对应的状态路径（比如 EEEEEEIIII…，表示外显子/内含子等状态序列）
- θ：模型的参数

状态转移概率：从一个状态走向另一个状态的可能性

（如从 E 转到 E、E 转到 I 的可能性）

发射概率：在 E 状态下生成 C/T/A/G 的概率，在 I 状态下生成 A/T 的概率更大。



序列比对与分析/序列特征解析

假设我们要预测 DNA 序列 **AT** 是否来自 “外显子 (E)”还是 “内含子 (I)”

模型参数**θ**：

1. 状态转移概率 (Transition)
   - P(E→E) = 0.9，P(E→I) = 0.1
   - P(I→I) = 0.8，P(I→E) = 0.2
2. 发射概率 (Emission)
   - 在 E 状态：A=0.25, T=0.25, C=0.25, G=0.25 （均匀分布）
   - 在 I 状态：A=0.4, T=0.4, C=0.1, G=0.1 （富含 A/T）
3. 观测序列： S= AT

$$P(S, \pi \mid HMM, \theta) = P(\pi) \times P(S|\pi)$$

假设状态路径 **π = EE**（即两个碱基均来自外显子）
初始进入 E 状态的概率（假设=**1**）
第一个碱基 A 来自 E：**P=0.25**
状态转移 E→E：**P=0.9**
第二个碱基 T 来自 E：**P=0.25**

$$P(S, \pi) = 1 \times 0.25 \times 0.9 \times 0.25 = 0.05625$$

取对数

$$\log P(S, \pi) = \log(0.05625) \approx -2.88$$

序列比对与分析/序列特征解析

假设我们要预测 DNA 序列 **AT** 是否来自 “外显子 (E)”还是 “内含子 (I)”

模型参数**θ**：

1. 状态转移概率 (Transition)
   - P(E→E) = 0.9，P(E→I) = 0.1
   - P(I→I) = 0.8，P(I→E) = 0.2
2. 发射概率 (Emission)
   - 在 E 状态：A=0.25, T=0.25, C=0.25, G=0.25 （均匀分布）
   - 在 I 状态：A=0.4, T=0.4, C=0.1, G=0.1 （富含 A/T）
3. 观测序列： S= AT

$$P(S,\pi \mid HMM,\theta) = P(\pi) \times P(S|\pi)$$

假设状态路径 **π = II**（即两个碱基均来自内含子）
初始进入 I 状态的概率（假设=**1**）
第一个碱基 A 来自 I：**P=0.4**
状态转移 I→I：**P=0.8**
第二个碱基 T 来自 I：**P=0.4**

$$P(S,\pi) = 1 \times 0.4 \times 0.8 \times 0.4 = 0.128$$

$$\log P(S,\pi) \approx -2.05$$

序列比对与分析/序列特征解析

假设我们要预测 DNA 序列 **AT** 是否来自“外显子 (E)”还是“内含子 (I)”

结论

- 序列 AT 如果来自 **EE**，log P = -2.88
- 序列 AT 如果来自 **II**， log P = -2.05
- 因为 -2.05 > -2.88，说明在这个模型下，**序列 AT 更可能来自内含子 (I)**

序列比对与分析/序列特征解析

- 1997 年，MIT 的 Burge 和 Karlin 开发了 **GenScan**，这是一个基于广义 HMM 的人类及脊椎动物基因预测软件。

- GenScan 能够识别序列中的多种统计特征：
  - 密码子使用频率
  - 外显子 / 内含子边界信号
  - 启动子信号
  - 起始 / 终止区域

- GeneScan 把这些信息结合在一起，提供全面的基因结构预测。

序列比对与分析/序列特征解析

近年来，随着数据的积累与算力的提升，**支持向量机(SVM)、随机森林或神经网络**等机器学习方法也被广泛应用于基因预测。与早期的显式建模不同，这些方法通过对已知的基因组注释、转录组测序数据等训练数据的学习以“自动” 识别出序列中的特定模式，并用这些模式来预测新序列中的基因。

这些方法虽然强大，但也有不足：

- 基因组特征在不同物种间存在差异（例如 AT/GC 含量差异很大）。
- 在未知物种上“从头预测”时，往往需要事先设定物种相关参数。

因此，未来趋势是结合 **大规模数据训练** 与 **物种特异性知识**，进一步提高预测准确性。

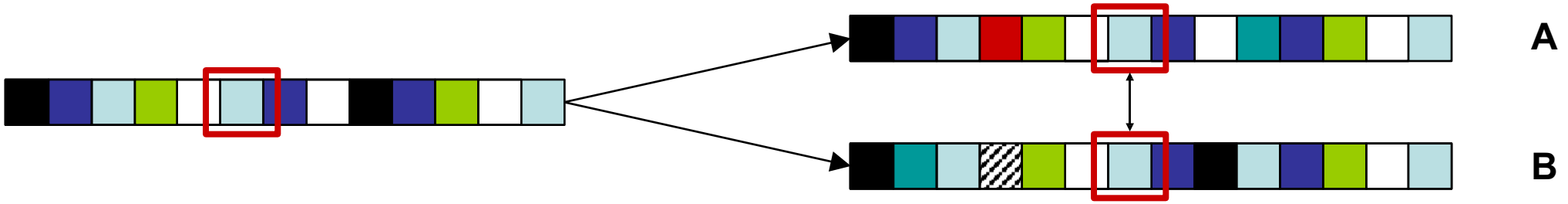
第二节：序列比对和分析

本章导语

- 研究序列关系的核心是：**定量衡量序列间的相似性和差异性。**
- 在这里要区分几个重要概念：
  - **同源 (Homologous)**：一个演化概念，表示两个序列来源于共同的祖先。
  - **相似 (Similarity)**：序列上碱基或氨基酸的相似程度，常用比对方法测定。
- 在实践中：
  - 我们通常通过“序列相似”来推断“同源关系”；
  - 根据“序列-结构-功能”的链条，**相似序列通常意味着功能也相似。**
- 因此，已知序列的功能常常可以帮助我们推断未知序列的功能。

序列比对 Sequence Alignment: in Biology

- 序列比对的目的就是把不同序列对齐，把功能或进化上相对应的位置排在一起。
- **核心思想**：比对不是简单地把字母排好，而是根据 **功能关系** 或 **进化关系** 来对齐。



序列比对　Sequence Alignment: in Math

- **输入 (Input)**：
  - 两条或多条序列（*S1*, *S2*, …, *Sn*）
  - 以及一个**打分函数** *f*
- **输出 (Output)**：
  - 找到一个比对结果，使得它的分数 **最高 (optimal score)**

$$\text{argmax}_{ali} f(ali(S_1, S_2, \ldots, S_n))$$

打分规则　Scoring Alignment

```
GAATC      GAAT-C      -GAAT-C
CATAC      C-ATAC      C-A-TAC
```

替代
(substitution)

```
GAATC-     GAAT-C      GA-ATC
CA-TAC     CA-TAC      CATA-C
```

插入/缺失
(indel)

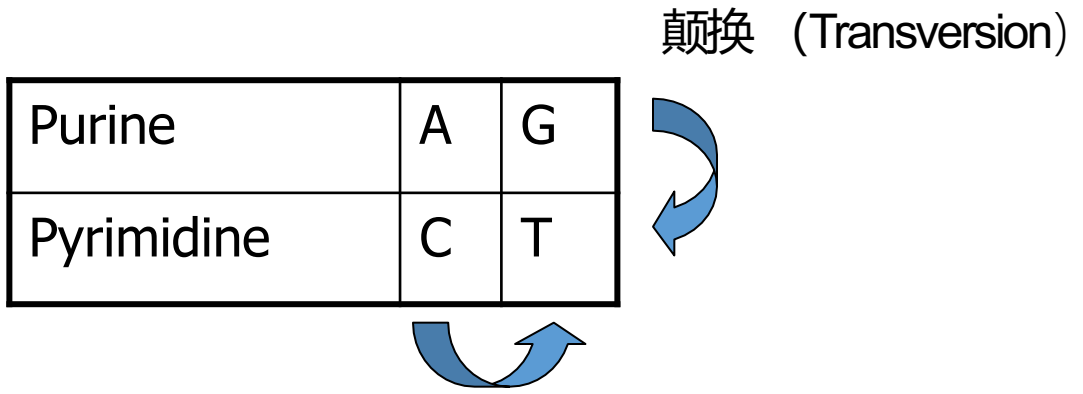
- Scoring function: measure the **quality** of a given alignment.
  - 替换矩阵（Scoring matrix）：同一个位置的碱基/氨基酸比对时得到的分值
  - 缺口惩罚（Gap penalty）：在比对中引入空位（gap），要扣分

打分矩阵 Scoring Matrix

- Measure the likelihood of a given substitution happened **in the real world**.
  - Substitutions that are **more likely** should get a **higher** score
  - Substitutions that are **less likely** should get a **lower** score

- Scoring Matrices are designed to **detect signal above background**, i.e. to detect similarities beyond what would be observed **by chance alone**

核苷酸打分矩阵 Nucleotide Scoring Matrix



GAATC

CATCC

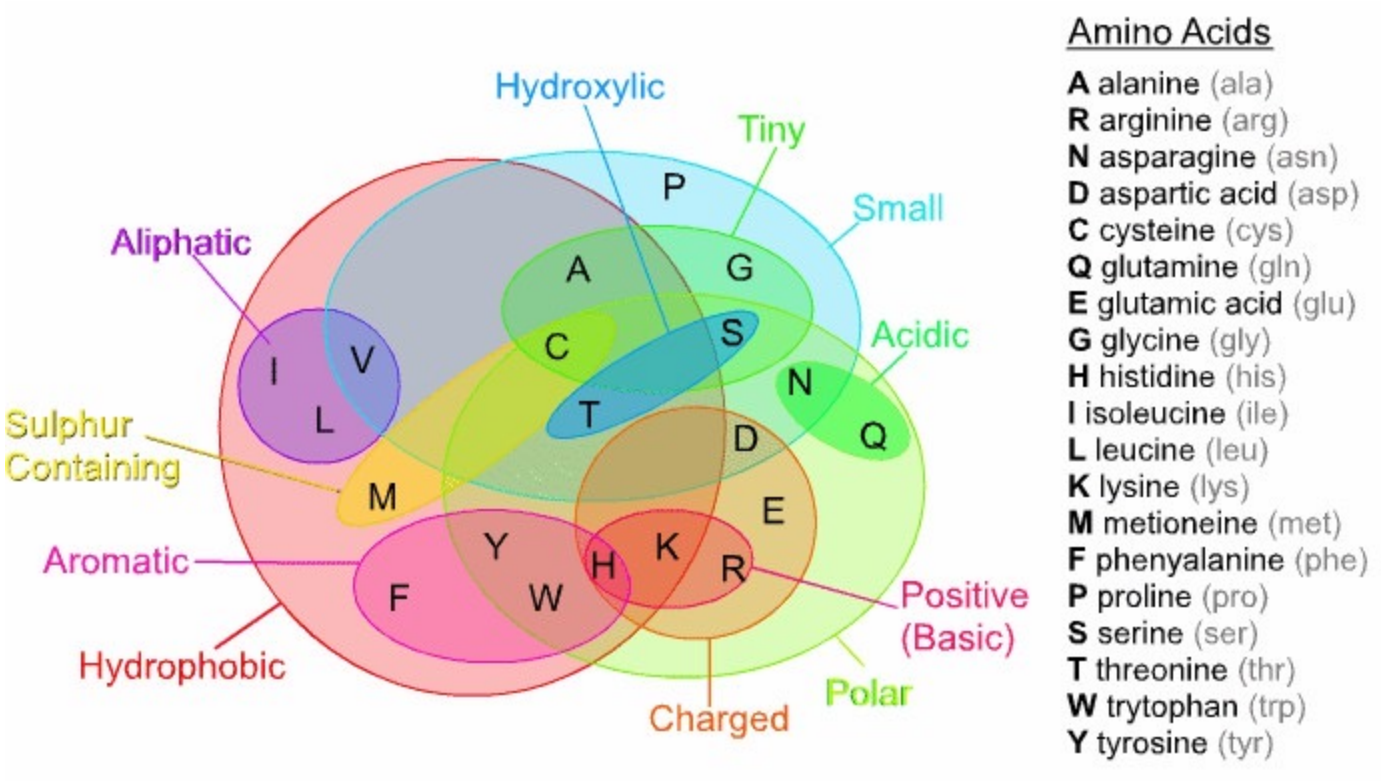
-7 + 2 + (-7) + (-5) + 2 = **-15**

A hypothetical **substitution matrix**:

	A	C	G	T
A	2	−7	−5	−7
C	−7	2	−7	−5
G	−5	−7	2	−7
T	−7	−5	−7	2

氨基酸打分矩阵 Amino acid Scoring Matrix

不同氨基酸有不同物理化学性质（疏水性、带电性、极性等）



(Adopted from Prof. Jingchu Luo)

PAM Matrix

- PAM = Point Accepted Mutation
- Developed by Dayhoff et al. 1978.

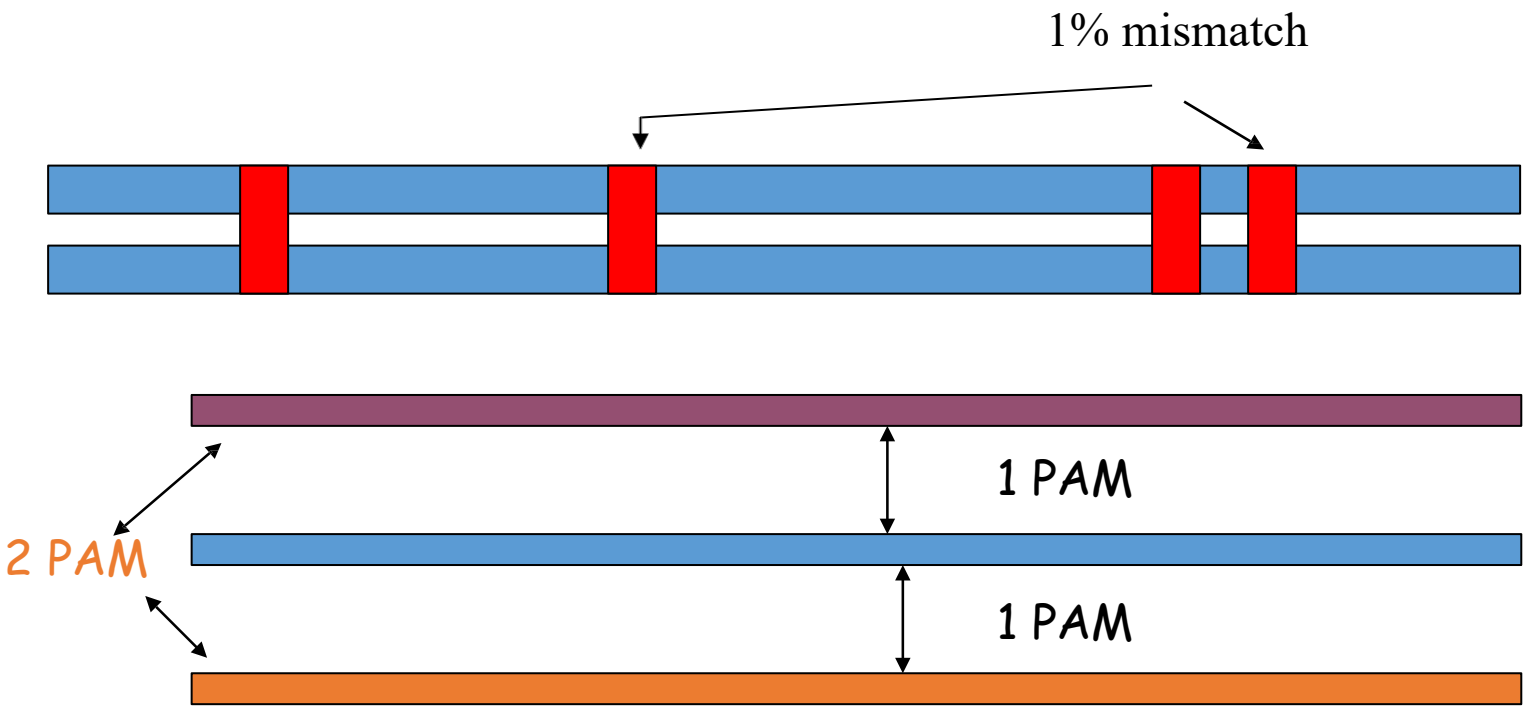
Dayhoff 及其同事研究了**71组**进化高度相关的蛋白质中的**1572种**变化。统计了其中氨基酸的突变数据

	A Ala	R Arg	N Asn	D Asp	C Cys	Q Gln	E Glu	G Gly	H His	I Ile	L Leu	K Lys	M Met	F Phe	P Pro	S Ser	T Thr	W Trp	Y Tyr	V Val
A																				
R	30																			
N	109	17																		
D	154	0	532																	
C	33	10	0	0																
Q	93	120	50	76	0															
E	266	0	94	831	0	422														
G	579	10	156	162	10	30	112													
H	21	103	226	43	10	243	23	10												
I	66	30	36	13	17	8	35	0	3											
L	95	17	37	0	y	75	15	17	40	253										
K	57	477	322	85	0	147	104	60	23	43	39									
M	29	17	0	0	0	20	7	7	0	57	207	90								
F	20	7	7	0	0	0	0	17	20	90	167	0	17							
P	345	67	27	10	10	93	40	49	50	7	43	43	4	7						
S	772	137	432	98	117	47	86	450	26	20	32	168	20	40	269					
T	590	20	169	57	10	37	31	50	14	129	52	200	28	10	73	696				
W	0	27	3	0	0	0	0	0	3	0	13	0	0	10	0	17	0			
Y	20	3	36	0	30	0	10	0	40	13	23	10	0	260	0	22	23	6		
V	365	20	13	17	33	27	37	97	30	661	303	17	77	10	50	43	186	0	17	
	A Ala	R Arg	N Asn	D Asp	C Cys	Q Gln	E Glu	G Gly	H His	I Ile	L Leu	K Lys	M Met	F Phe	P Pro	S Ser	T Thr	W Trp	Y Tyr	V Val

Source: Dayhoff (1972). Reproduced with permission from National Biomedical Research Foundation.

PAM Matrix

进化距离为 1 PAM： 一段序列在 “1 PAM” 演化距离上，每 100 个残基中**平均有 1 个被接受的替换**



- **1 PAM = one step of evolution**
- **PAM_2 = two step of evolution = PAM_1 * PAM_1**
- **PAM_{250}**
  - **= PAM_1*PAM_{249}**
  - **= PAM_1^{250}**

BLOSUM：基于保守片段的经验统计矩阵

- **BLOSUM = BLOcks SUbstitution Matrix（developed by Henikoff）**
  - 从数据库取**局部保守片段**（不依赖全局比对）
  - 为避免近缘序列重复计数，对**序列相似度 ≥ 阈值 T** 的序列先**聚类**并降权（例如 **BLOSUM62**：把 ≥62% 身份度的序列聚为一类）。
  - 在各对齐位点统计氨基酸对的**观测频率** f_{ij}，与背景频率 $p_i p_j$比较

$$S_{ij} = \log \frac{f_{ij}}{p_i p_j}$$

  - BLOSUM**数值大** = **距离近**（BLOSUM80 近 > BLOSUM62 > BLOSUM45 远）
  - “在真实同源比对里看到 i–j 这对的概率 / 在随机背景下碰到这对的概率”比值 > 1 → log 为正 →它们比随机更常见→**给正分**
  - 比值 < 1 → log 为负 →它们比随机更罕见→**给负分**

两种 scoring matrix 的比较

PAM250

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	12																			
S	0	2																		
T	−2	1	3																	
P	−3	1	0	6																
A	−2	1	1	1	2															
G	−3	1	0	−1	1	5														
N	−4	1	0	−1	0	0	2													
D	−5	0	0	−1	0	1	2	4												
E	−5	0	0	−1	0	0	1	3	4											
Q	−5	−1	−1	0	0	−1	1	2	2	4										
H	−3	−1	−1	0	−1	−2	2	1	1	3	6									
R	−4	0	−1	0	−2	−3	0	−1	−1	1	2	6								
K	−5	0	0	−1	−1	−2	1	0	0	1	0	3	5							
M	−5	−2	−1	−2	−1	−3	−2	−3	−2	−1	−2	0	0	6						
I	−2	−1	0	−2	−1	−3	−2	−2	−2	−2	−2	−2	−2	2	5					
L	−6	−3	−2	−3	−2	−4	−3	−4	−3	−2	−2	−3	−3	4	2	6				
V	−2	−1	0	−1	0	−1	−2	−2	−2	−2	−2	−2	−2	2	4	2	4			
F	−4	−3	−3	−5	−4	−5	−4	−6	−5	−5	−2	−4	−5	0	1	2	−1	9		
Y	0	−3	−3	−5	−3	−5	−2	−4	−4	−4	0	−4	−4	−2	−1	−1	−2	7	10	
W	−8	−2	−5	−6	−6	−7	−4	−7	−7	−5	−3	2	−3	−4	−5	−2	−6	0	0	17
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W

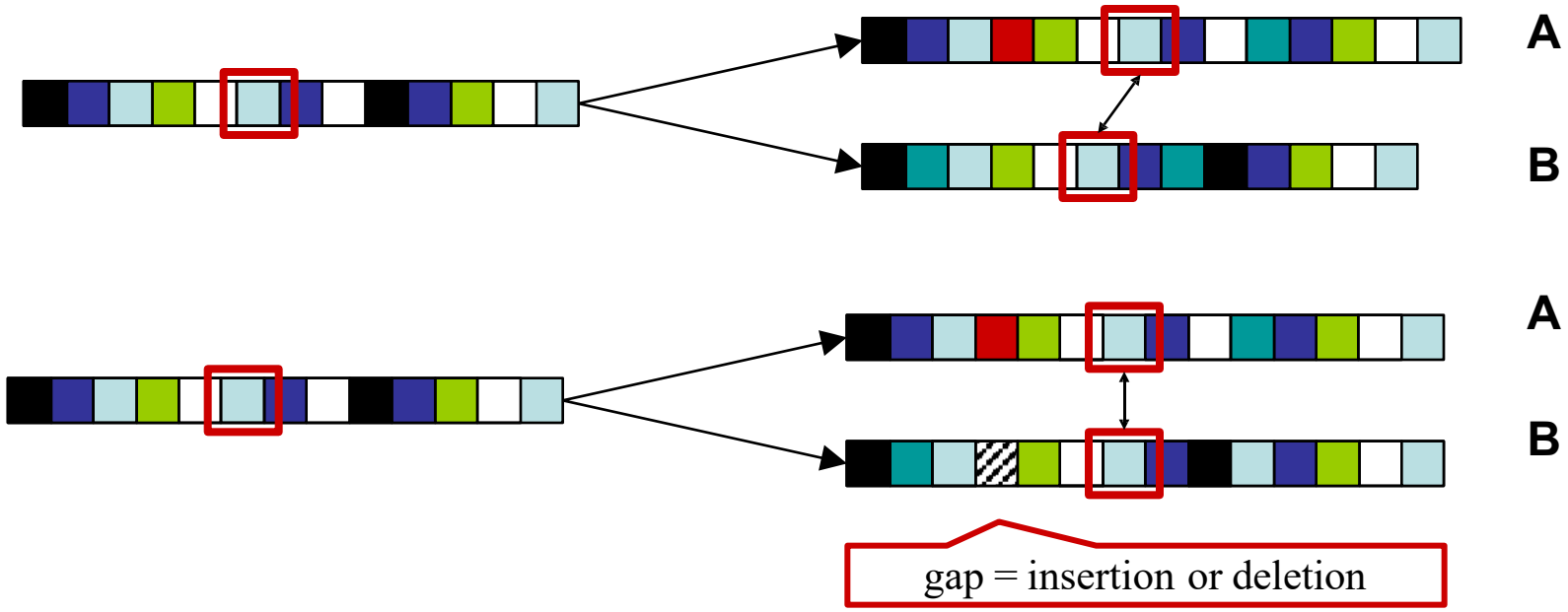
BLOSUM62

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

在 BLAST、ClustalW、MAFFT、MUSCLE 等软件中都是直接使用这张表

Gap penalty

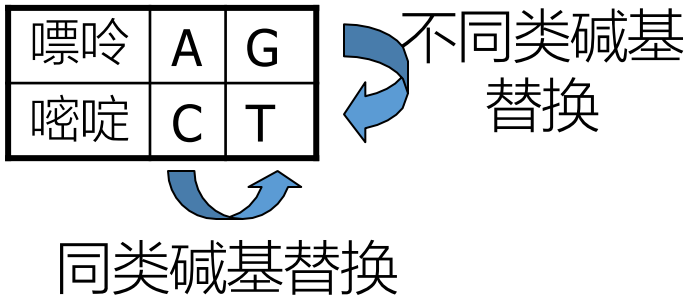
- Gap : an **Insertion** or **Deletion** during evolution
- Often have a **negative** score as "**penalty**"
- Much **less frequent** than residue substitution



Gap penalty

- 碱基替换打分

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2



- 每个空位罚5分

GAAT−C

C−ATAC

(-7) + (-5) + (2) + (2) + (-5) + 2= -11

序列比对 Sequence alignment

```
GAATC      GAAT-C      -GAAT-C
CATAC      C-ATAC      C-A-TAC

GAATC-     GAAT-C      GA-ATC
CA-TAC     CA-TAC      CATA-C
```

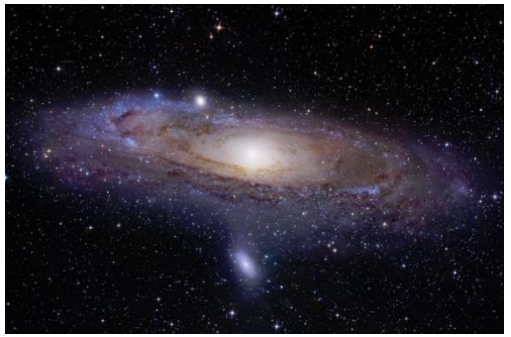
序列比对的组合复杂性

If gaps are allowed in every position and of every length, naive enumeration is exponential in the length of the sequences.

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2n}}{\sqrt{\pi n}}$$

For two sequences with 300 letters, 10^{180} possible alignments exist

The visible universe is estimated to contain $10^{78} \sim 10^{80}$ atoms (from: wikianswers)

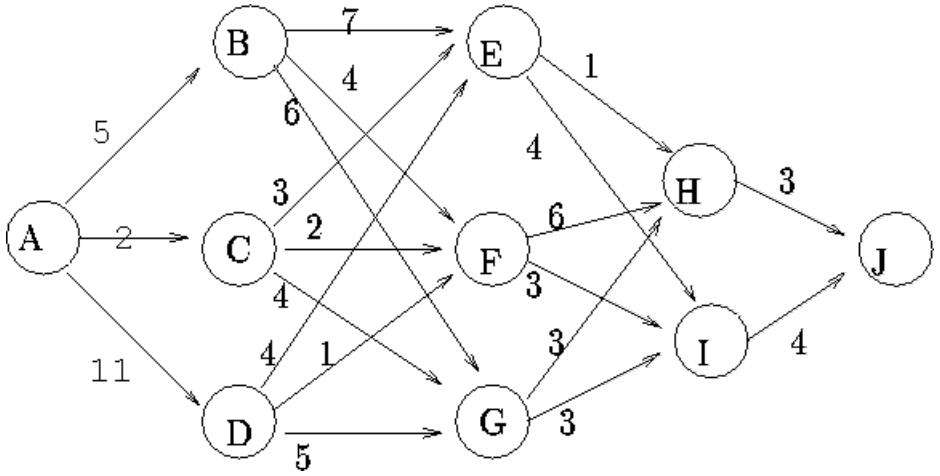


动态规划算法 (Dynamic Programming, DP)

动态规划(Dynamics Programming)，一种用来解决具有最优子结构(optimal substructure)性质优化问题的计算机算法

- 大问题问题的最优解可以从其子问题的最优解来有效地构建

➔将原始问题分解为若干个规模较小的同构子问题



$$F(A,J) = \min \begin{cases} F(A,H) + Dist(H,J) \\ F(A,I) + Dist(I,\mathrm{J}) \end{cases} \qquad F(A,A) = 0$$

全局比对 （Needleman-Wunsch算法）

全局比对是指将两条序列里面的所有字符进行比对，主要被用来寻找关系密切的序列。

假设x和y是需要比对的两个序列
F(i,j) 是$x_{1...i}$ and $y_{1...j}$之间最优比对的得分
s(A,B)是用A替换B(错配)的得分；d 是空位得分(线性)

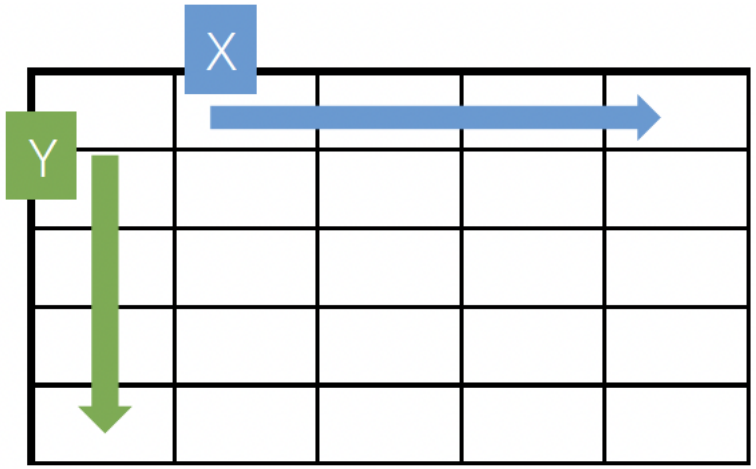
$$F(0,0)=0$$

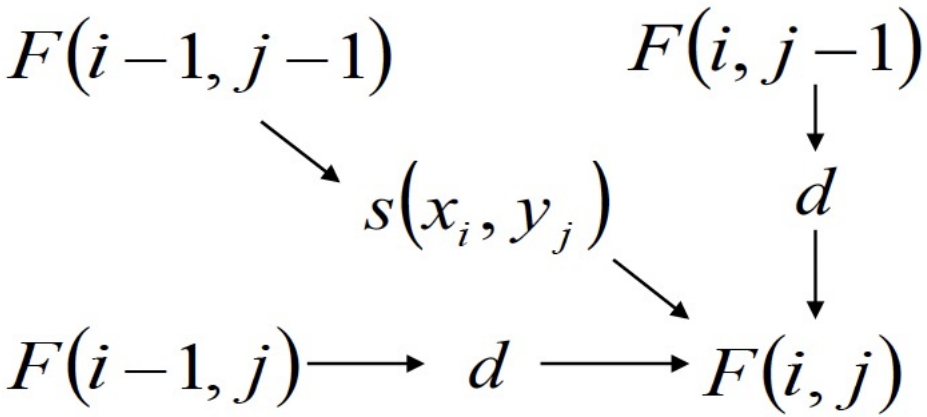
$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j) & x_i \text{ 比对到} y_j\\ F(i-1,j)+d & x_i \text{ 比对到空位}\\ F(i,j-1)+d & y_j \text{ 比对到空位}\end{cases}$$

全局比对

$$F(0,0)=0$$

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j) & x_i \text{ 比对到} y_j\\ F(i-1,j)+d & x_i \text{ 比对到空位}\\ F(i,j-1)+d & y_j \text{ 比对到空位}\end{cases}$$





全局比对算法示例

- 输入序列 S1: AAG
- 输入序列 S2: AGC

		A	A	G
A				
G				
C				

全局比对算法示例

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2

寻找 AAG and AGC 的最优比对

空位得分 d=-5

		A	A	G
	0			
A				
G				
C				

$$F(0,0)=0$$

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+d\\F(i,j-1)+d\end{cases}$$

全局比对算法示例

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2

寻找 AAG and AGC 的最优比对
空位得分 d=-5

		A	A	G
	0 →	-5 →	-10 →	-15
A	-5			
G	-10			
C	-15			

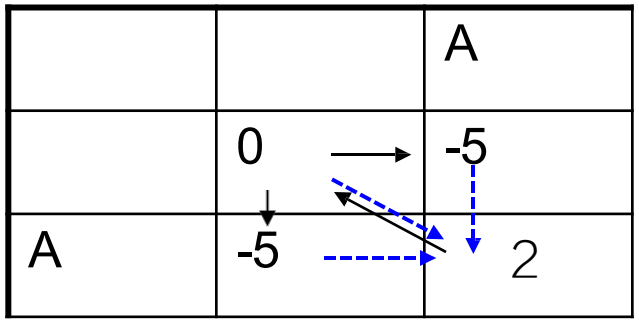
$$F(0,0)=0$$

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+d\\F(i,j-1)+d\end{cases}$$

全局比对算法示例

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2

寻找 AAG and AGC 的最优比对
空位得分 d=-5



$$F(0,0)=0$$

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+d\\F(i,j-1)+d\end{cases}$$

F(1,1) =

F(0,0) + s (A, A)　　0 + 2 = 2

F(0,1) + d　　−5 + (−5) = −10

F(1,0) + d　　−5 + (−5) = −10

全局比对算法示例

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2

寻找 AAG and AGC 的最优比对
空位得分 d=-5

		A	A	G
	0 →	-5 →	-10 →	-15
A	-5 ↓	2 →	-3 →	-8
G	-10 ↓	-3 ↓	-3 ↓	-1 ↓
C	-15	-8	-8	**-6**

$$F(0,0)=0$$

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+d\\F(i,j-1)+d\end{cases}$$

全局比对算法示例

从矩阵右下角逐步回溯至左上角。每个箭头指向序列中前一个位置的符号

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2

A A G –

– A G C

A A G –

A – G C

寻找 AAG and AGC 的最优比对

空位得分 d=-5

		A	A	G
	0	-5		
A		2	-3	
G				-1
C				**-6**

Needleman–Wunsch 算法：端到端全局比对

J. Mol. Biol. (1970) 48, 443–453

A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins

SAUL B. NEEDLEMAN AND CHRISTIAN D. WUNSCH

Department of Biochemistry, Northwestern University, and Nuclear Medicine Service, V. A. Research Hospital Chicago, Ill. 60611, U.S.A.

(*Received 21 July 1969*)

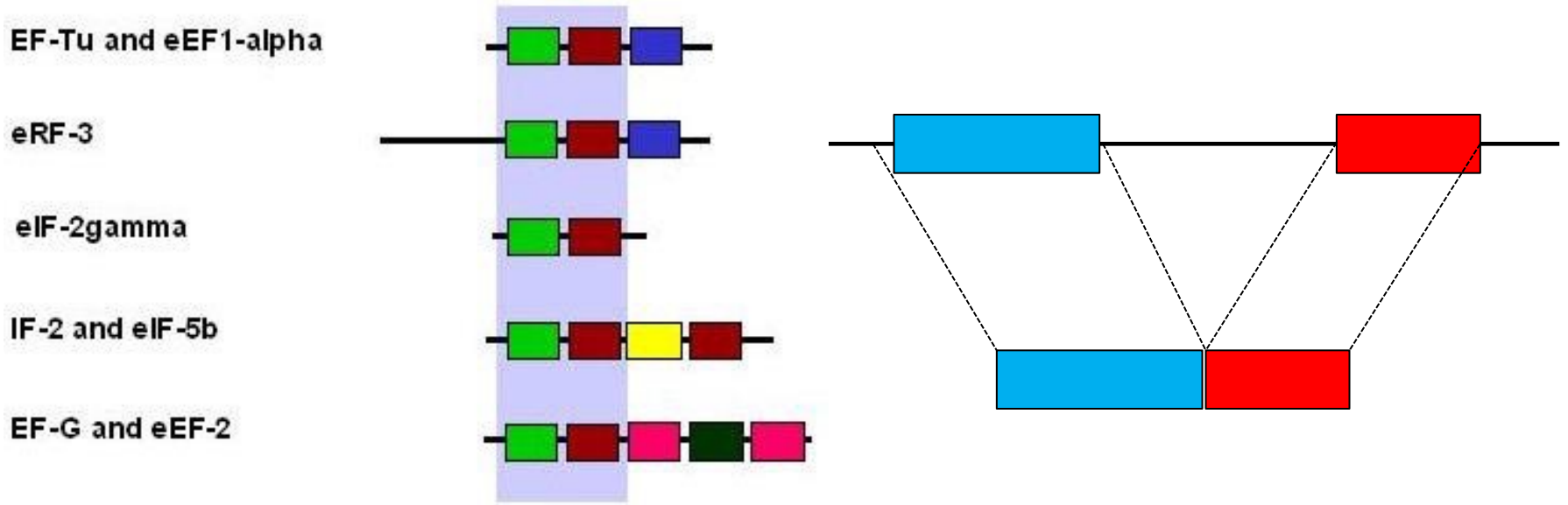
A computer adaptable method for finding similarities in the amino acid sequences of two proteins has been developed. From these findings it is possible to determine whether significant homology exists between the proteins. This information is used to trace their possible evolutionary development.

The maximum match is a number dependent upon the similarity of the sequences. One of its definitions is the largest number of amino acids of one protein that can be matched with those of a second protein allowing for all possible interruptions in either of the sequences. While the interruptions give rise to a very large number of comparisons, the method efficiently excludes from consideration those comparisons that cannot contribute to the maximum match.

Comparisons are made from the smallest unit of significance, a pair of amino acids, one from each protein. All possible pairs are represented by a two-dimensional array, and all possible comparisons are represented by pathways through the array. For this maximum match only certain of the possible pathways must be evaluated. A numerical value, one in this case, is assigned to every cell in the array representing like amino acids. The maximum match is the largest number that would result from summing the cell values of every pathway.

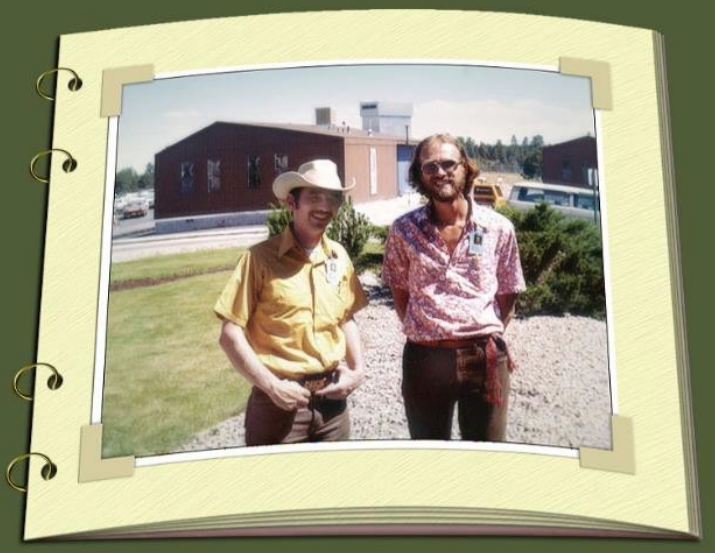
局部比对

局部比对不追求对两个完整的序列进行比对，而是在每个序列中使用某些局部区域片段进行比对，代表算法是**Smith-Waterman**局部比对算法



Smith-Waterman 算法：局部比对

Smith and Waterman at Los Alamos, New Mexico
Photo by David Lipman, taken summer of 1980



(http://www.cmb.usc.edu/people/msw/SmithWaterman.html)

J. Mol. Biol. (1981), **147**, 195–197

Identification of Common Molecular Subsequences

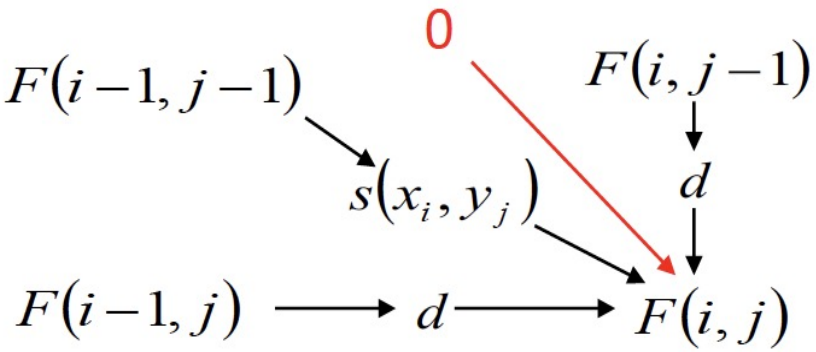
The identification of maximally homologous subsequences among sets of long sequences is an important problem in molecular sequence analysis. The problem is straightforward only if one restricts consideration to contiguous subsequences (segments) containing no internal deletions or insertions. The more general problem has its solution in an extension of sequence metrics (Sellers 1974; Waterman *et al.*, 1976) developed to measure the minimum number of "events" required to convert one sequence into another.

These developments in the modern sequence analysis began with the heuristic homology algorithm of Needleman & Wunsch (1970) which first introduced an iterative matrix method of calculation. Numerous other heuristic algorithms have been suggested including those of Fitch (1966) and Dayhoff (1969). More mathematically rigorous algorithms were suggested by Sankoff (1972), Reichert *et al.* (1973) and Beyer *et al.* (1979), but these were generally not biologically satisfying or interpretable. Success came with Sellers (1974) development of a true metric measure of the distance between sequences. This metric was later generalized by Waterman *et al.* (1976) to include deletions/insertions of arbitrary length. This metric represents the minimum number of "mutational events" required to convert one sequence into another. It is of interest to note that Smith *et al.* (1980) have recently shown that under some conditions the generalized Sellers metric is equivalent to the

双序列局部比对动态规划算法公式

$$F(0,0)=0$$

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+d\\F(i,j-1)+d\\0\end{cases}$$

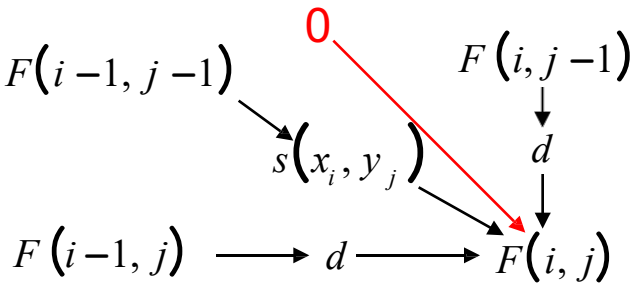


局部比对算法示例

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2

寻找AAG and AGC的最优局部比对
空位得分 d = -5

		A	A	G
A				
G				
C				

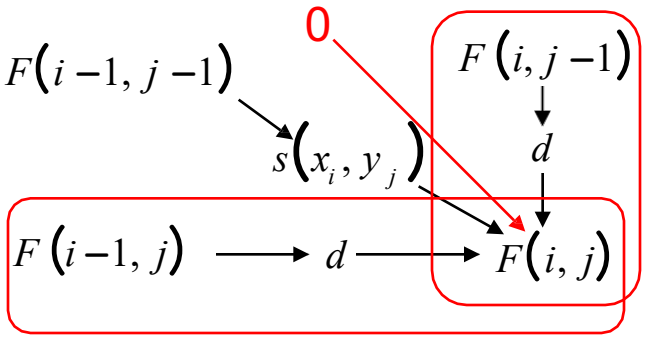


局部比对算法示例

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2

寻找AAG and AGC的最优局部比对
空位得分 d = -5

		A	A	G
	0	0	0	0
A	0			
G	0			
C	0			



局部比对算法示例

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2

寻找AAG and AGC的最优局部比对
空位得分 d = -5

		A	A	G
	0	0	0	0
A	0	2	2	0
G	0	0	0	4
C	0	0	0	0

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+d\\F(i,j-1)+d\\0\end{cases}$$

局部比对算法示例

	A	C	G	T
A	2	-7	-5	-7
C	-7	2	-7	-5
G	-5	-7	2	-7
T	-7	-5	-7	2

寻找AAG and AGC的最优局部比对
空位得分 d = -5

		A	A	G
	0	0	0	0
A	0	2	2	0
G	0	0	0	4
C	0	0	0	0

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+d\\F(i,j-1)+d\\0\end{cases}$$

局部比对算法示例

从矩阵中最大值所在位置开始回溯到0为止

A G
A G

		A	A	G
	0	0	0	0
A	0	2	2	0
G	0	0	0	4
C	0	0	0	0

局部比对算法示例

另一个最优局部比对结果

A
A

		A	A	G
	0	0	0	0
A	0	2	2	0
G	0	0	0	4
C	0	0	0	0

全局比对 与 局部比对 的差异

$$F(0,0)=0$$

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+d\\F(i,j-1)+d\end{cases}$$

全局比对

$$F(0,0)=0$$

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+d\\F(i,j-1)+d\\0\end{cases}$$

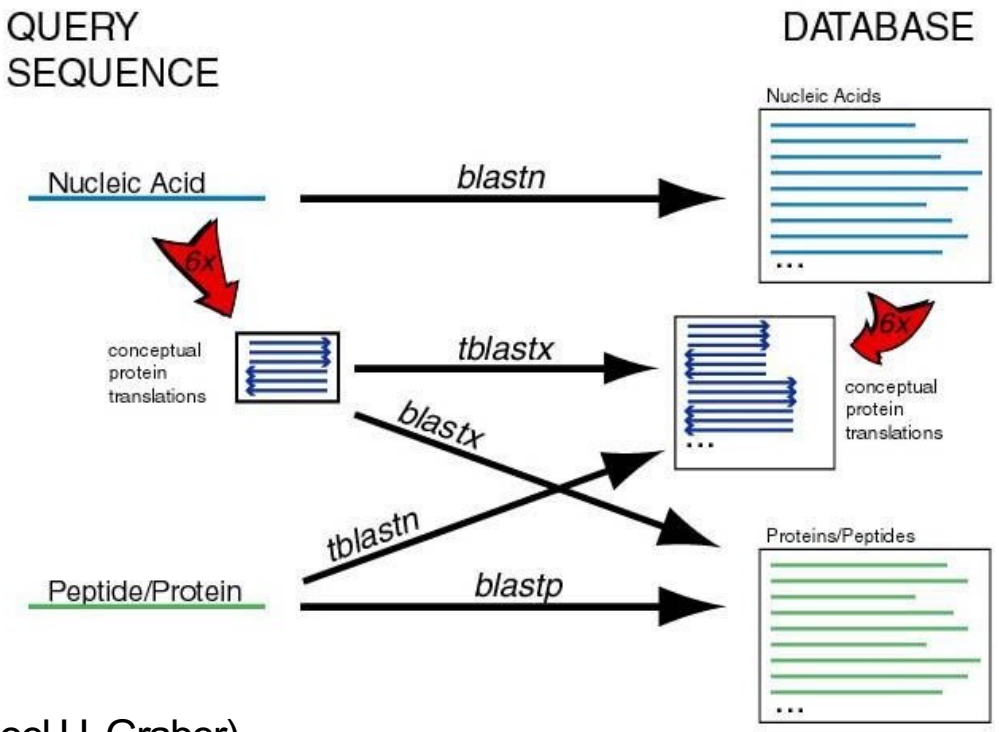
局部比对

序列比对的应用

- **序列数据库检索**
- **基于序列比较的序列聚类**
- **多重序列比对**

应用一：序列数据库检索

- **基于局部比对搜索工具(Basic Local Alignment Search Tool, BLAST)**



(Joel H. Graber)

1. blastn
   - 查询：核酸序列
   - 数据库：核酸数据库
   - 用途：DNA vs DNA
2. blastp
   - 查询：蛋白质序列
   - 数据库：蛋白质数据库
   - 用途：protein vs Protein
3. blastx
   - 查询：核酸序列(转化为6个阅读框的蛋白质翻译)
   - 数据库：蛋白质数据库
   - 用途：DNA vs Protein
4. tblastn
   - 查询：蛋白质序列
   - 数据库：核酸数据库
   - 用途： Protein vs DNA
5. tblastx
   - 查询：核酸序列（6 个阅读框翻译）
   - 数据库：核酸数据库（6 个阅读框翻译）
   - 用途： DNA vs DNA（但比较的是翻译后的蛋白层面）

应用一：序列数据库检索

- 在检验搜索结果方面，因为数据库通常非常庞大，随机噪声可能会产生看似显著但实际上没有生物学意义的比对。
- Blast 引入了期望值（***E*-value**) 来衡量特定比对的统计显著性。
- E： 在给定的数据库参数下，若随机选取的一段序列的比对得分大于或等于当前实际比对得分，则这种随机选取的序列的期望个数即期望值。

$$E = kmne^{-\lambda S}$$

S：比对得分（alignment score）

m,n：分别是查询序列和数据库序列的长度

k, λ：统计学参数（和打分矩阵有关）

应用一：序列数据库检索

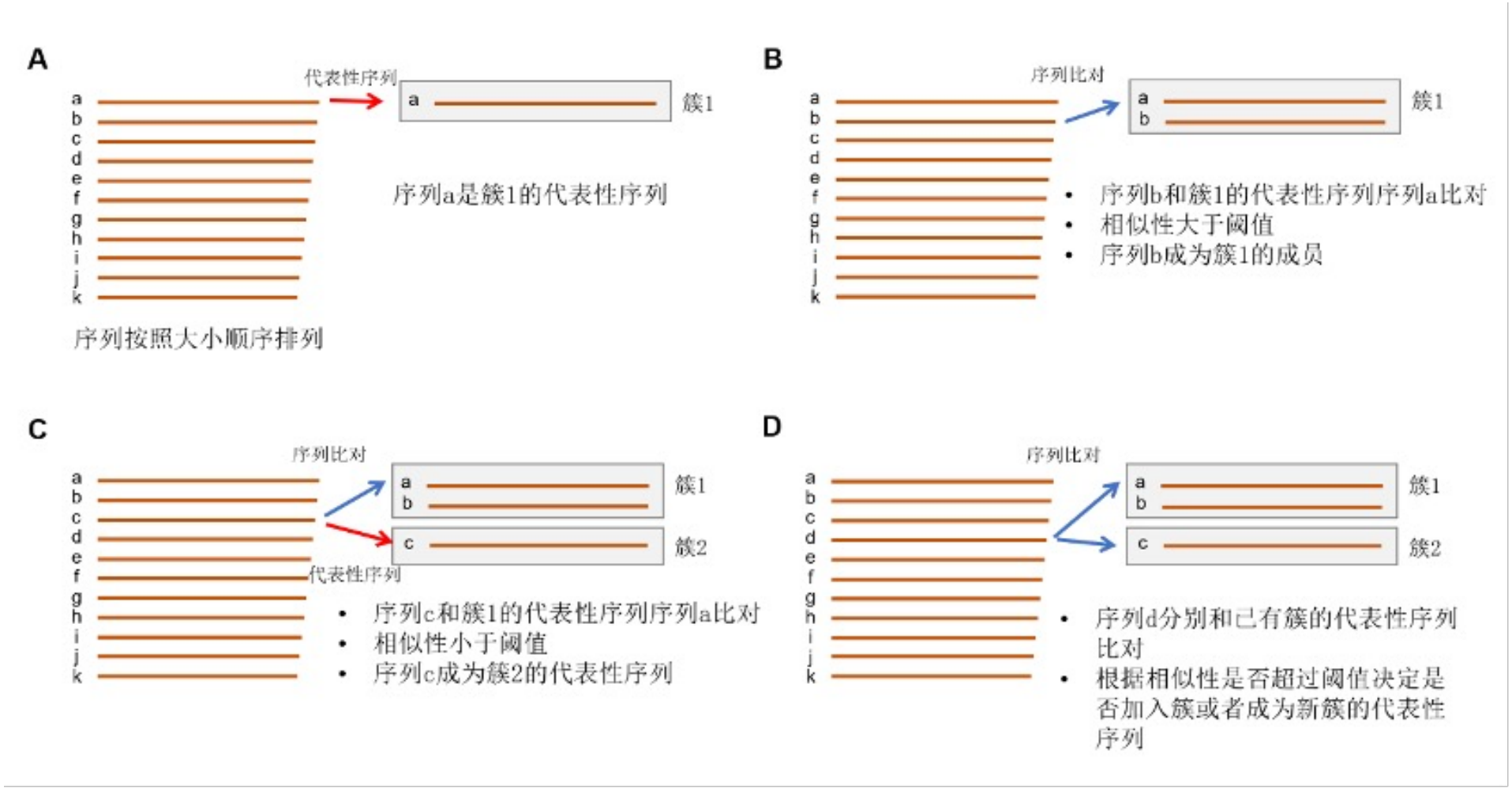
- Blast 首次实现了对大规模序列数据库的快速检索，从而使研究人员能够快速识别和分析生物序列
- 在基因识别与注释、蛋白质功能预测、同源基因鉴定等领域均有广泛应用
- 目前，Blast已成为生物信息学领域中最重要的工具之一。

应用二：基于序列比较的序列聚类

- 通常认为，序列上相类似的蛋白质或者基因往往具有相类似的功能，而且相似的程度往往和功能相似的程度正相关；类似的，序列相似程度在**不考虑趋同进化**等特殊事件的前提下也可以作为序列间进化关系远近的表征。因此，通过将蛋白或者DNA序列按照互相之间相似的程度分成多个簇(cluster)进行序列聚类 (sequence clustering)，可以实现对蛋白质或DNA分子的功能与演化分类。
- 常见算法：
  - CD- HIT
  - H clust
  - Linclust
- 它们用的都是 **贪心增量策略（greedy incremental strategy）**：从长到短排序，依次挑代表序列，再把相似的放到它的簇里。

应用二：基于序列比较的序列聚类

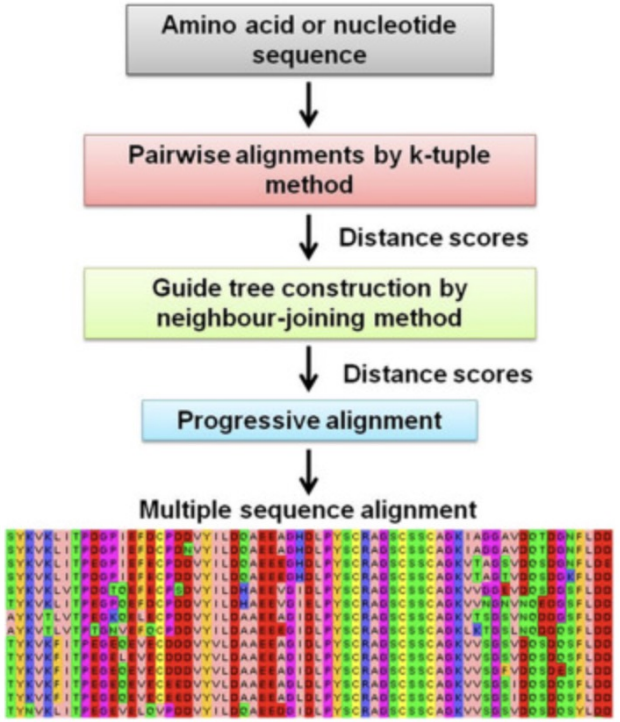
- 使用**贪心增量策略**进行序列聚类的原理



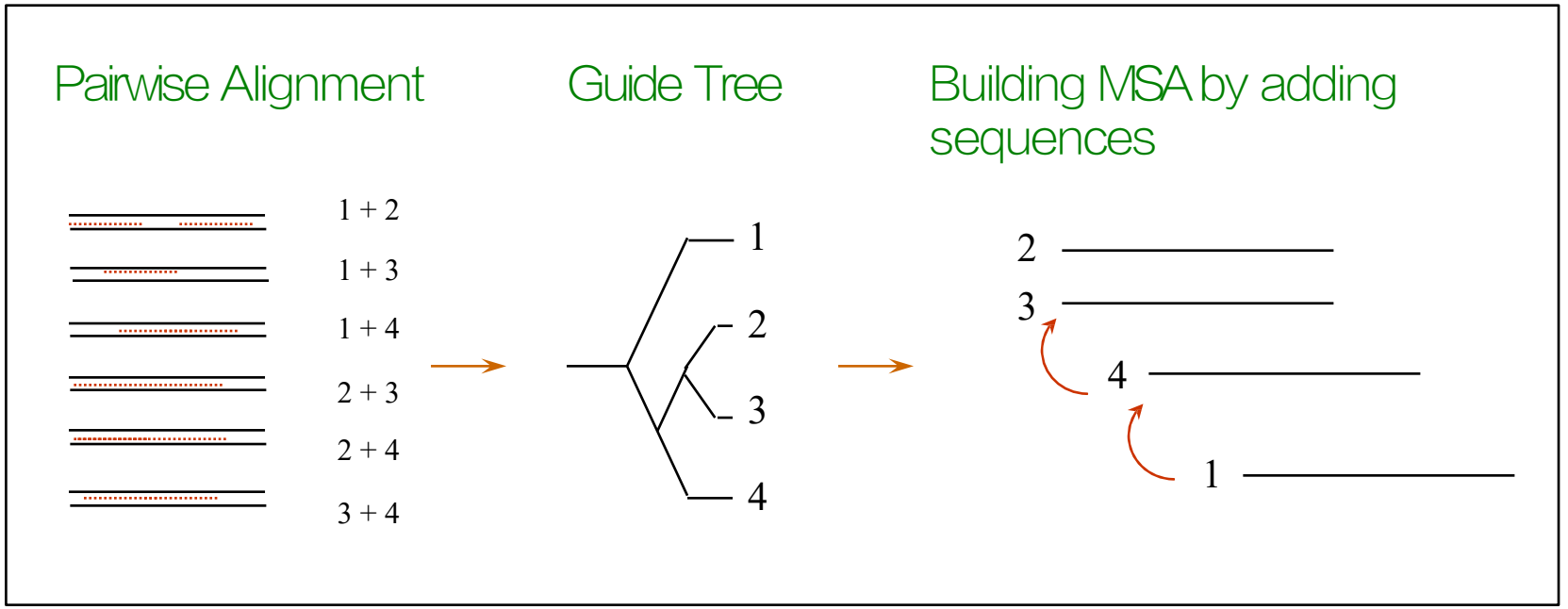
应用三：多重序列比对

- 多重序列比对（multiple sequence alignment, MSA) 是双序列比对的一般化推广。
- 这些基于动态规划的算法，当比对的序列增加到3条或更多时，计算复杂性呈指数级增长。
  - 假定比对 P 条序列，每条序列长度为 n，比对的复杂度 $O = n^{p}$

代表性算法： Cluster W, Muscle, Mafft, T-coffee



Workflow of ClustalW showing three steps



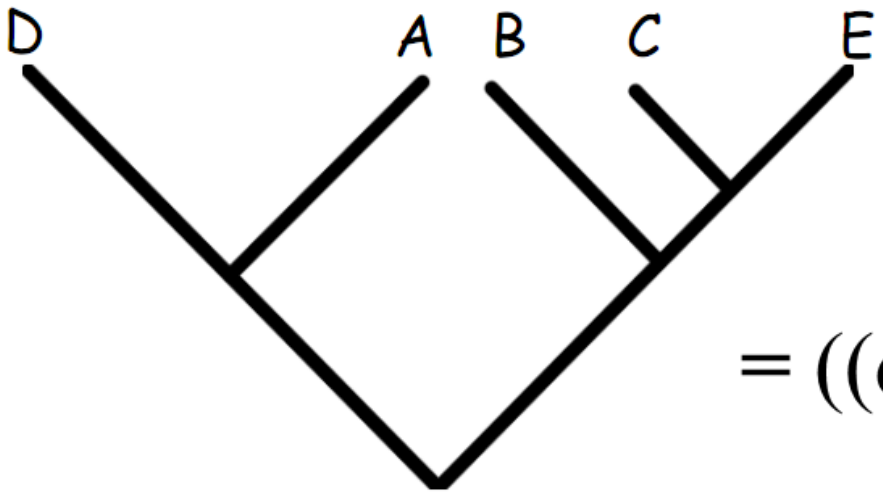
第三节：分子演化树构建

本章导语

- **通过系统比较不同物种或不同个体间的序列差异，可以构建演化树以解析其间的演化联系，并进而表征特定DNA/蛋白质分子从共同祖先逐渐演化分化的过程。演化树(又称系统发育树，Phylogenetic Tree)是一种针对特定演化关系的图形表示，类似于家谱图，演化树提供了生物多样性和演化历史的重要信息，可用于解释生物特征的起源和演化。**

认识系统发育树（phylogeny）

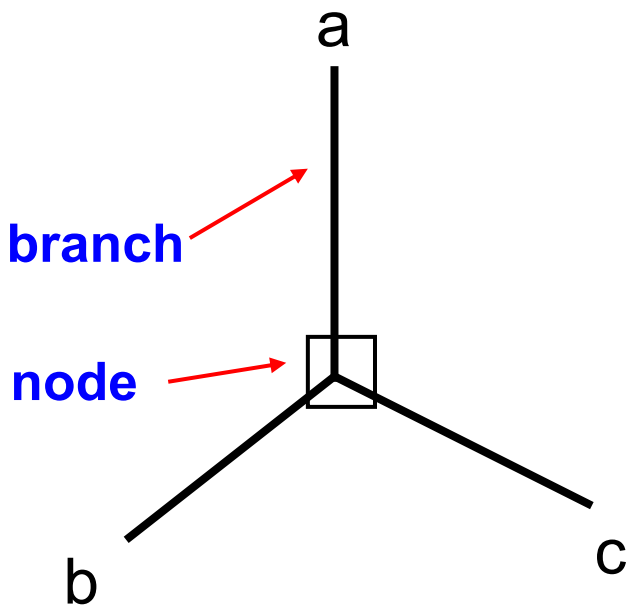
树的Newick格式表示法



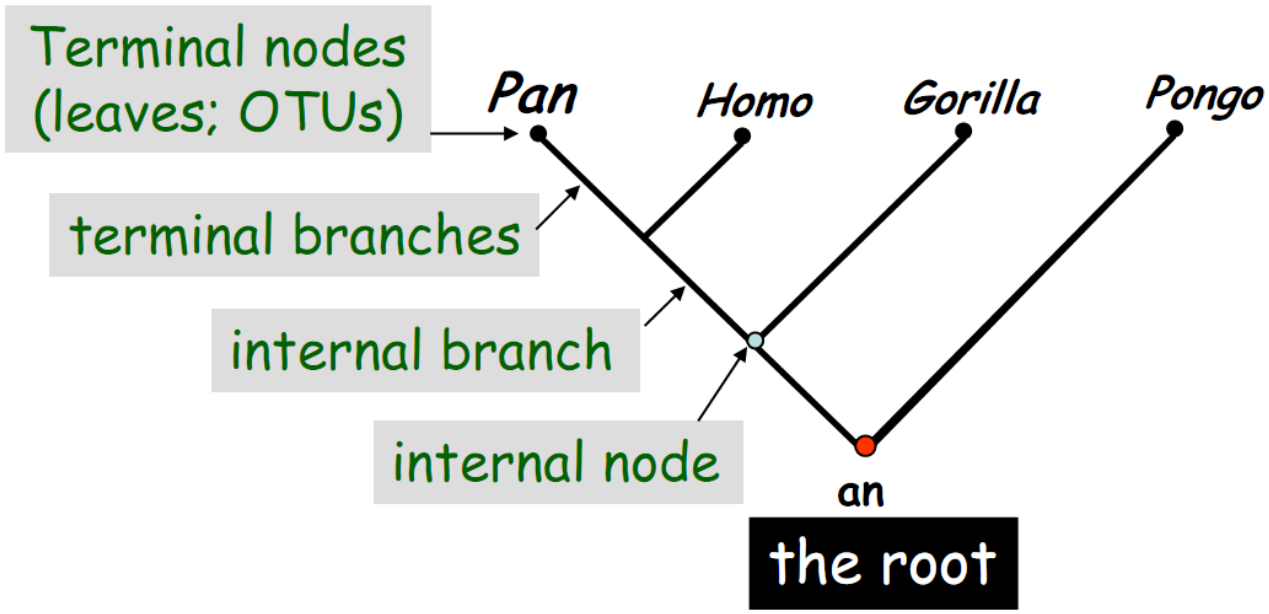
Newick format

= ((d,a),(b,(c,e))) or ((b,(e,c),(a,d))

认识系统发育树（phylogeny）



Unrooted tree



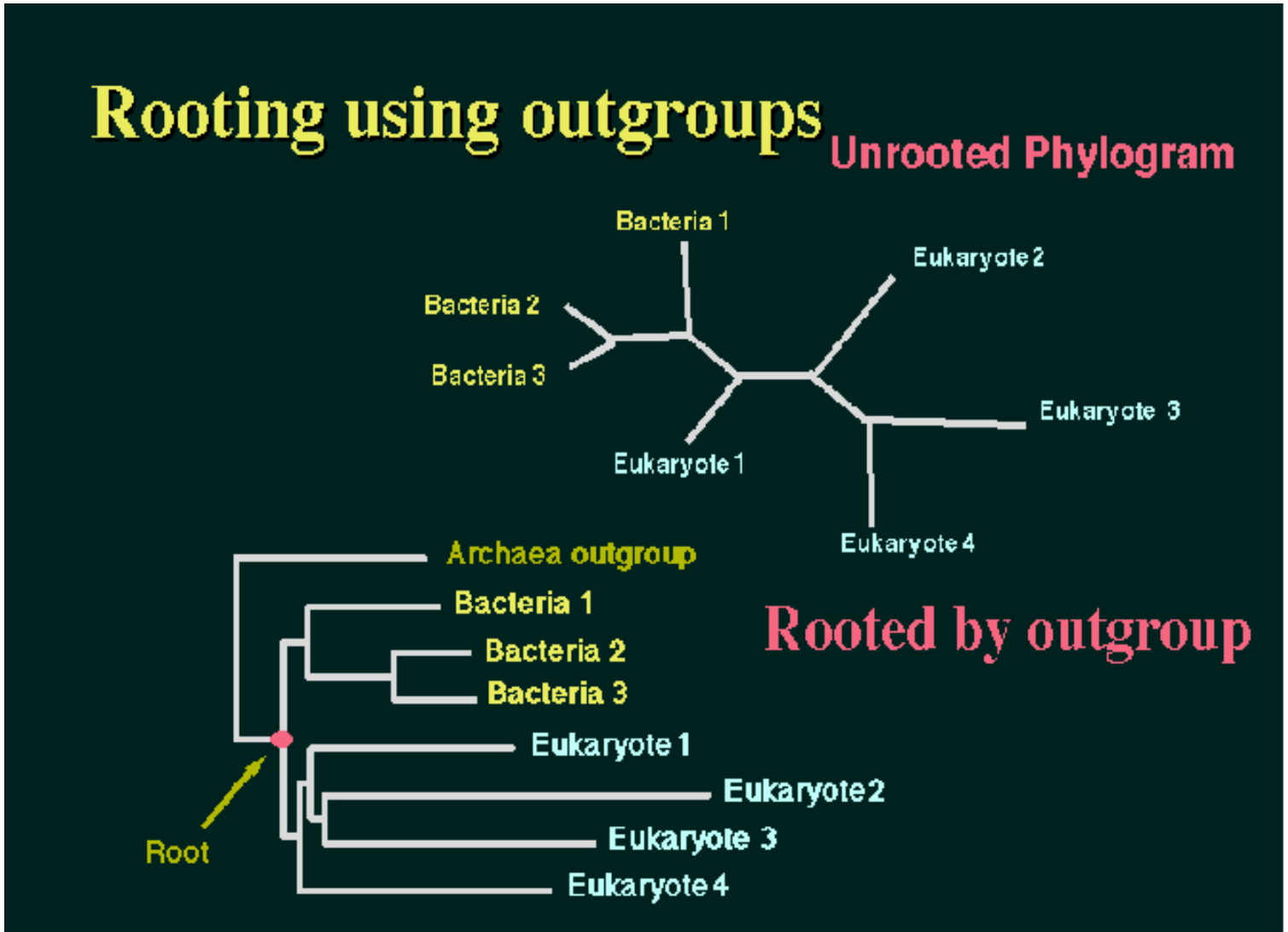
Rooted tree

系统发育树一般是一种二权树的结构，每一个树枝(Branch)代表着一段共同的进化历史，每一个树的结点(node)代表着一个物种的分歧事件

注：对于给定数量的物种，无根树的数目总是比有根树少

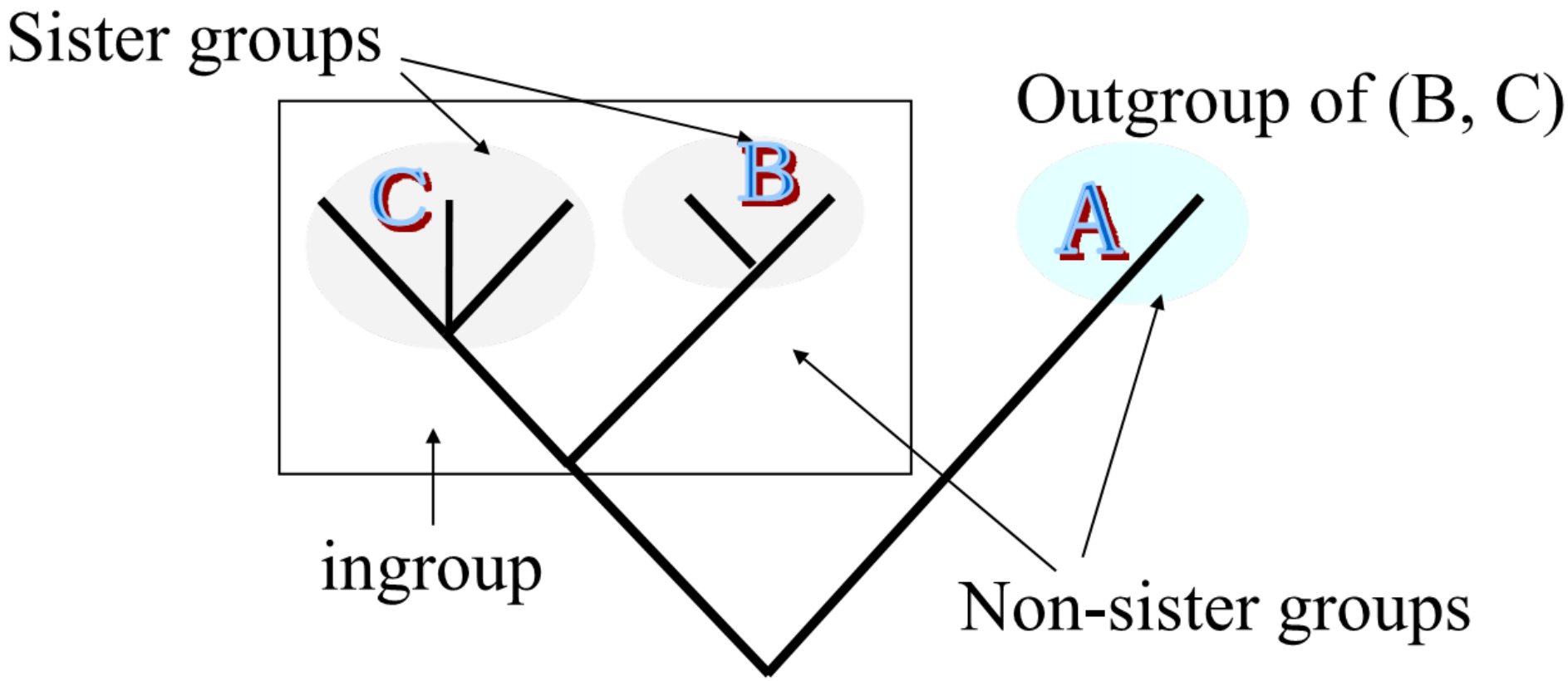
认识系统发育树（phylogeny）

外类群 Outgroup



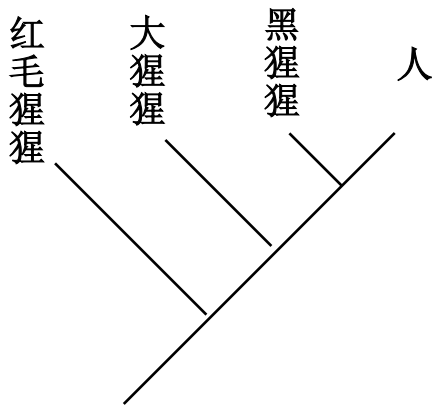
认识系统发育树（phylogeny）

姐妹群(sister group)关系



认识系统发育树（phylogeny）

系统发育树只代表了物种间的关系，它不是一种阶梯，不代表谁比谁更高等



左图并不表明人是从黑猩猩进化而来的，人与黑猩猩是进化上的堂兄弟，它们所拥有的共同祖先既不是人也不是黑猩猩。人并不比黑猩猩“高级”或者说更“进化”，人与黑猩猩只是在各自的进化过程中产生了自己独特的特性而已。

常用的建树方法

- Distance-based methods
  - UPGMA
  - Neighbor Joining (邻接法)
- Character-based methods
  - Maximum Parsimony (最大简约法)
  - Maximum Likelihood (极大似然法)
  - Bayesian Inference (贝叶斯法)

P-distance

Site ↓

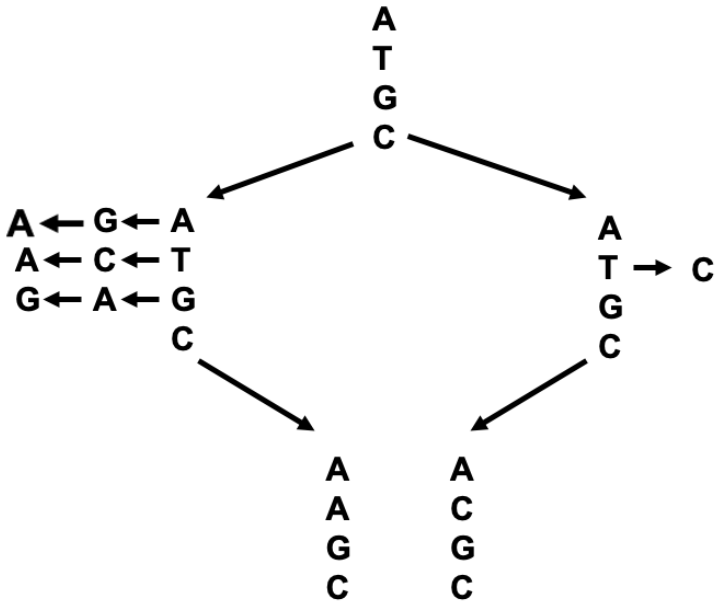
	1	2	3	4	5	6	7	8	9	10
Ⅰ	A	T	A	G	G	A	C	T	C	G
Ⅱ	A	T	G	G	G	T	C	T	C	G
Ⅲ	A	G	C	G	G	C	C	T	T	A
Ⅳ	A	G	C	G	G	G	A	T	T	G
Ⅴ	A	G	T	G	G	T	A	A	T	T

← Character

两条序列间的距离被定义为平均每个位点核苷酸置换的期望数。如何计算两条序列间的距离？最简单的一种计算方法就是差异位点比例，有时称为**P距离**。比如上图Ⅰ Ⅱ两条序列间的P距离就为20% (0.2).

核苷酸替代模型

P距离用于估计关系非常近的的序列还可以，但在大多数情况下对置换数估计明显不足，因为一个可变位点可能是一次置换的结果，也可能是多次置换的结果，甚至一个不变的位点也可能经历过回复置换。同一位点的多重置换导致了一些隐藏的变化，所以P距离并不是进化时间的线性函数。为了更准确地估计置换数目，我们需要一个概率模型来描述核苷酸间的变化，这就是核苷酸替代模型（nucleotide substitution model）。

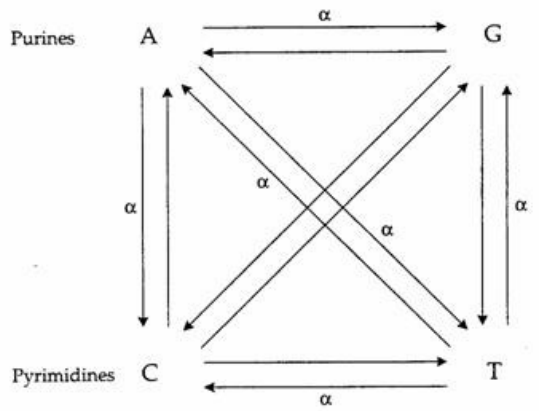


JC69 Model

JC69 model (Jukes and Cantor, 1969)

JC69模型是最简单的置换模型，它假定四种核苷的出现频率都固定为1/4，所有核苷间的突变几率一致，速率为μ

Jukes and Cantor's model (JC)



- Equal substitution rate (α)
- Equal base frequencies

在这个模型下，两条序列的距离 $d = -\frac{3}{4}\ln(1 - \frac{4}{3}p)$

可以看出，**JC69 model** 只有一个可变参量μ，碱基突变速率

核苷酸替代模型

模型名称	全称	参数数量	主要假设
JC69	Jukes–Cantor (1969)	1	所有碱基频率相等，所有替换速率相等。
K80	Kimura 2-Parameter (1980)	2	区分转换（transition）与颠换（transversion）速率，但碱基频率仍相等。
F81	Felsenstein (1981)	4	各碱基频率不同，但替换速率相同。
HKY85	Hasegawa–Kishino–Yano (1985)	5	区分转换/颠换速率，允许碱基频率不等。
TN93	Tamura–Nei (1993)	6	两种转换（A↔G 与 C↔T）速率不同，碱基频率可不等。
GTR	General Time Reversible	9	所有碱基间替换速率均可不同，且碱基频率可不等。

Inferring Phylogenetic Tree by UPGMA

UPGMA (Unweighted Pair-Group Method using arithmetic Average)

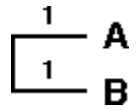
- 输入：一个 **距离矩阵**（表示序列/物种两两之间的进化距离）
- 流程：
  1. 在矩阵中找到距离最小的一对（最相似的一对）
  2. 把它们聚成一个新的类簇（cluster），并且计算它与其它类簇之间的新平均距离
  3. 更新距离矩阵
  4. 重复步骤 1-3，直到所有样本都聚到一棵树上。

- 特点：UPGMA 默认分子钟假设（molecular clock），也就是认为不同分支进化速率相同，因此树是 **等时树（ultrametric tree）**。

Inferring Phylogenetic Tree by UPGMA

1. 起始： 距离矩阵
   - 输入是一个序列/物种之间的**距离矩阵**（越小表示越相似）
   - 找出矩阵中最小的距离（比如 A 和 B = 2）

	A	B	C	D	E
B	2				
C	4	4			
D	6	6	6		
E	6	6	6	4	
F	8	8	8	8	8



Inferring Phylogenetic Tree by UPGMA

2. 合并最近的两个单元 (Cluster)
   - 将它们作为一个“复合节点 (Composite Unit)”
   - 在树上画一个分支，把它们连在一起，并标注分支长度

3. 更新距离矩阵
   - 对于其它单元，计算它们与新形成的复合单元的距离 $\mathrm{dist}((A,B),X) = \frac{\mathrm{dist}(A,X) + \mathrm{dist}(B,X)}{2}$

```
dist (A,B),C = (dist A,C + dist B,C) / 2 = (4 + 4) / 2 = 4
dist (A,B),D = (dist A,D + dist B,D) / 2 = (6 + 6) / 2 = 6
dist (A,B),E = (dist A,E + dist B,E) / 2 = (6 + 6) / 2 = 6
dist (A,B),F = (dist A,F + dist B,F) / 2 = (8 + 8) / 2 = 8
```

	A	B	C	D	E
B	2				
C	4	4			
D	6	6	6		
E	6	6	6	4	
F	8	8	8	8	8

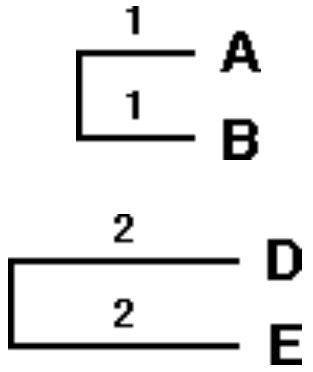
	AB	C	D	E
C	4			
D	6	6		
E	6	6	4	
F	8	8	8	8

Inferring Phylogenetic Tree by UPGMA

4. 迭代
   - 找出更新后的矩阵中最小的距离，再合并
   - 重复更新 → 直到所有物种都合并进一棵树

◆ Second Iteration

	A,B	C	D	E
C	4			
D	6	6		
E	6	6	4	
F	8	8	8	8

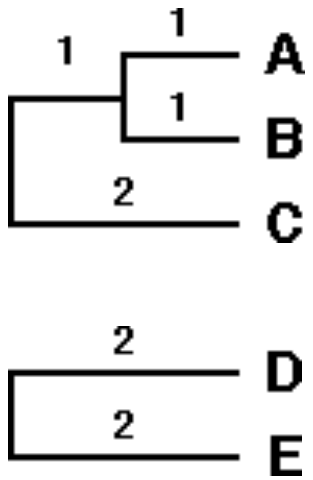


Inferring Phylogenetic Tree by UPGMA

4. 迭代
   - 找出更新后的矩阵中最小的距离，再合并
   - 重复更新 → 直到所有物种都合并进一棵树

◆ Third Iteration

	A,B	C	D,E
C	4		
D,E	6	6	
F	8	8	8

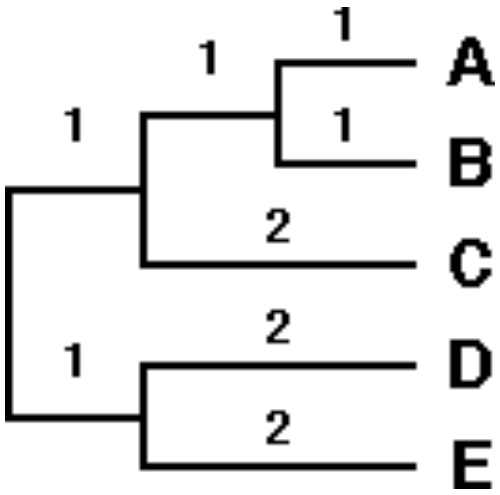


Inferring Phylogenetic Tree by UPGMA

4. 迭代
   - 找出更新后的矩阵中最小的距离，再合并
   - 重复更新 → 直到所有物种都合并进一棵树

◆ Forth Iteration

	AB,C	D,E
D,E	6	
F	8	8

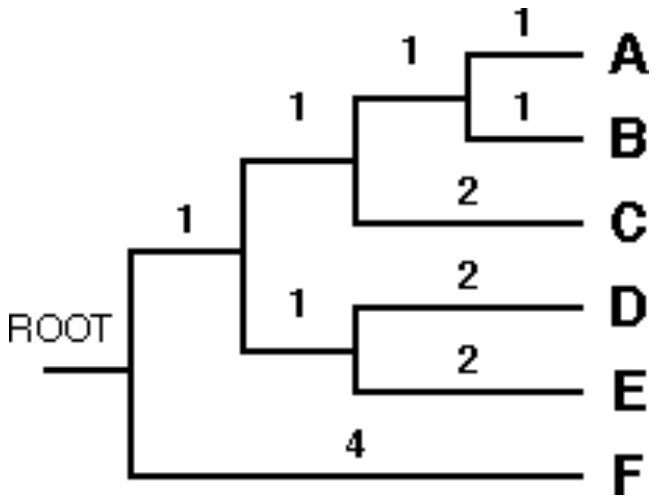


Inferring Phylogenetic Tree by UPGMA

4. 迭代
   - 找出更新后的矩阵中最小的距离，再合并
   - 重复更新 → 直到所有物种都合并进一棵树

◆ Fifth Iteration

	ABC,DE
F	8



邻接法 (Neighbors Joining , NJ)

- **Proposed by Saitou and Nei in 1987**
- 与UPGMA不同，NJ 不要求分子钟假设（即进化速率相同），它允许 **不同分支的进化速率不一样**
- 邻接法引入了“邻居”的概念，通过迭代连接最近的“邻居”来构建最终的进化树
- 目标是找到一棵 **最小进化树**，即**树的所有分支长度总和最小** 的那棵树

Masatoshi Nei, Kyoto prize-winning evolutionary geneticist, dies at 92

BY SUDHIR KUMAR AND GREG FORNIA · 5/19/23

Neighbor Joining Algorithm

	A	B	C	D	E	F	G	*r*
A	NA							93.0
B	63	NA						80.8
C	94	79	NA					87.0
D	111	96	47	NA				96.0
E	67	16	83	100	NA			84.8
F	23	58	89	106	62	NA		88.0
G	107	92	43	20	96	102	NA	92.0

（1）计算每个序列到其他序列的平均距离

$$r_i = \frac{1}{N-2}\sum_{k \neq i} d_{ik}$$

$$r_A = \frac{1}{5}(d_{AB} + d_{AC} + d_{AD} + d_{AE} + d_{AF} + d_{AG})$$

$$r_A = \frac{1}{5}(63 + 94 + 111 + 67 + 23 + 107) = \frac{465}{5} = 93.0$$

$$r_B = \frac{1}{5}(63 + 79 + 96 + 16 + 58 + 92) = \frac{404}{5} = 80.8$$

$$r_C = \frac{1}{5}(94 + 79 + 47 + 83 + 89 + 43) = \frac{435}{5} = 87.0$$

Neighbor Joining Algorithm

	A	B	C	D	E	F	G
A	NA						
B	63	NA					
C	94	79	NA				
D	111	96	47	NA			
E	67	16	83	100	NA		
F	23	58	89	106	62	NA	
G	107	92	43	20	96	102	NA

r
93.0
80.8
87.0
96.0
84.8
88.0
92.0

	A	B	C	D	E	F	G
A	NA	-110.8	-86.0	-78.0	-110.8	-158.0	-78.0
B		NA	-88.8	-80.8	-149.6	-110.8	-80.8
C			NA	-136.0	-88.8	-86.0	-136.0
D				NA	-80.8	-78.0	**-168.0**
E					NA	-110.8	-80.8
F						NA	-78.0
G							NA

(2) 计算校正后的距离

$$D_{ij} = d_{ij} - r_i - r_j$$

$$D_{AB} = 63 - 93.0 - 80.8 = -110.8$$

$$D_{DG} = 20 - 96 - 92 = -168$$ (最小，选为邻居)

$$D_{CF} = 89 - 87 - 88 = -86$$

$$D_{BE} = 16 - 80.8 - 84.8 = -149.6$$

$$\cdots$$

其中最小的是 D 与 G，所以本轮配对邻居：$\boxed{D\text{–}G}$。

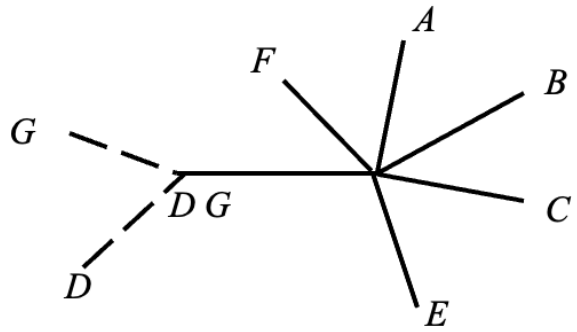
Neighbor Joining Algorithm

	A	B	C	D	E	F	G
A	NA						
B	63	NA					
C	94	79	NA				
D	111	96	47	NA			
E	67	16	83	100	NA		
F	23	58	89	106	62	NA	
G	107	92	43	20	96	102	NA

r
93.0
80.8
87.0
96.0
84.8
88.0
92.0

	A	B	C	D	E	F	G
A	NA	-110.8	-86.0	-78.0	-110.8	-158.0	-78.0
B		NA	-88.8	-80.8	-149.6	-110.8	-80.8
C			NA	-136.0	-88.8	-86.0	-136.0
D				NA	-80.8	-78.0	**-168.0**
E					NA	-110.8	-80.8
F						NA	-78.0
G							NA

（3）合并DG，重新计算D、G两点到合并后DG点的距离



$$d_{i,(ij)} = \frac{1}{2}d_{ij} + \frac{1}{2}(r_i - r_j), \qquad d_{j,(ij)} = \frac{1}{2}d_{ij} + \frac{1}{2}(r_j - r_i)$$

$$d_{d,dg} = \frac{1}{2}d_{d,g} + \frac{1}{2}(r_d - r_g) = \frac{1}{2} * 20 + \frac{1}{2}(96 - 92) = 12$$

$$d_{g,dg} = \frac{1}{2}d_{d,g} + \frac{1}{2}(r_g - r_d) = \frac{1}{2} * 20 + \frac{1}{2}(92 - 96) = 8$$

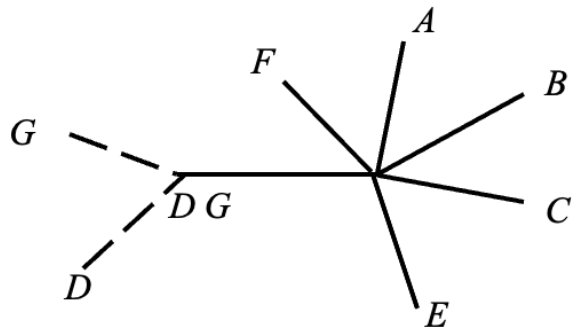
Neighbor Joining Algorithm

	A	B	C	D	E	F	G
A	NA						
B	63	NA					
C	94	79	NA				
D	111	96	47	NA			
E	67	16	83	100	NA		
F	23	58	89	106	62	NA	
G	107	92	43	20	96	102	NA

r
93.0
80.8
87.0
96.0
84.8
88.0
92.0

	A	B	C	E	F	**DG**
A	NA					
B	63	NA				
C	94	79	NA			
E	67	16	83	NA		
F	23	58	89	62	NA	
DG	99	84	35	88	94	NA

(4) 计算其他点到合并后DG点的距离



$$d_{(ij),k} = \frac{d_{ik} + d_{jk} - d_{ij}}{2}$$

- $d_{(DG),A} = (111 + 107 - 20)/2 = \boxed{99}$
- $d_{(DG),B} = (96 + 92 - 20)/2 = \boxed{84}$
- $d_{(DG),C} = (47 + 43 - 20)/2 = \boxed{35}$
- $d_{(DG),E} = (100 + 96 - 20)/2 = \boxed{88}$
- $d_{(DG),F} = (106 + 102 - 20)/2 = \boxed{94}$

Neighbor Joining Algorithm

	A	B	C	E	F	DG	r
A	NA						86.5
B	63	NA					75.0
C	94	79	NA				95.0
E	67	16	83	NA			79.0
F	23	58	89	62	NA		81.5
DG	99	84	35	88	94	NA	100.0

	A	B	C	E	F	DG
A	NA	-98.5	-87.5	-98.5	-145.0	-87.5
B		NA	-91.0	-138.0	-98.5	-91.0
C			NA	-91.0	-87.5	-160.0
E				NA	-98.5	-91.0
F					NA	-87.5
DG						NA

→ 下一步合并的邻居是 C 与 (DG)

(5) 重复步骤1和步骤2，得到新的距离矩阵

(6) 重复步骤3，计算分支长度

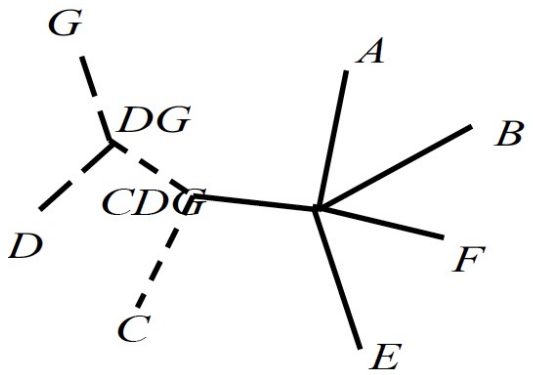
公式：

$$d_{i,(ij)} = \frac{1}{2}d_{ij} + \frac{1}{2}(r_i - r_j)$$

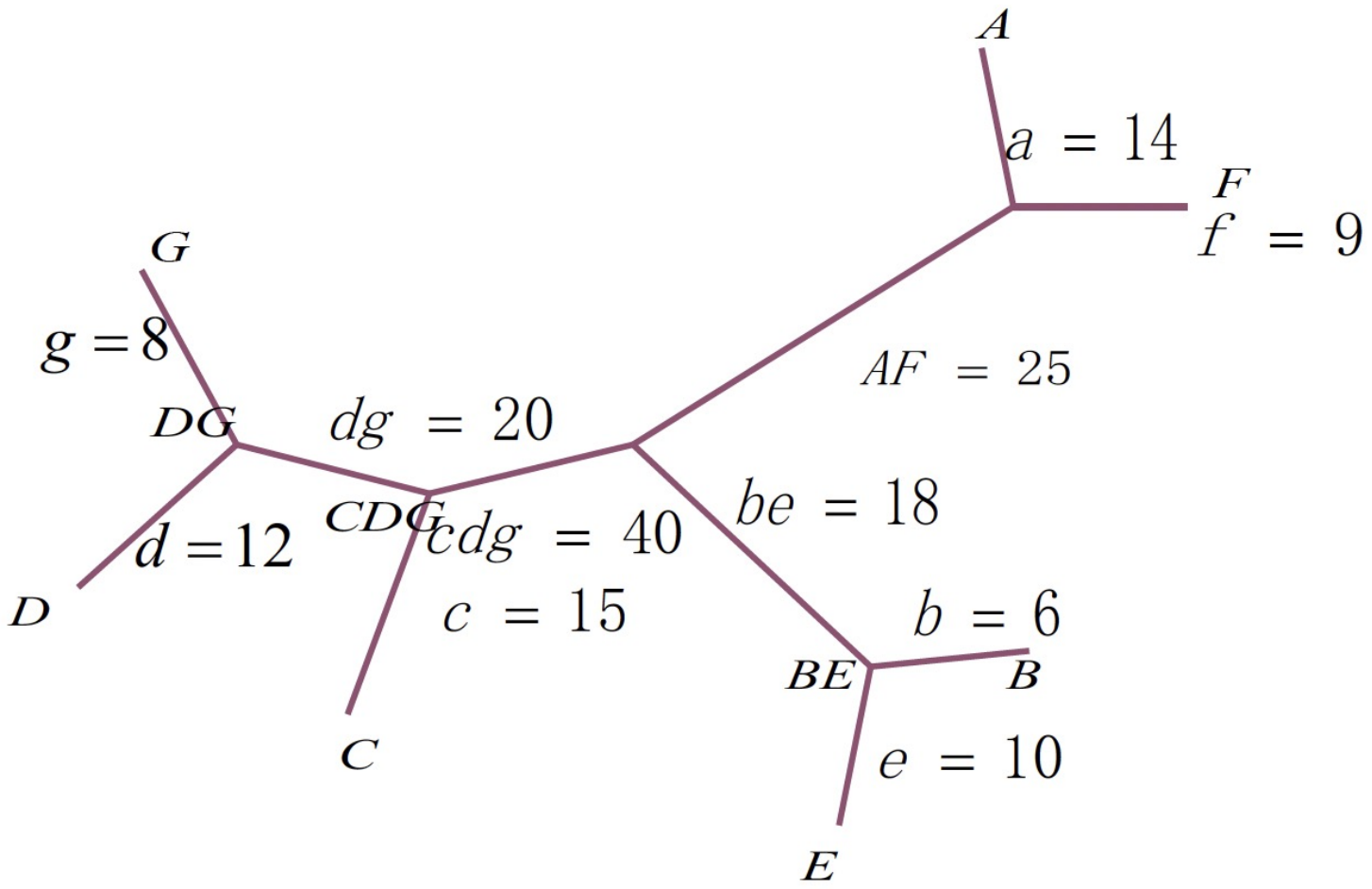
代入 $i = C, j = (DG)$:

$$d_{C,(CDG)} = \frac{1}{2}(35) + \frac{1}{2}(95 - 100) = 17.5 - 2.5 = \boxed{15.0}$$

$$d_{(DG),(CDG)} = 35 - 15 = \boxed{20.0}$$



Neighbor Joining Algorithm



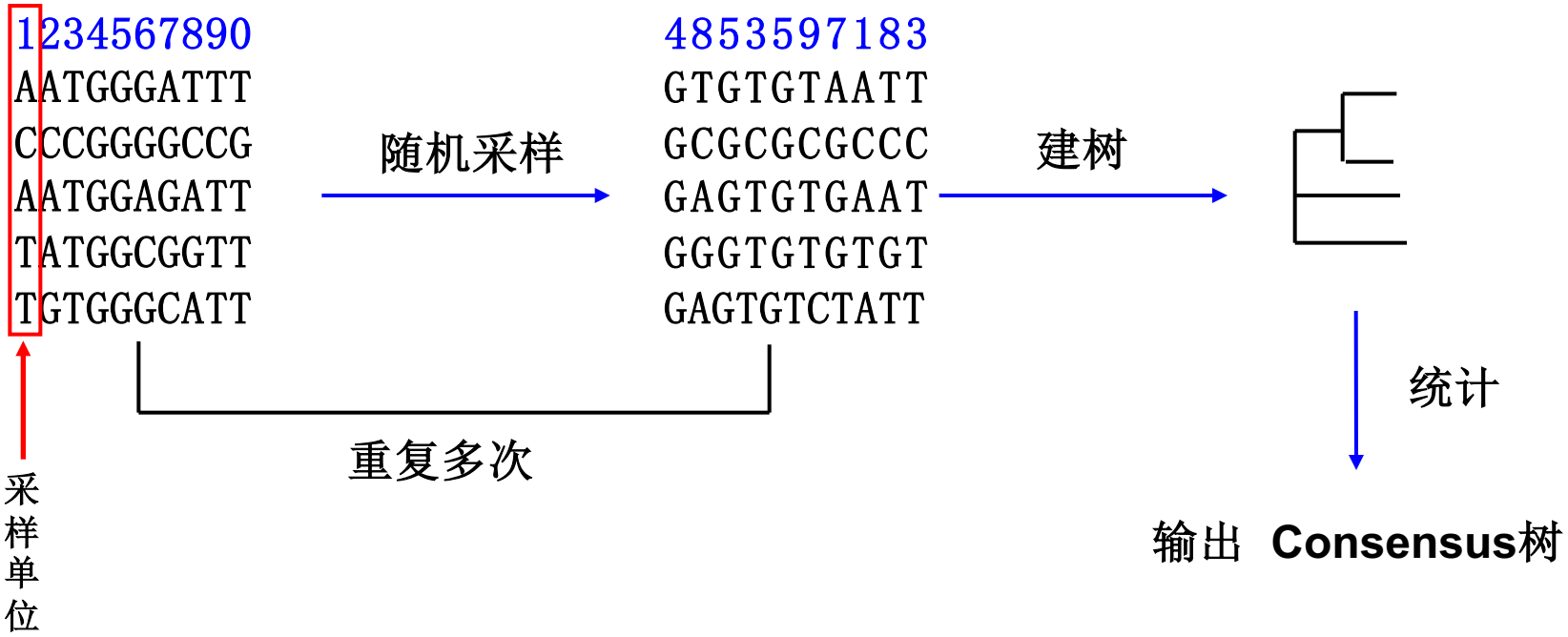
如何检验你的树

- 当你建好一棵树后，你只是得到了这棵树的分枝情况（有时也会有枝长），但是这些分枝情况的可靠性有多高呢？我们不知道。通常情况下，我们必须对我们所得的树进行分枝可靠性检查，为了达到这个目的，我们可以采用**BOOTSTRAP**法（自展法），这是一种被广泛采用的对系统树进行检测的方法。

- BOOTSTRAP的策略是，从一个数据集中随机采样，将采得的数据重新构建一个与原来一样大的数据集，然后根据这个新产生的数据集建树，这个过程重复多次（一般为100次），所以也就产生了100棵树。计算机对这100棵树进行统计，统计每一种分枝情况在各棵树中出现的次数，最后将出现最多的分枝情况进行组合，输出最终的统计树（也称Consensus树）。如果Bootstrap的重复采样数为100，一种分支情况在100次中出现了90次，那么当然要比那些只出现了20次的分枝来的可靠，同时我们就称这个分枝有90%的Bootstrap值支持。

- Bootstrap检测的采样次越多，分析的系统误差就会越小，但由于每一个采样都要重新建树，所以采样次数太多会大大降低速度，所以采样数一般在100到1000之间。

Bootstrap



BOOTSTRAP检测的基本流程

一般来说, **Bootstrap>90%, strong; bootstrap 70-90%, high; bootstrap 60-70%, moderate; bootstrap 50-60%, weak.**

基于HiF1A基因构建的系统发育树