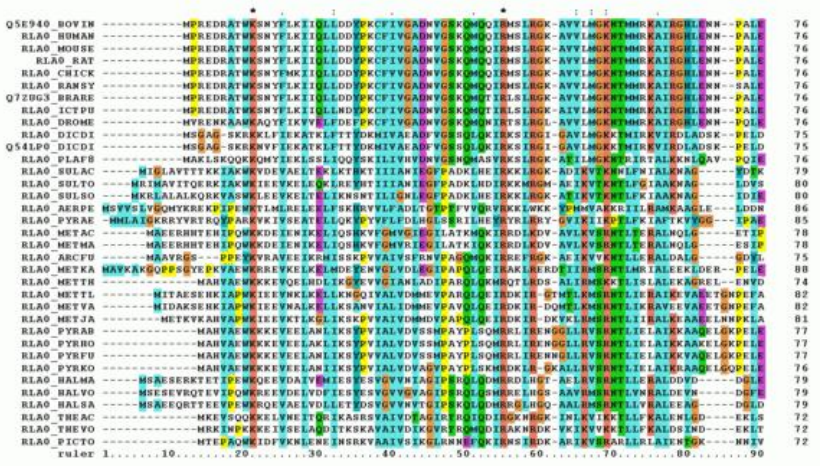
生物信息学：
组学时代的生物信息数据挖掘和理解

2020年秋



有关信息

- 授课教师：宁康，张礼斌，陈鹏
  - Email: ningkang@hust.edu.cn
  - Office: 华中科技大学东十一楼504室
  - Phone: 87793041, 18627968927

- 课程网页
  - http://www.microbioinformatics.org/Bioinformatics.html
  - QQ群：

课程安排

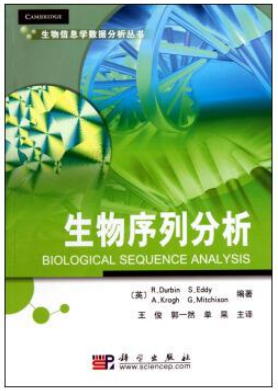
（生物信息中的算法设计与概率统计模型）

- 生物背景和课程简介
- 生物信息学和生物数据挖掘
  - 生物数据的格式及其意义
    - 序列数据
    - 树状数据
    - 网络数据
    - 表达数据等
  - 生物数据库及其用法
  - 生物信息基本算法
    - 双序列联配
    - 多序列联配
    - 基因组组装算法
    - 基因预测和功能注释
    - 系统发育树构建
    - 蛋白质结构预测
    - 生物调控网络解析
  - 组学数据分析方法
    - 基因组变异分析
    - 基因表达和比较分析
    - 非编码RNA分析
    - 蛋白组分析
    - 宏基因组分析
  - 系统生物学与交叉科学
- 面向生物大数据挖掘的深度学习

研究对象：
生物序列，
进化树，
生物网络，
基因表达
…

方法：
生物计算与生物信息

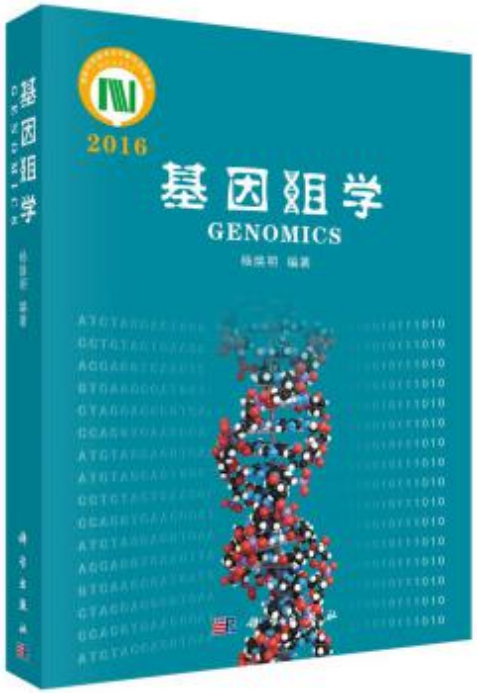
教材及参考书目

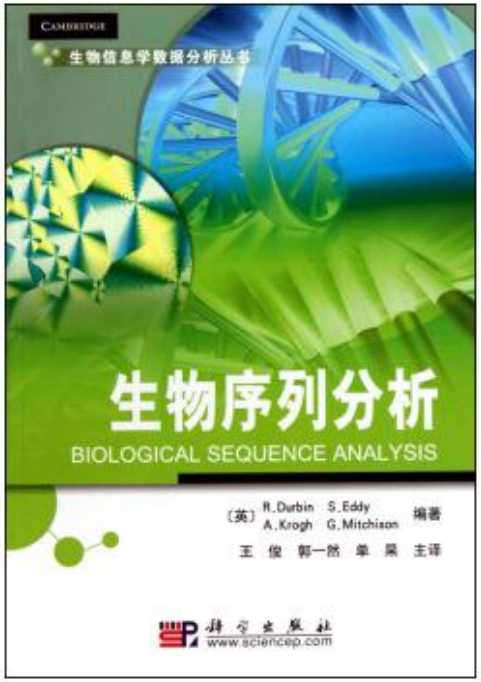


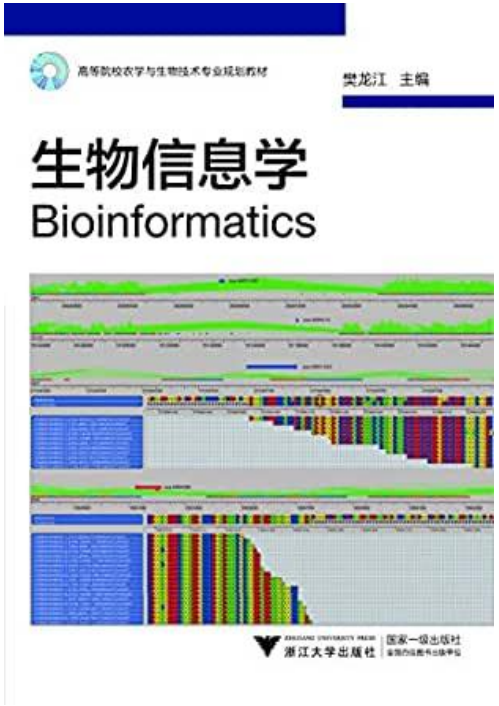
- **教学参考书：**
- 《生物序列分析》（第1版）.科学出版社. 2010年8月出版. R. Durbin等编著，王俊等主译.

- **课外文献阅读：**
- 《生物信息学》（第1版）. 浙江大学出版社. 2017年3月出版. 樊龙江主编.
- 《基因组学》（第1版）. 科学出版社. 2016年10月出版. 杨焕明主编.

References


2016
基因组学
GENOMICS


CAMBRIDGE
生物信息学数据分析丛书
生物序列分析
BIOLOGICAL SEQUENCE ANALYSIS
R.Durbin S.Eddy
A.Krogh G.Mitchison
编著
王俊 郭一然 单渠 主译
www.sciencep.com


樊龙江 主编
生物信息学
Bioinformatics
浙江大学出版社

Slides credits

——生物信息学研究方法概述：北京大学生物信息中心

——生物统计学：卜东波@中国科学院计算技术研究所，邓明华@北京大学

——神经网络与深度学习: 邱锡鹏@复旦大学

——Introduction to Computational Biology and Bioinformatics: Xiaole Shirley Liu Lab@Havard University

——Combinatorial Methods in Computation Biology: Ken Sung Lab@NUS

——Deep Learning in the Life Sciences: MIT

Sequence alignment

- 特点：天然的形式化
  - 碱基：A,C,T,G四种
  - 常见氨基酸：20种
- 目标：
  - 以DNA序列作为源头
  - 揭示“基因组信息结构的复杂性及遗传语言的根本规律”；
  - 之后进行蛋白质结构和功能预测。

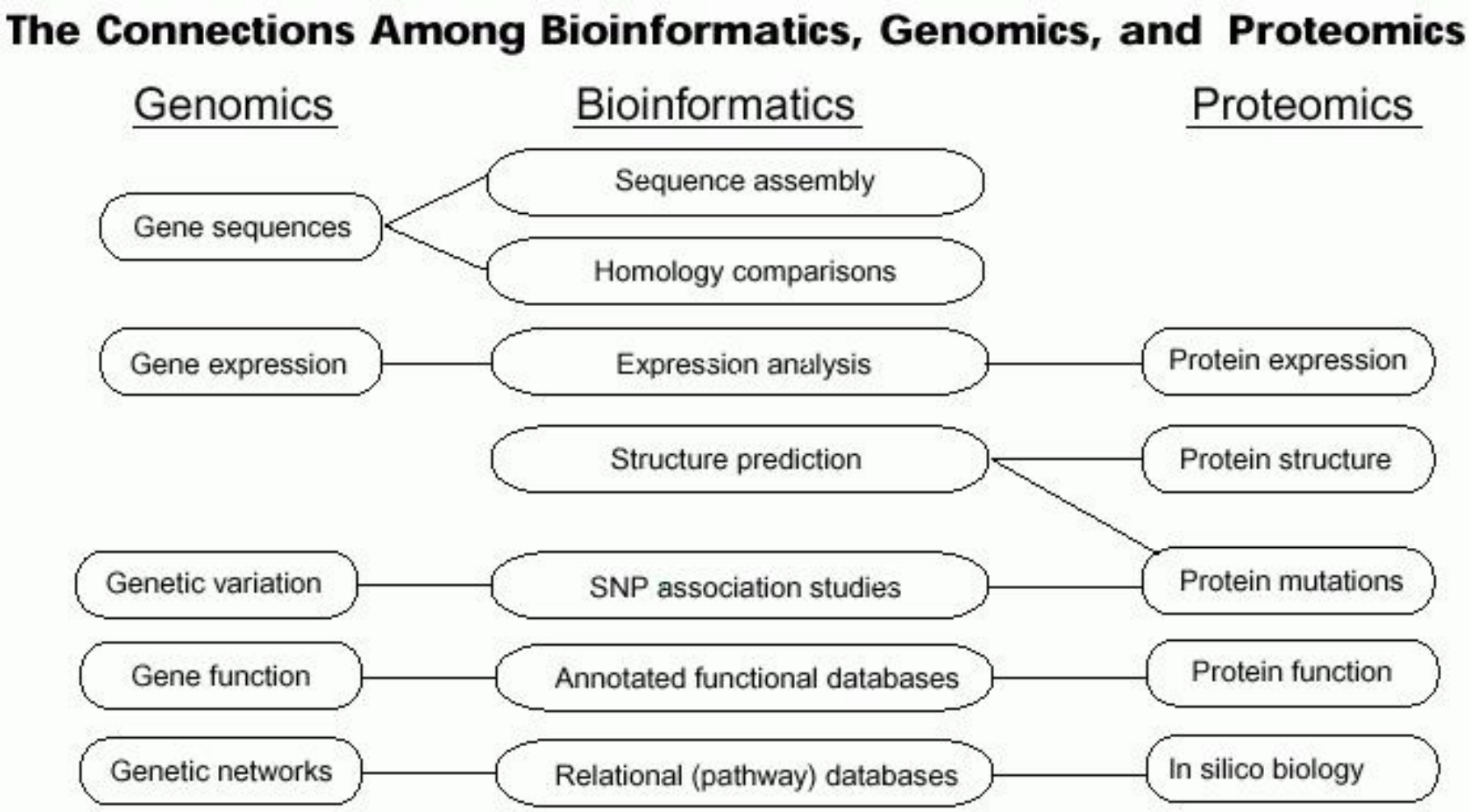
生物信息学的两个挑战

- 高性能计算：
  - 海量的数据
  - 每14个月翻一番
- 算法：
  - 海量的数据使得原有算法不适用
  - 新需求

生物信息学的研究流程

- 第一步：生物学问题的提出
  - 生物学为主
- 第二步：数学建模、算法设计
  - 信息科学为主
- 第三步：结果解释、实验验证
  - 生物学

生物信息学脉络


The Connections Among Bioinformatics, Genomics, and Proteomics
Genomics
Bioinformatics
Proteomics
Gene sequences
Sequence assembly
Homology comparisons
Gene expression
Expression analysis
Protein expression
Structure prediction
Protein structure
Genetic variation
SNP association studies
Protein mutations
Gene function
Annotated functional databases
Protein function
Genetic networks
Relational (pathway) databases
In silico biology

生物信息学问题概览（1）

- 基因组时期：序列—结构—功能
  - DNA测序和拼接
  - 比对
  - 进化树
  - 蛋白质质谱鉴定
  - 序列注释：基因预测、细胞定位
  - 结构预测：RNA结构预测、蛋白质折叠
  - 。。。。。。

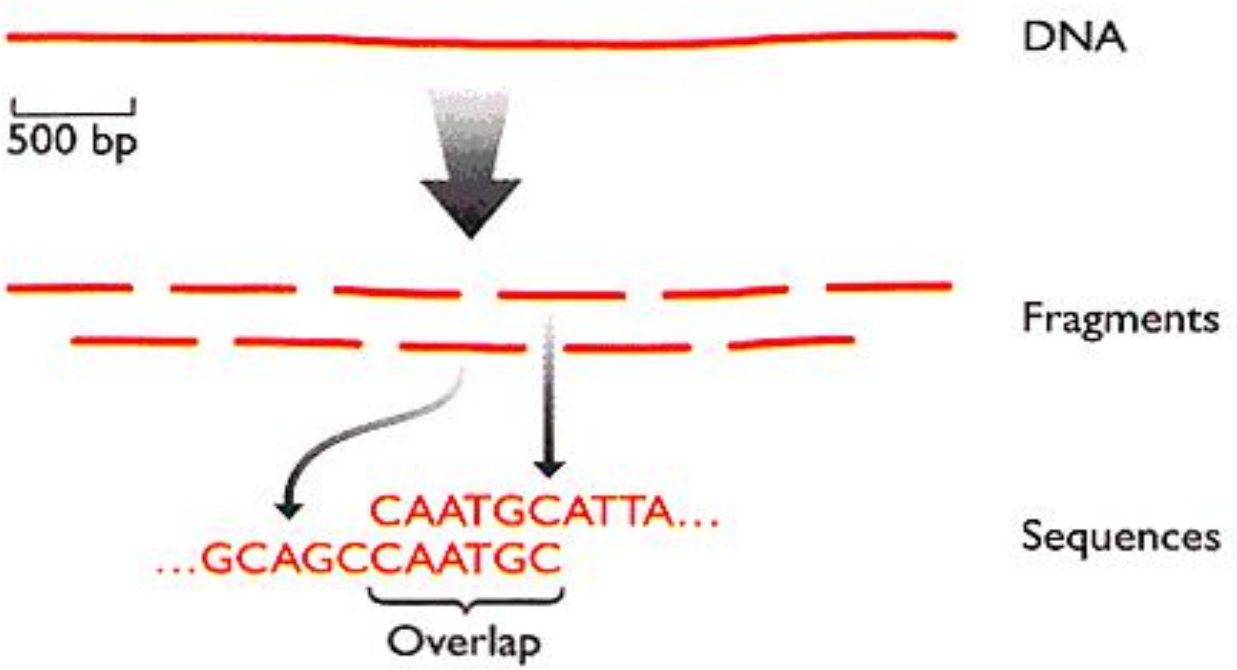
生物信息学问题概览（2）

- 后基因组时期：相互作用－网络－功能
  - 生物芯片（DNA芯片、蛋白质芯片）
  - 相互作用网络
  - 调控网络
  - E-Cell
  - 药物设计
  - 。 。 。 。 。 。

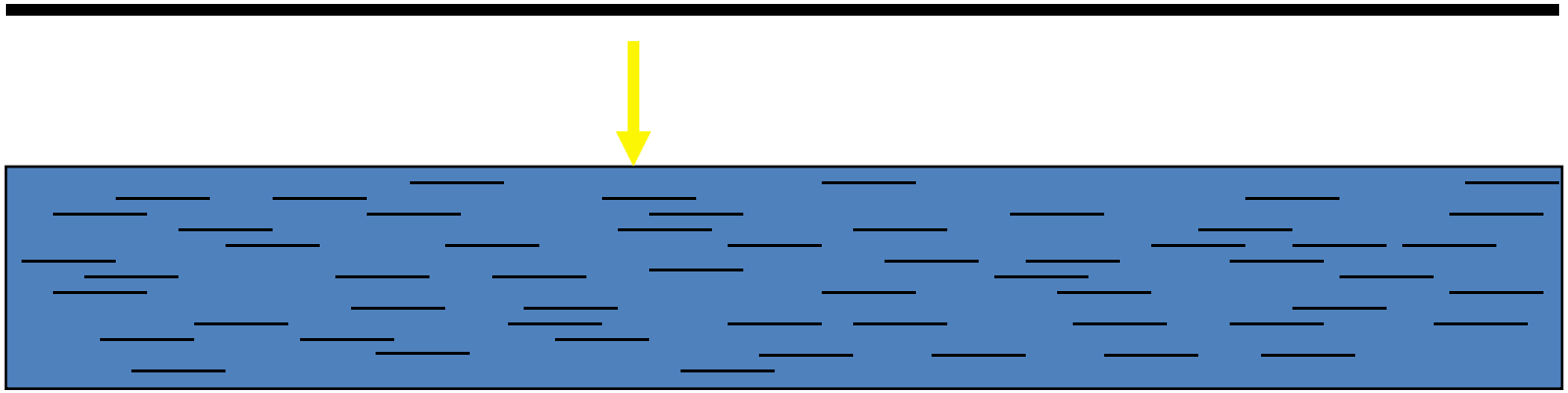
1. 大规模测序和拼接

生物学问题：

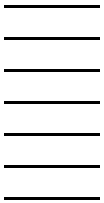
– 从DNA片段恢复原始序列



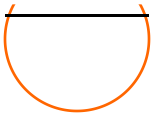
DNA整体



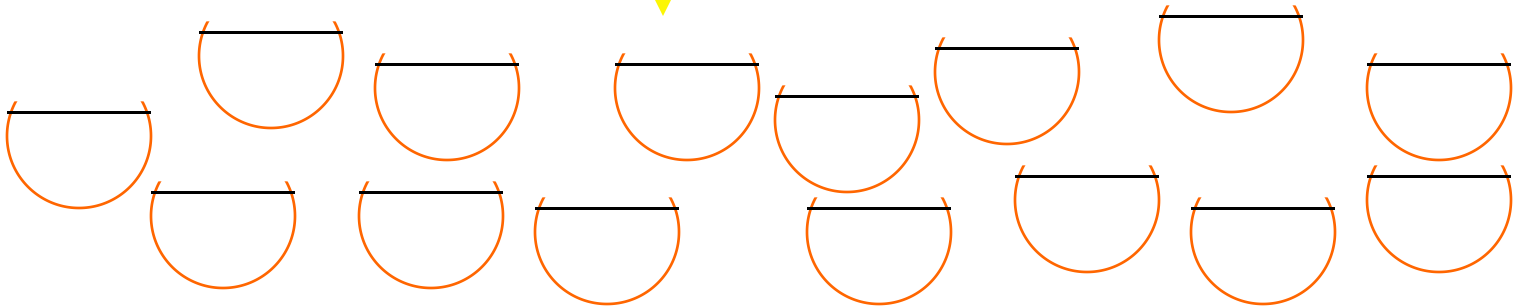
切成小段



小段和载体结合



结合后进行测序



全自动的测序仪器： MegaBace

需要拼接！

- 因为整个基因组太长（上M),而每次只能测得一个500的小片断(read)
- 问题：如何根据read恢复原始顺序？
- 类比：10本圣经，都从随机点起始剪成500个字母左右的小纸条，问：给你这么一堆小纸条，你能读出圣经来吗？

拼接问题的数学描述

- 数学问题：
  - 公共超串
- 输入：设有字符串S，预先估计其长度大约为n，现在已知一个字符串集合R={R_1,R_2…R_n}，其中每个R_i都是S的一个子串。问：原始序列S是什么？
- 算法：
  - Hamiltion路径类
  - Euler路径类
  - Local Search类

2. 序列比对

- 生物学问题：
  - 序列的相似性—>同源性
- 原始序列：

  S: acgctg

  T: catgt

可行解：

1. S: a c g c t g -
   T: - c – a t g t
2. S: a c g c t g -
   T: - c a – t g t
3. S: - a c g c t g
   T: c a t g - t -

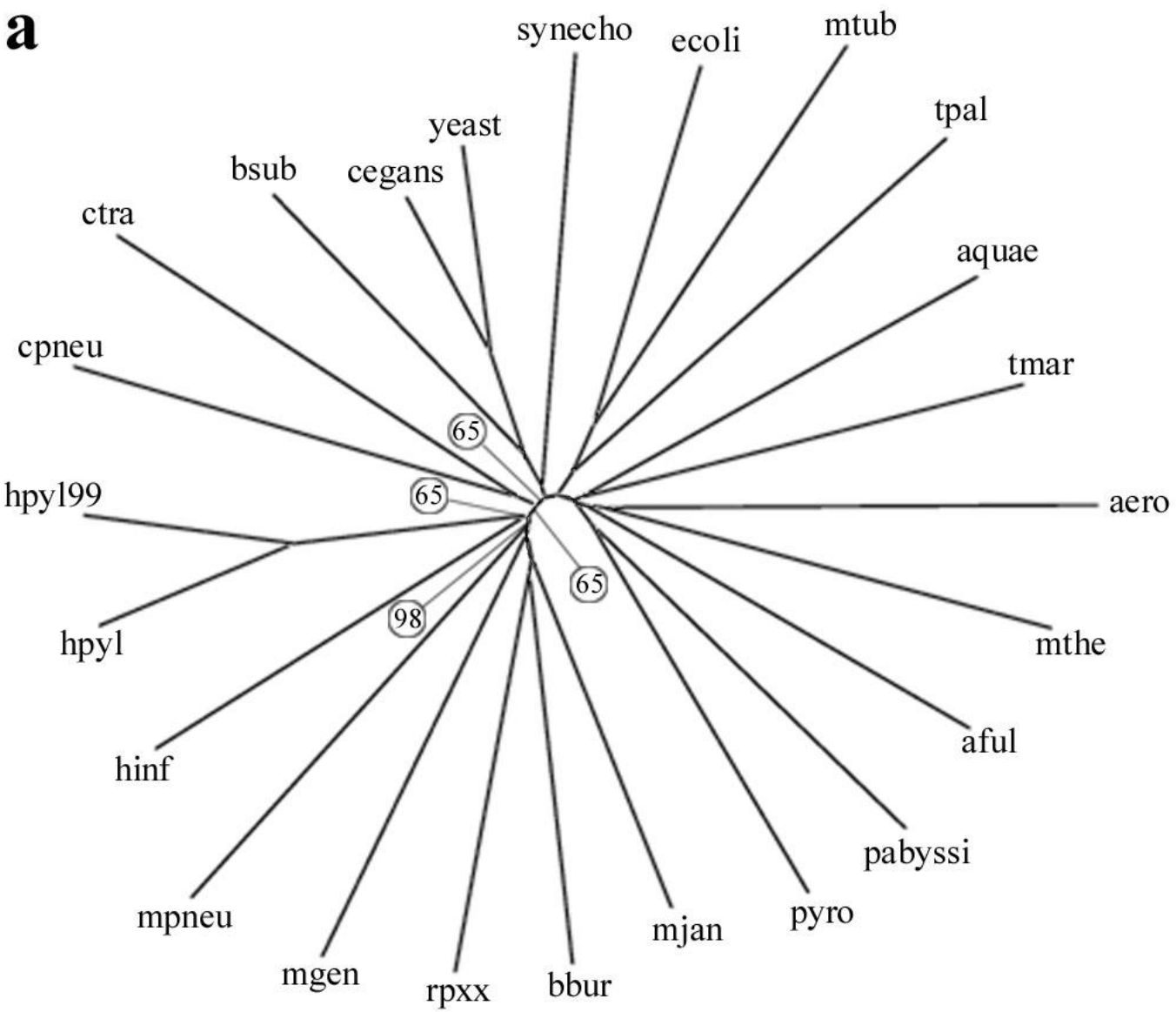
序列联配：

- 两序列联配：
  - 全局联配(Global Alignment)
  - 局部联配(Local Alignment)
  - 空位处罚(Gap Penalty)
- 多序列联配
- 全基因组比对

Open Problems:

- 快速的多序列比对算法
- 快速的全基因组比对算法

- 生
  -
  -

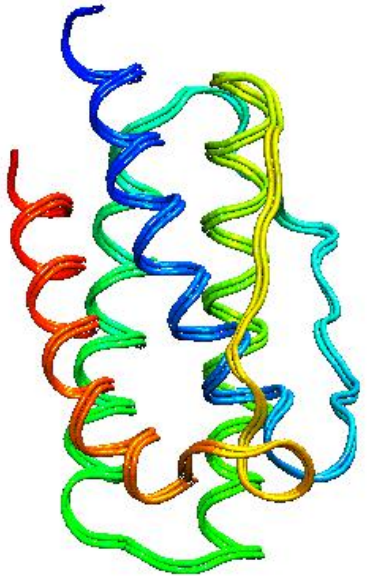

a
synecho
ecoli
mtub
tpal
yeast
bsub
cegans
ctra
aquae
cpneu
tmar
65
hpyl99
65
aero
65
98
hpyl
mthe
aful
hinf
pabyssi
mpneu
mjan
pyro
mgen
rpxx
bbur

进化树问题的数学描述

- 输入：
  - N个物种的特征（DNA、形态。。。）
- 输出：
  - 以这N个特征为叶节点的一颗树
  - 距离法：
    - 聚类谱系树
  - 简约法：
    - 最小突变树

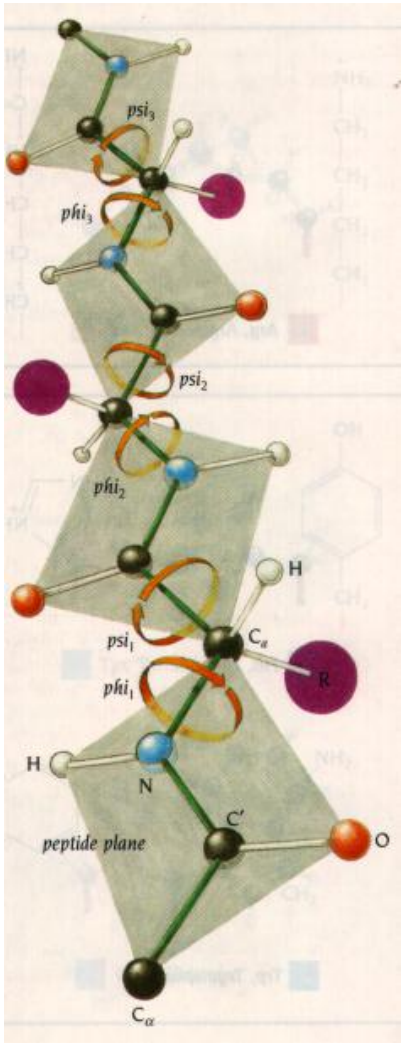
4. 结构预测

- 结构大致决定功能
  - 一级结构 (氨基酸序列)
  - 二级结构 (螺旋、片层、回环)
  - 超二级结构（aba…)
  - 三级结构 (由二级结构组合成三维构像)
  - 四级结构：多个亚基
- 实验测定方法：
  - x-ray晶体衍射
  - NMR核磁共振
- 实验耗时、昂贵
  - 一个蛋白质结构测定需要$200K or more
  - 需数月或者更长
  - 有些蛋白质还无法测定



蛋白质结构（2）

- 理论上可计算的。
  - 能量最低原则
  - 变元：
    - 主干的psi/phi angles
    - 侧链的旋转
- 优化问题，但是
  - 搜索空间极其巨大
  - 局部极值点



三种预测方法

- *ab initio* 方法
  - 根据第一原理
  - 计算量极大，实际上不可行
- 同源建模方法：
  - 基本假设：序列同源—>结构相似
  - 有效，但是必须具有同源的序列
- Threading方法：
  - 基本假设: 自然界中蛋白质主链模式是有限的
  - **~90% 新蛋白质和PDB某个已知蛋白质结构相似**
  - 推论: 多个蛋白质会具有相同的主链模式
  - 预测问题—>识别问题
  - 能够处理序列上不相似，但是结构相似的情况

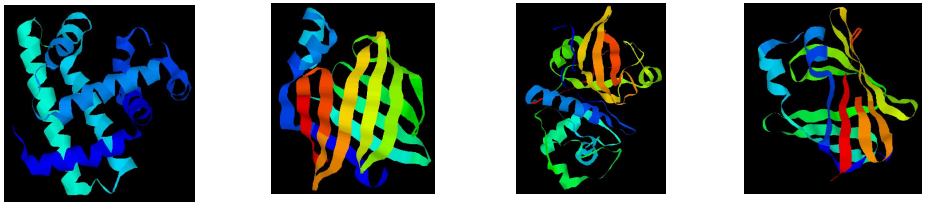
Threading方法

- 思路：
  - 将序列尽可能好地放入结构模板中；
  - 设计评价函数，对匹配情况进行打分；
- 关键：
  - 已知的结构模板库
  - 衡量匹配情况的打分函数
  - 寻求最优的算法；

序列：

MTYKLILNGKTKGETTTEAVDAATAEKVFQYANDNGVDGEWTYTE

模板库：

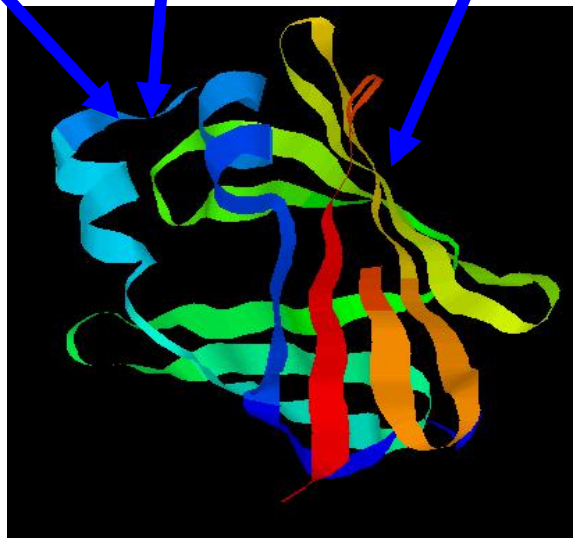


数学描述：

MTYKLILNGKTKGETTTEAVDAATAEKVFQYANDNGVDGEWTYTE

两个残基相邻的衡量:
E_p

空位罚分: E_g

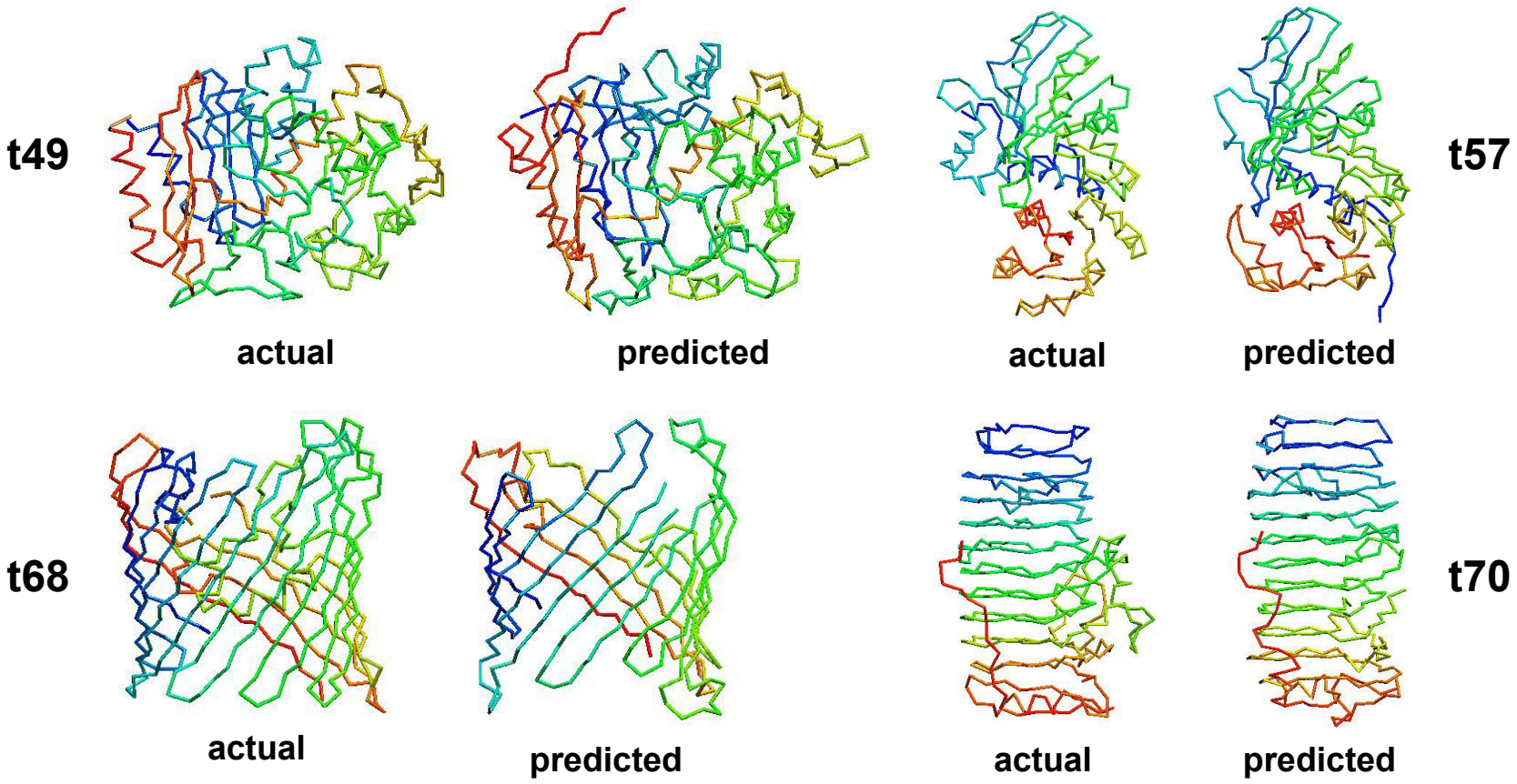


残基和结构的匹配：
environment: E_s

min （ E_p + E_s + E_g ）

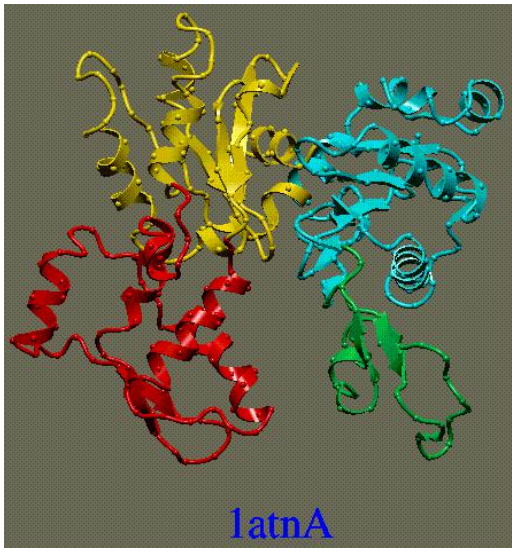
Protein Threading by PROSPECT

- prediction examples from CASP3 contest



5. 蛋白质DOMAIN识别

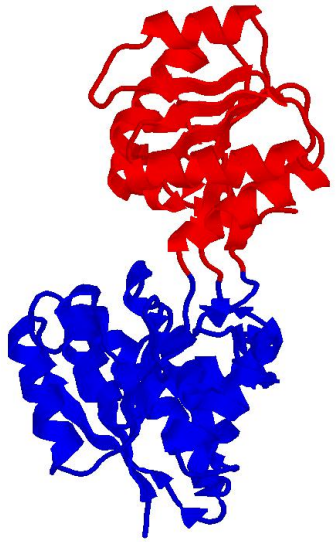
- 生物学观点：
  - 一个蛋白质结构可以包含多个DOMAIN:
  - DOMAIN是蛋白质折叠、功能和演化的基本单位
  - 不同的蛋白质具有相同的DOMAIN
  - 识别DOMAIN有助于蛋白质折叠
- 数学观点：
  - 最小割



actin

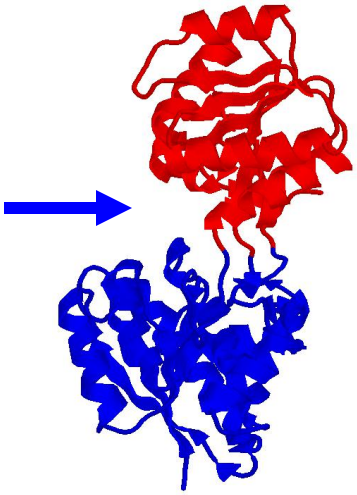
DOMAIN识别

- 生物学的不严格表述：
  - DOMAIN连接紧致，接近球状
  - DOMAIN之间作用相对较弱
- 可操作的定义：
  - DOMAIN内部残基相互作用较强
  - DOMAIN之间残基相互作用较弱
- 现有识别方法不实用
  - SCOP数据库靠手工来维护

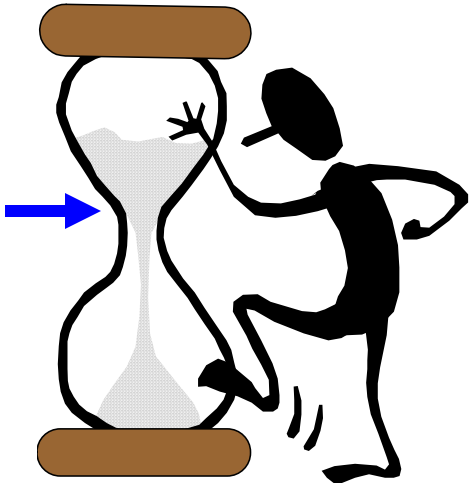


DOMAIN识别与最小割

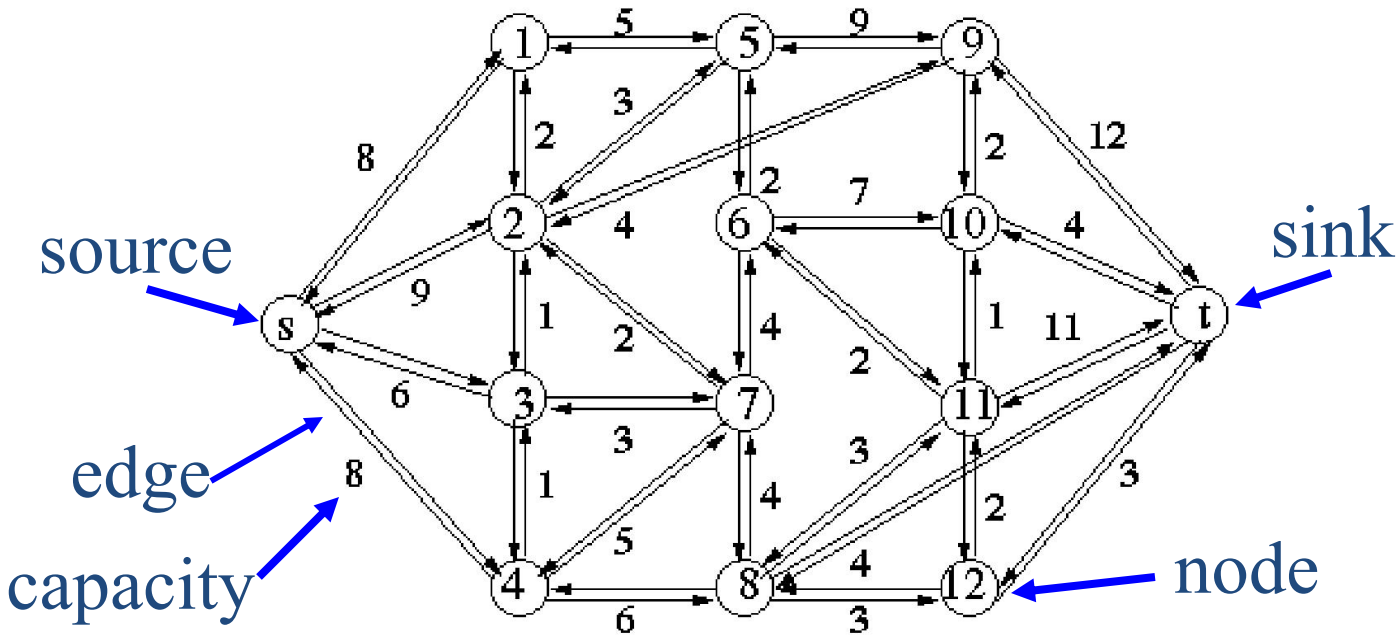
interface



bottleneck



Network Flow Problem



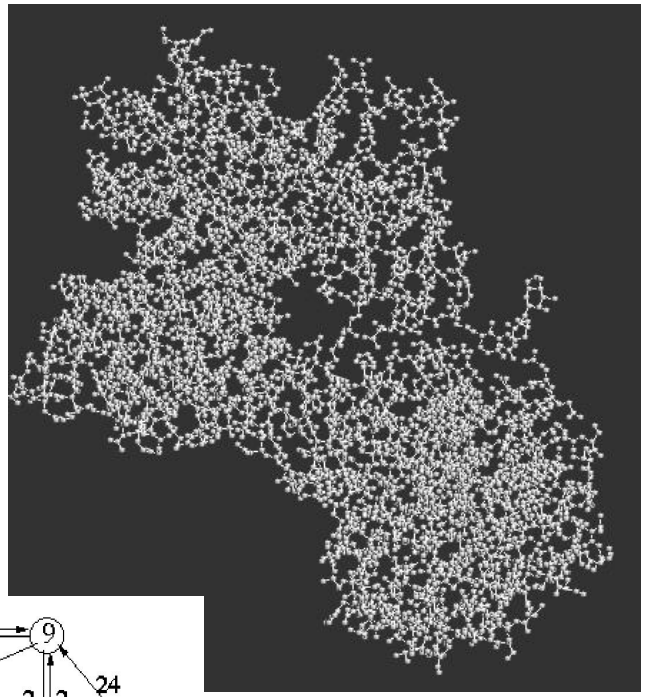
Ford-Fulkerson Theorem: the minimum cut of a network is equal to its maximum flow

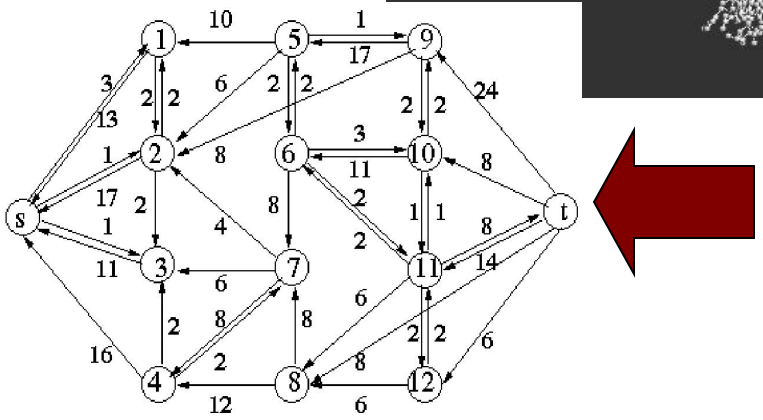
最小割

节点：一个节点表示一个残基

边：残基－残基之间的相互作用

容量：根据生物学知识，比如相互作用的种类和强度确定边的容量





经典解法及其结果：

maximum flow:

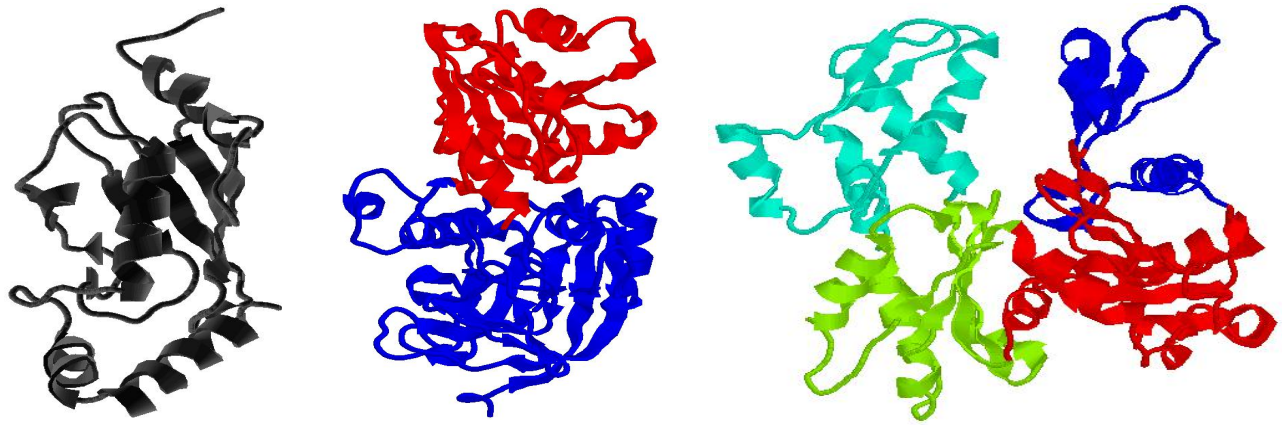
Edmonds-Karp algorithm (1972)

enumeration of all minimum cuts:

Picard-Queyranne algorithm (1980)

complexity: O (n^3)

(n is the number of residues in structure)



6. 序列注释

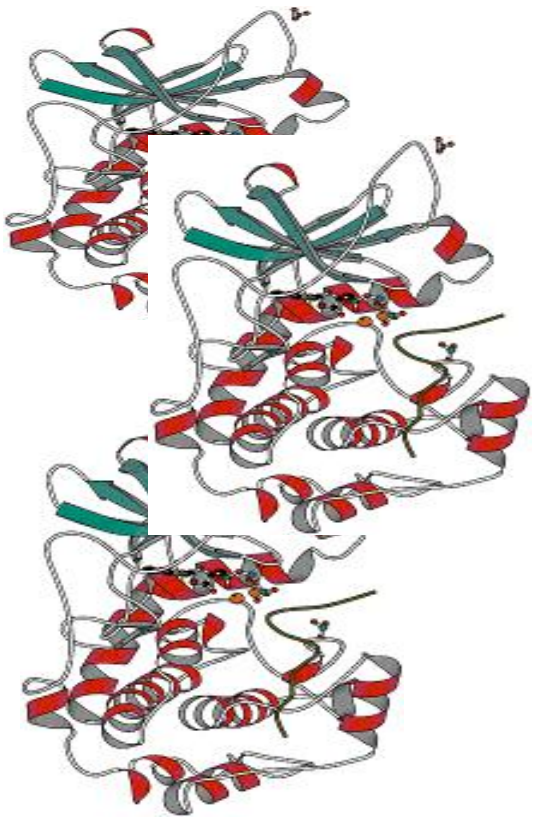
- 输入：
  - DNA序列
- 输出：
  - 各个功能位点：基因、启动子、ncRNA。。。
- 可以利用的知识：
  - 生物学规律
  - 正例和反例
- 当前最好的方法：
  - HMM
  - 形式语言

7. 蛋白质质谱鉴定

- 生物学问题：
  - 原有的Edman方法昂贵、耗时
  - 根据质谱来测定蛋白质氨基酸序列

- 什么是质谱？

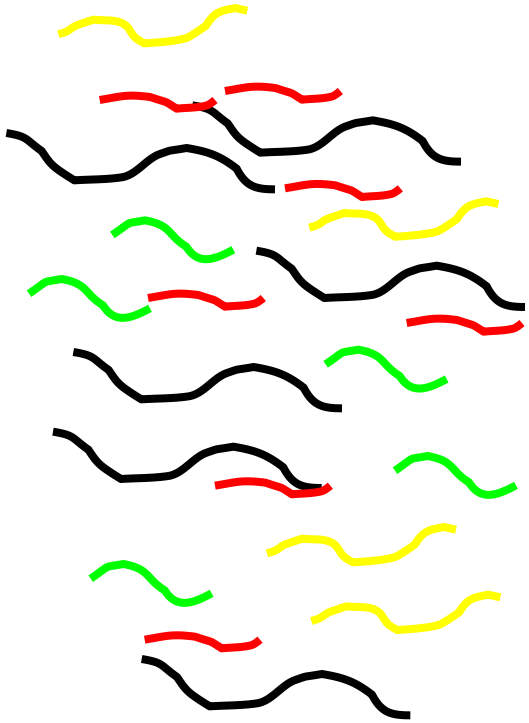
样品制备



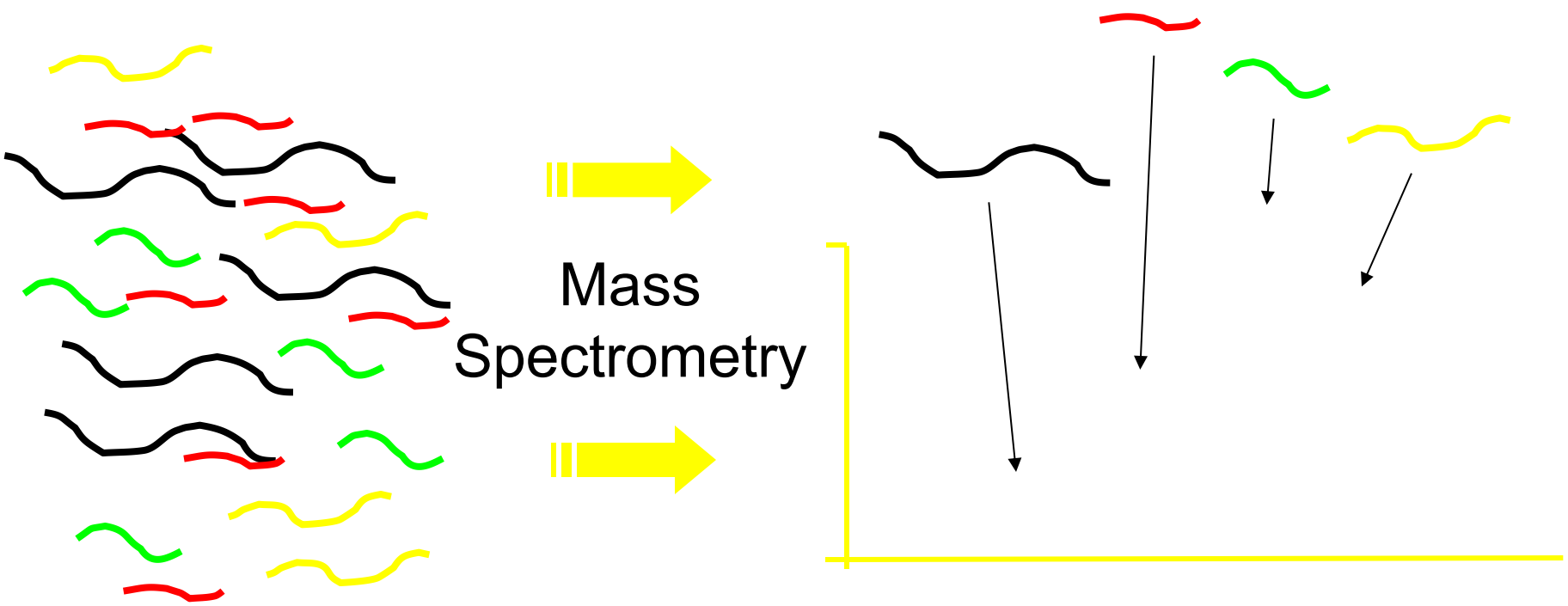
酶切

或者

物理方法切割

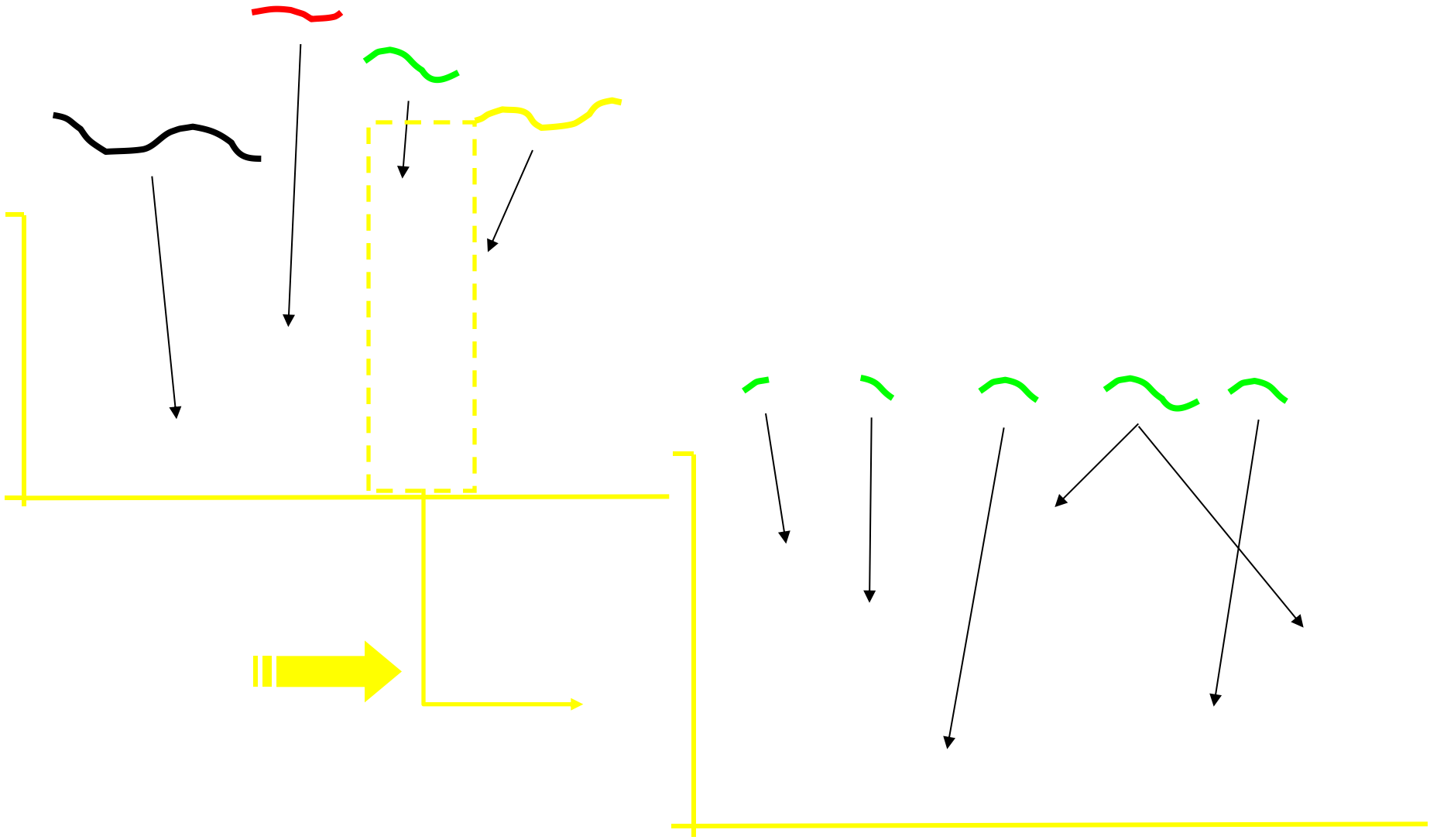


一级质谱：片段分离

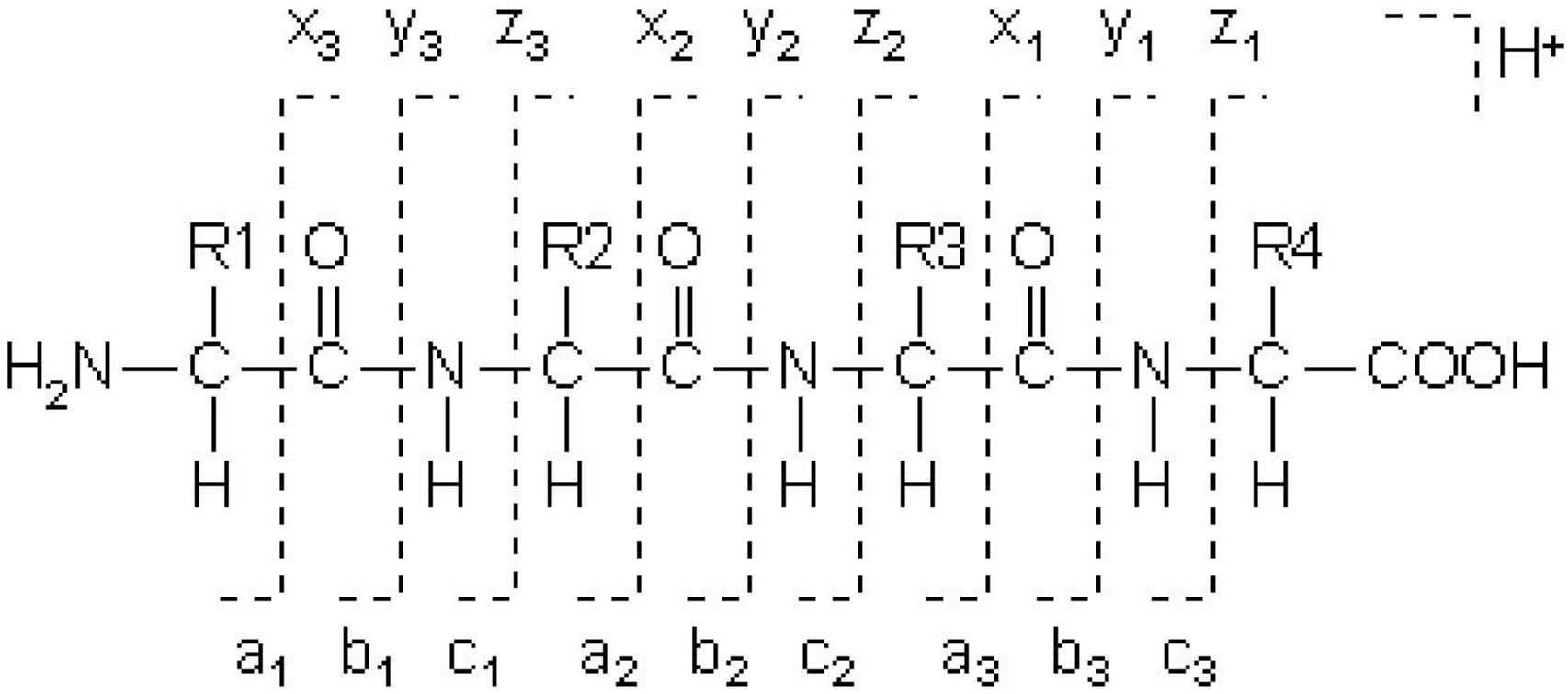


短肽段离子化，经过电场，依据荷质比的不同分离

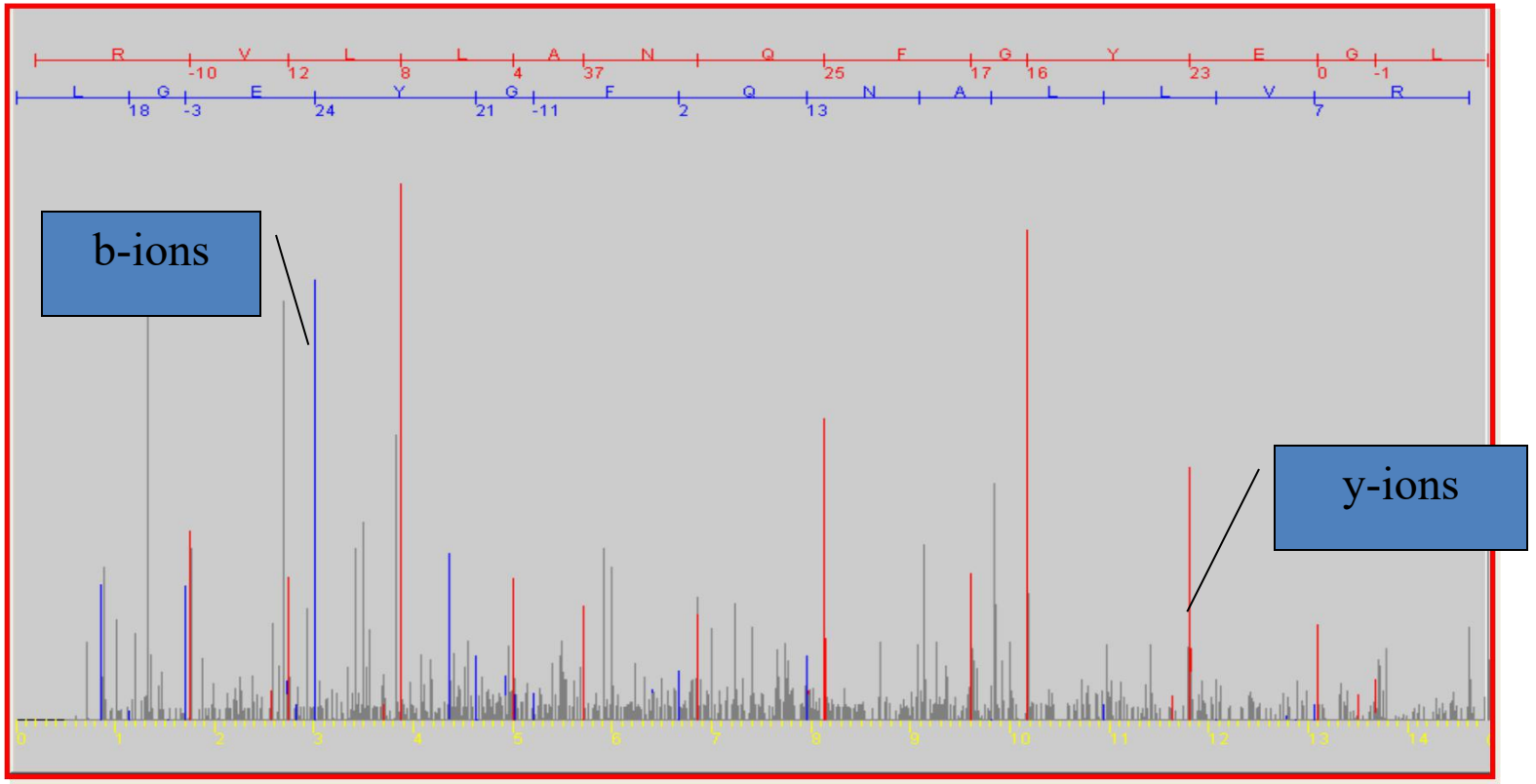
二级质谱：片段再打断



肽段可能形成的离子

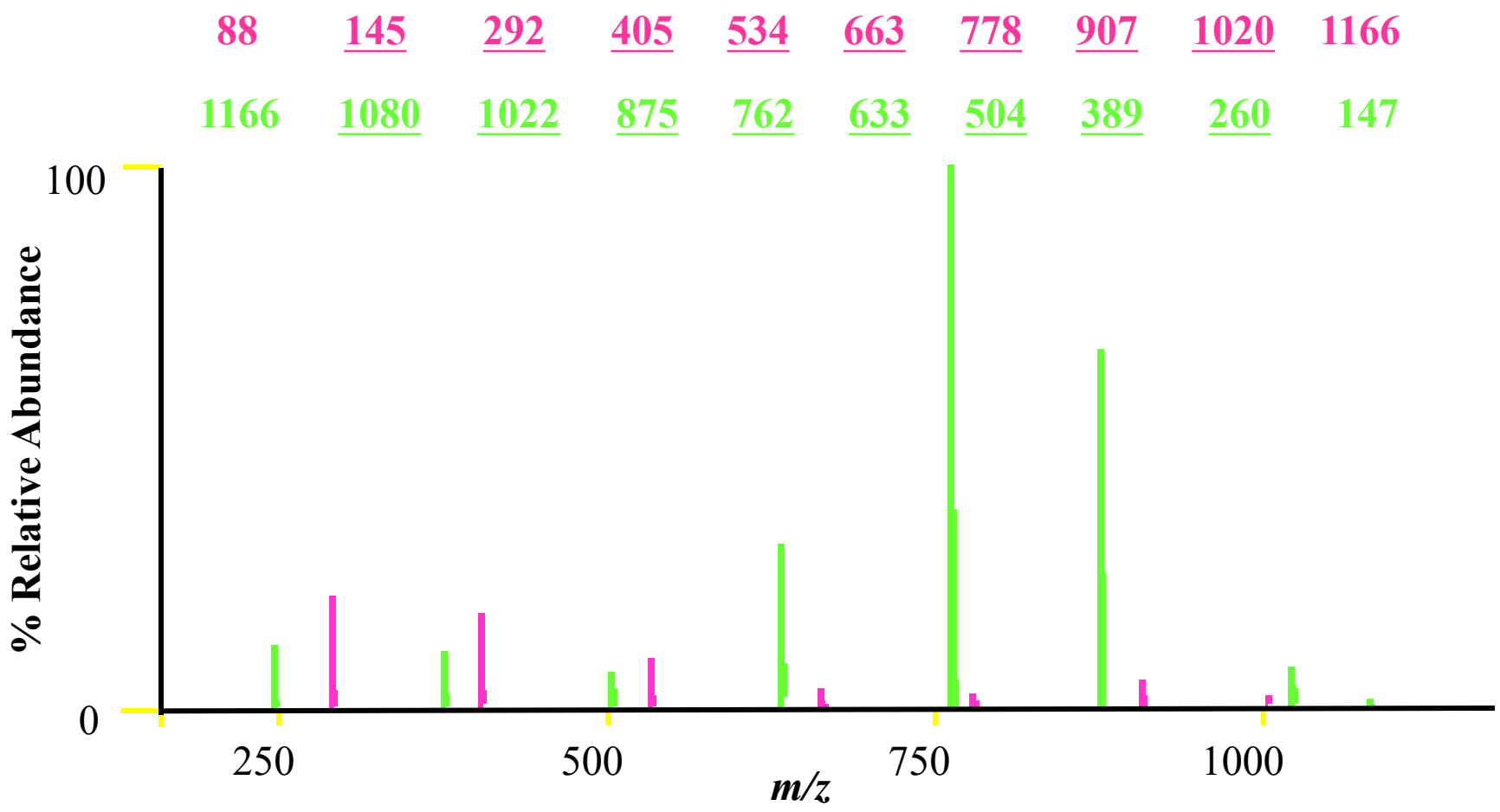

x3 y3 z3 x2 y2 z2 x1 y1 z1
H+
R1 O R2 O R3 O R4
H2N C C N C C N C C N C COOH
H H H H H
a1 b1 c1 a2 b2 c2 a3 b3 c3

MS/MS Spectrum



LGEYGFQNALLVR

DeNOVO: 从质谱猜原始序列



基于Local Search的序列拼接算法

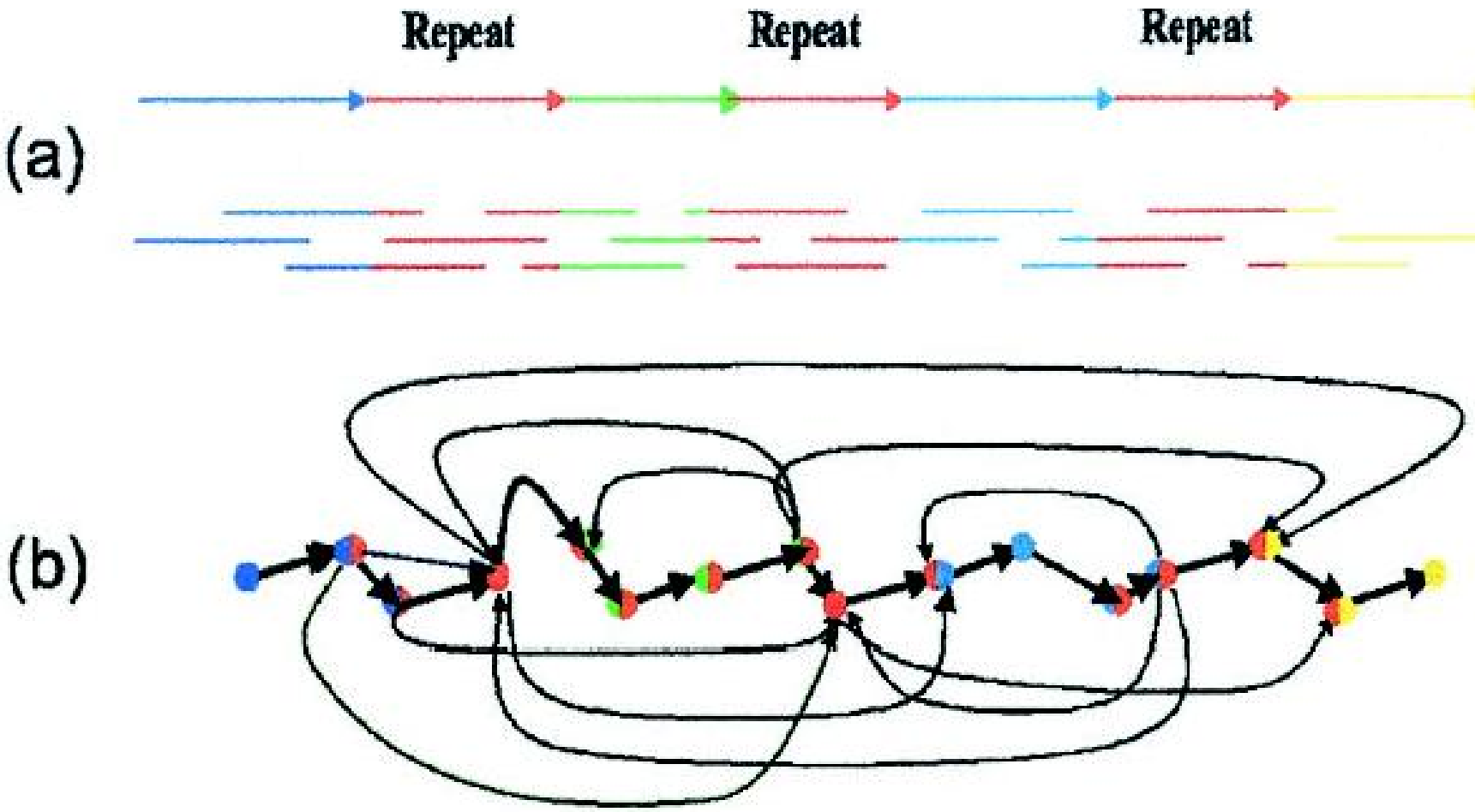
现有算法：

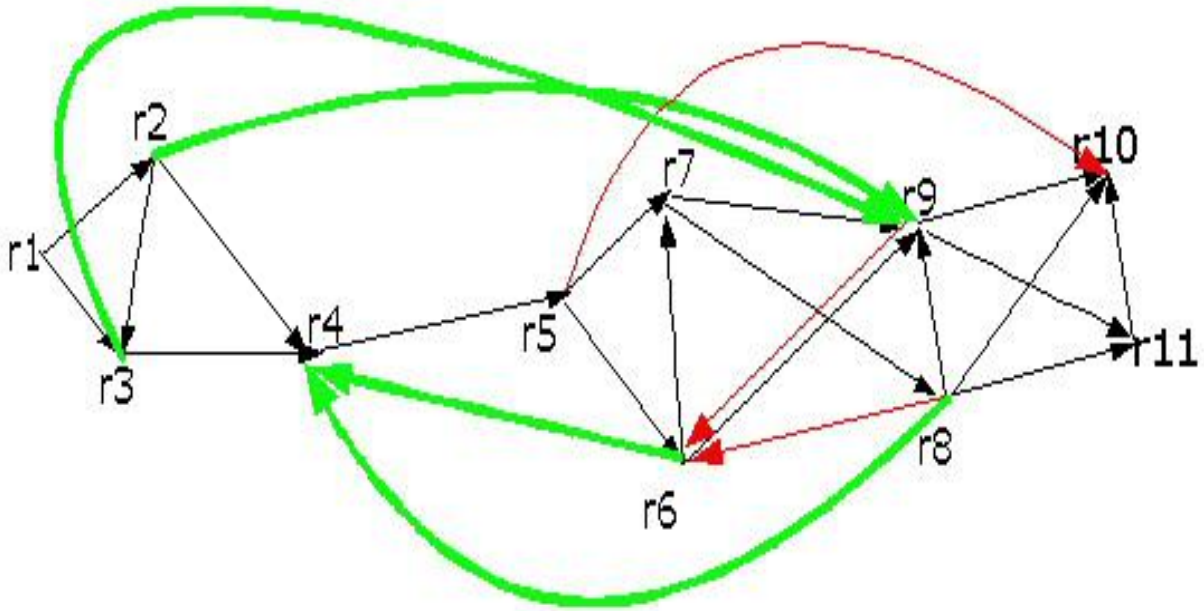
- Hamilton路径类算法
- Eulerian路径类算法

Hamilton路径类算法

- 包括： Phrap, CAP3, TIGR, GigAssembler
- 生成图：
  - 结点：每个片段自成一个结点
  - 边：如果两个结点间有Overlap
  - 沿DNA序列从头走到尾，将经过每个结点一次且仅一次
  - Hamilton Path

Hamilton Path




r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11

拼接三步曲

- STEP 1。Overlap
- STEP 2。Layout
- STEP 3。Consensus/Mosaic

STEP1。Overlap

- 这一步对所有的Read进行两两比对，通常采用快速Smith-Waterman算法，以确定两个Read之间是否有Overlap。
- 考虑到各个碱基的出错概率，常常对Overlap进行打分，衡量Overlap的可能性高低，一般采用LLR（Log Likehood Ratio）方法打分。

STEP 2。Layout

- 根据Read之间的重叠信息形成Contig，即将各个Read merge起来，形成一个逐次链接的链接体。
- 这一步实际上是在求一条Hamilton Path，通常采用的是贪心法。

STEP 3。Consensus

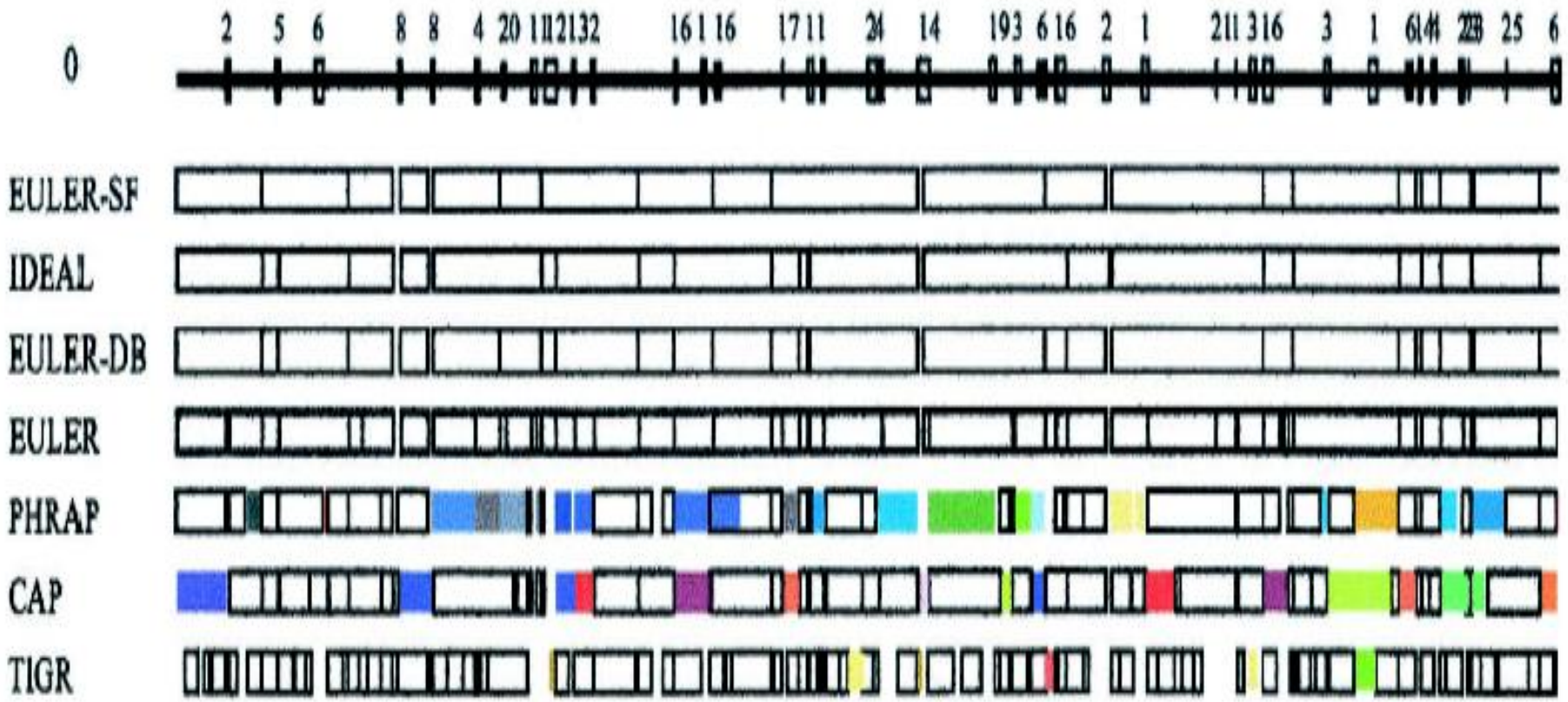
- 对于每个Contig，按照投票或者其他的原则计算出一个Sequence。

评价

- 判定是否存在Hamilton Path是NP完全问题
- 现在采用的是贪心法

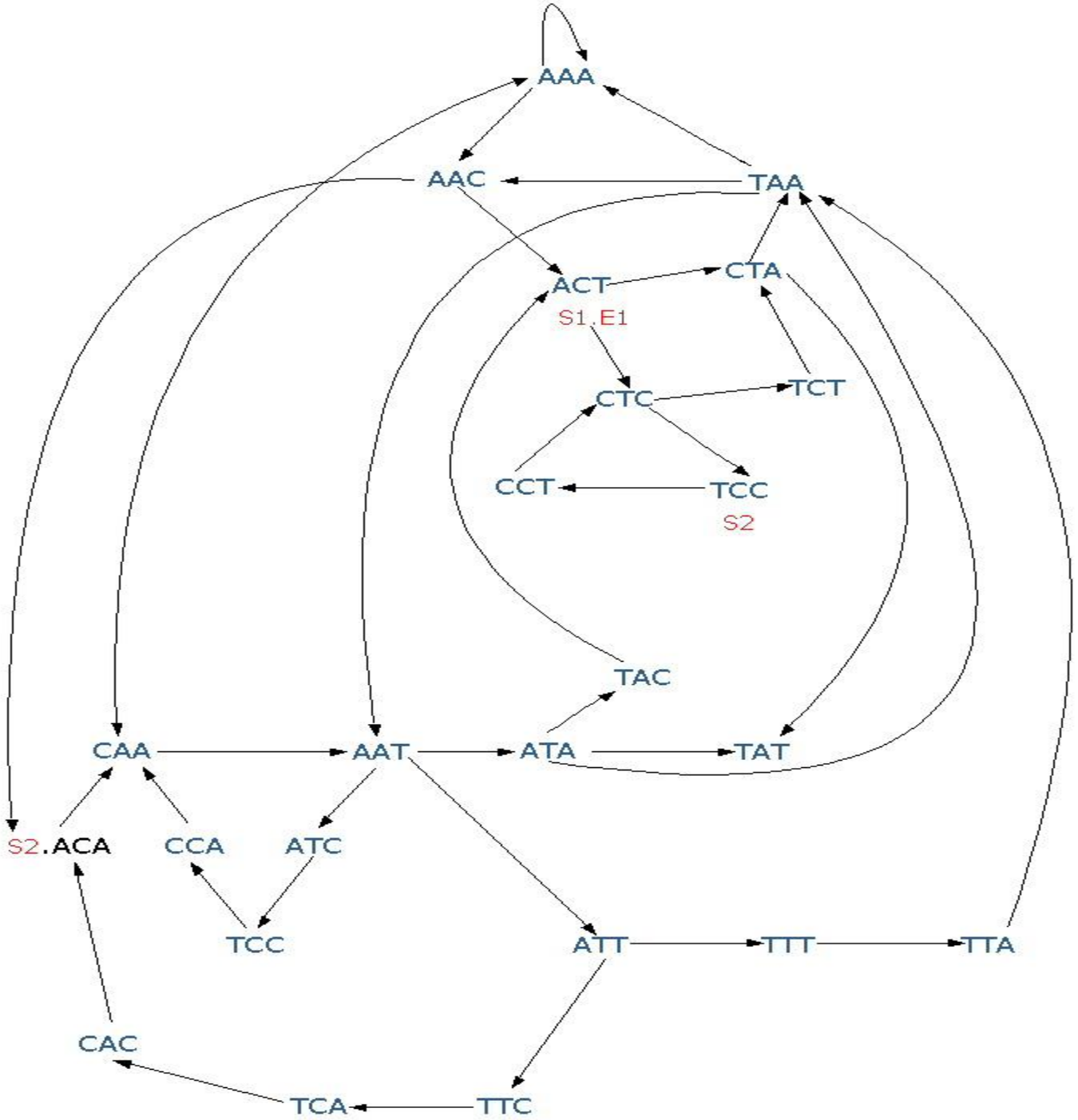
- 不存在有效算法
- 出错

出错例示


0
2 5 6 8 8 4 20 1 1 2 1 3 2 16 1 16 17 1 1 24 14 19 3 6 16 2 1 21 1 3 16 3 1 6 14 4 22 3 25 6
EULER-SF
IDEAL
EULER-DB
EULER
PHRAP
CAP
TIGR

EULER Path类算法

- 转换成图论问题 de Bruijn图
- 结点：l-mers
- 边：两个l-mers重叠（l-1)个单元
- 片段：图上一条路径
- 沿着DNA从头走到尾，将经过每条边一次且仅一次。Euler Path
- 附加条件：经过每条路径的特殊Euler Path.


AAA
AAC
TAA
ACT
S1.E1
CTA
CTC
TCT
CCT
TCC
S2
TAC
CAA
AAT
ATA
TAT
S2.ACA
CCA
ATC
TCC
ATT
TTT
TTA
CAC
TCA
TTC

STEP 1。Consensus

目的：排除Read中的错误，获得Error－Free 的Read

- 思路：使用Gk的近似－将所有的Read切割成小片k-mers
- k-mers: 是Solid如果出现在至少M个Read中；否则称为Weak。

STEP 2。de Bruijn图的构造

- 结点：每个k-mers是结点
- 边： de Bruijn边的构造方法，v-u当前仅当v的尾巴和w的头相同。
- 每个Read表示成一条Path，
- 每个Repeat表示成一个多入口、多出口的单一链，但是不知道出入口之间的对应关系。如果没有Read来覆盖这条单一链，则称为Tangle。

STEP 3. Eulerian Super Path

- 在de Bruijn图中寻找一条Eulerian Path，能够覆盖所有的Path。
- 变换方法，等价变换，使得成为寻找一条Eulerian Path的问题。

DeNOVO算法

质谱的简化模型：

- 考虑26个英文字母，
- 每个字母有权重W(c)
- 对字符串S，
- 已知质谱＝ $\{\sum_{j=0}^{i-1} W(Sj), \sum_{j=i}^{n-1} W(Sj) \mid i = 1..n-2\}$
- 问：字符串＝？

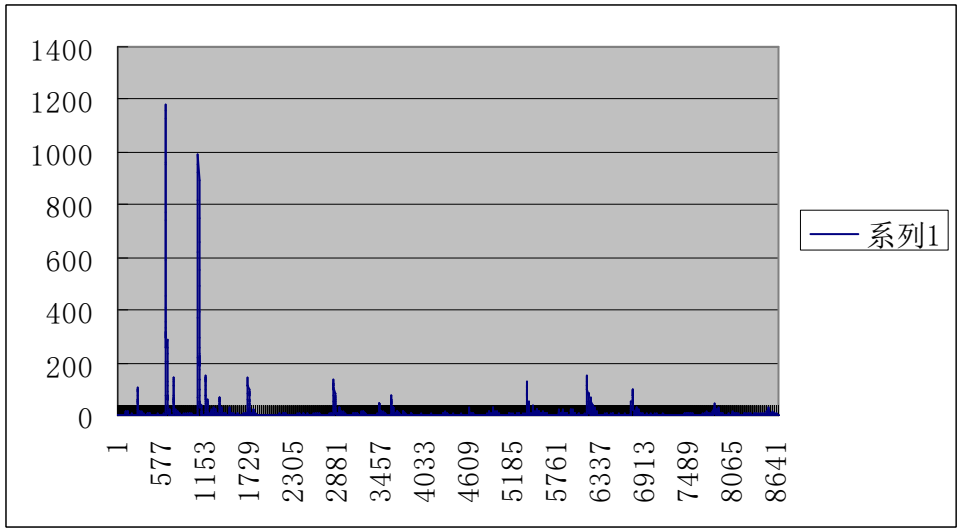
难点：

- 1。一次打断形成两个离子，混杂在一起
- 2。峰的类型不知道，是b离子,y离子还是混合体？
- 3。离子的修饰：脱氨、脱水、同位素等

组合优化问题：

- 已知谱L，
- 求序列$S=A_0A_1A_2\ldots A_n$，
- 满足：$\sum_i W(A_i) = M$
- 同时 max f(spectrum(S), L)
- 根据生物学知识确定合适的函数f

结果1:



答案:

LVNELTEFAK

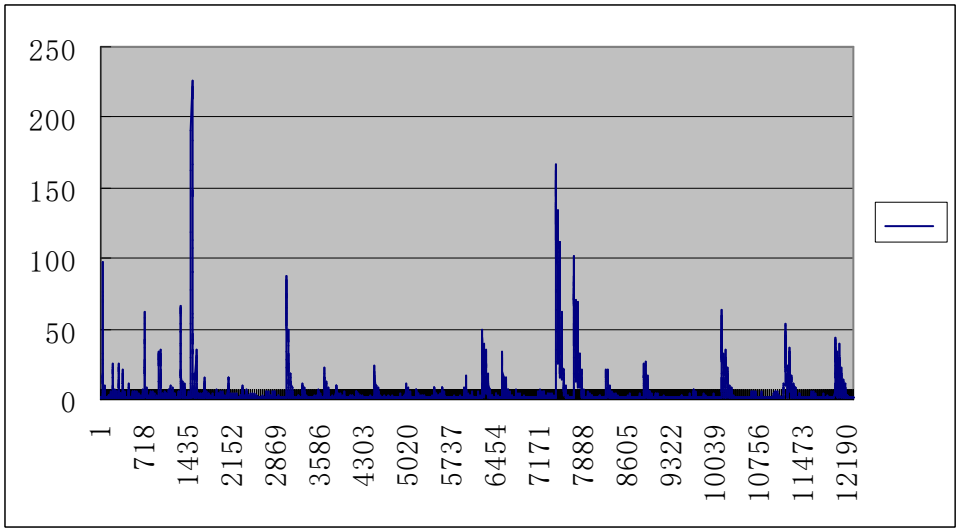
结果:

LVNELTEFAK

LVNELTHLPK

LVNELTYSPK

结果2:



答案:

HPEYAVSVLLR

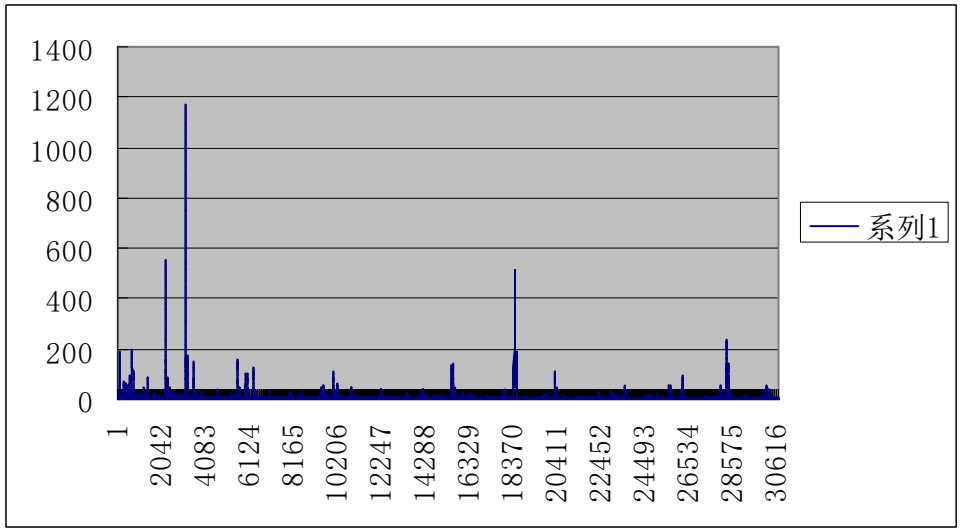
结果:

HPEYAVSVLLR

HPEYAVWLLR

HPEYAVPVFPK

结果3:



答案:

HLVDEPQNLLK

结果:

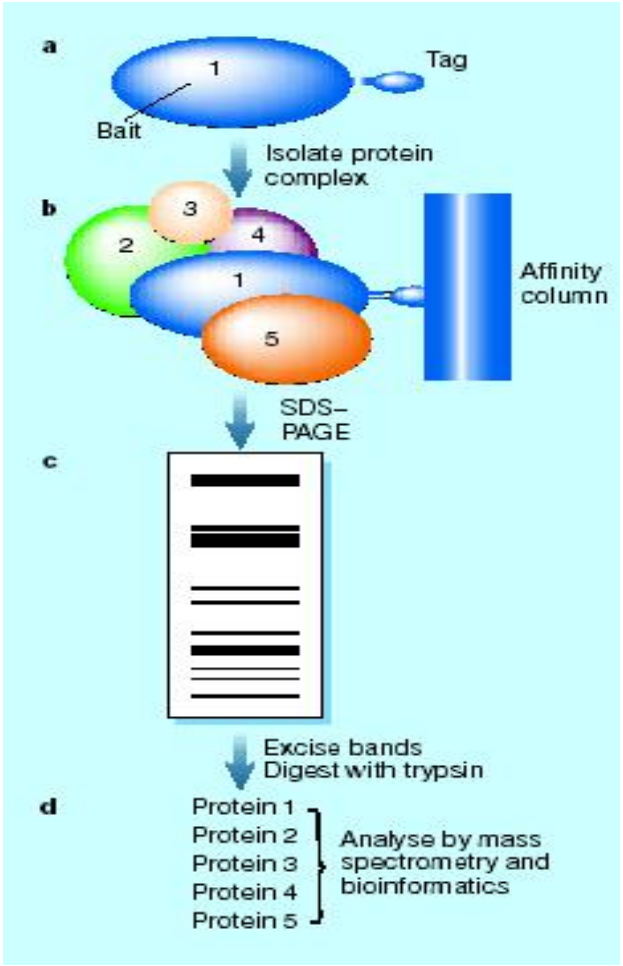
HLVDEAGPNLLK

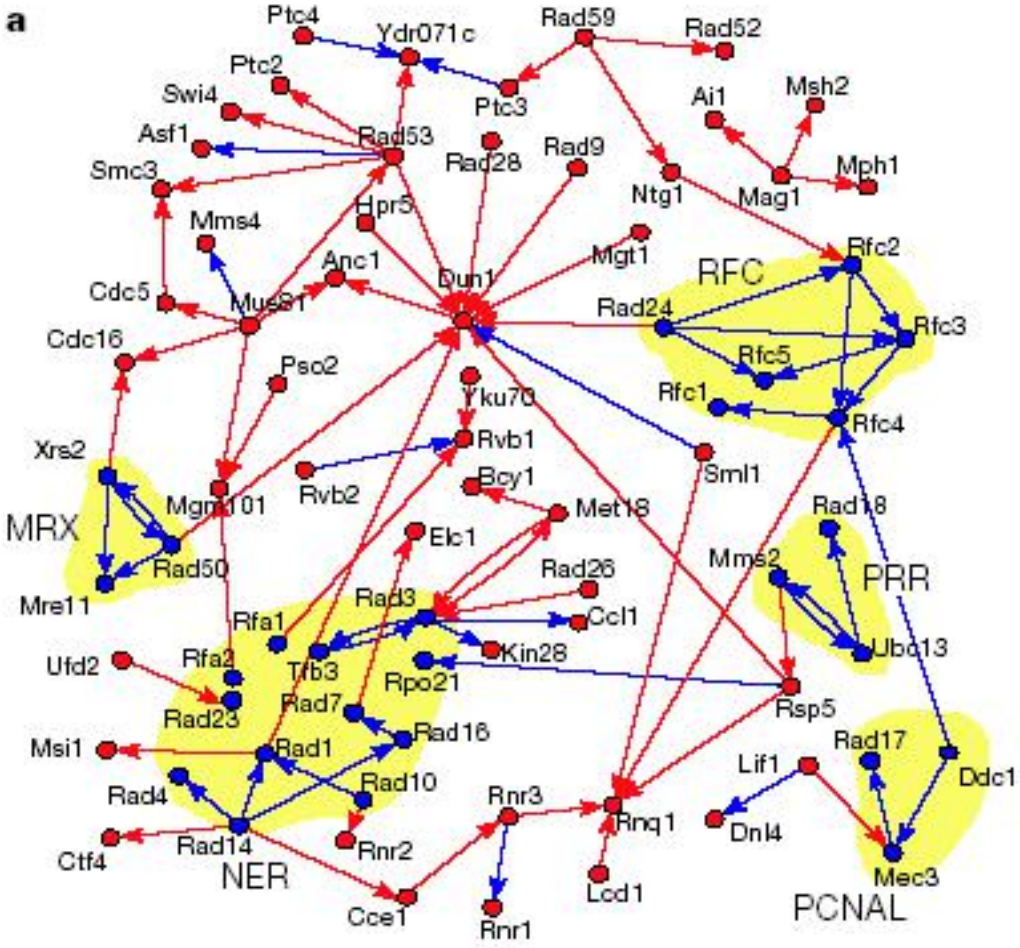
HLVDEQPNLLK

HLVDEPQNLLK

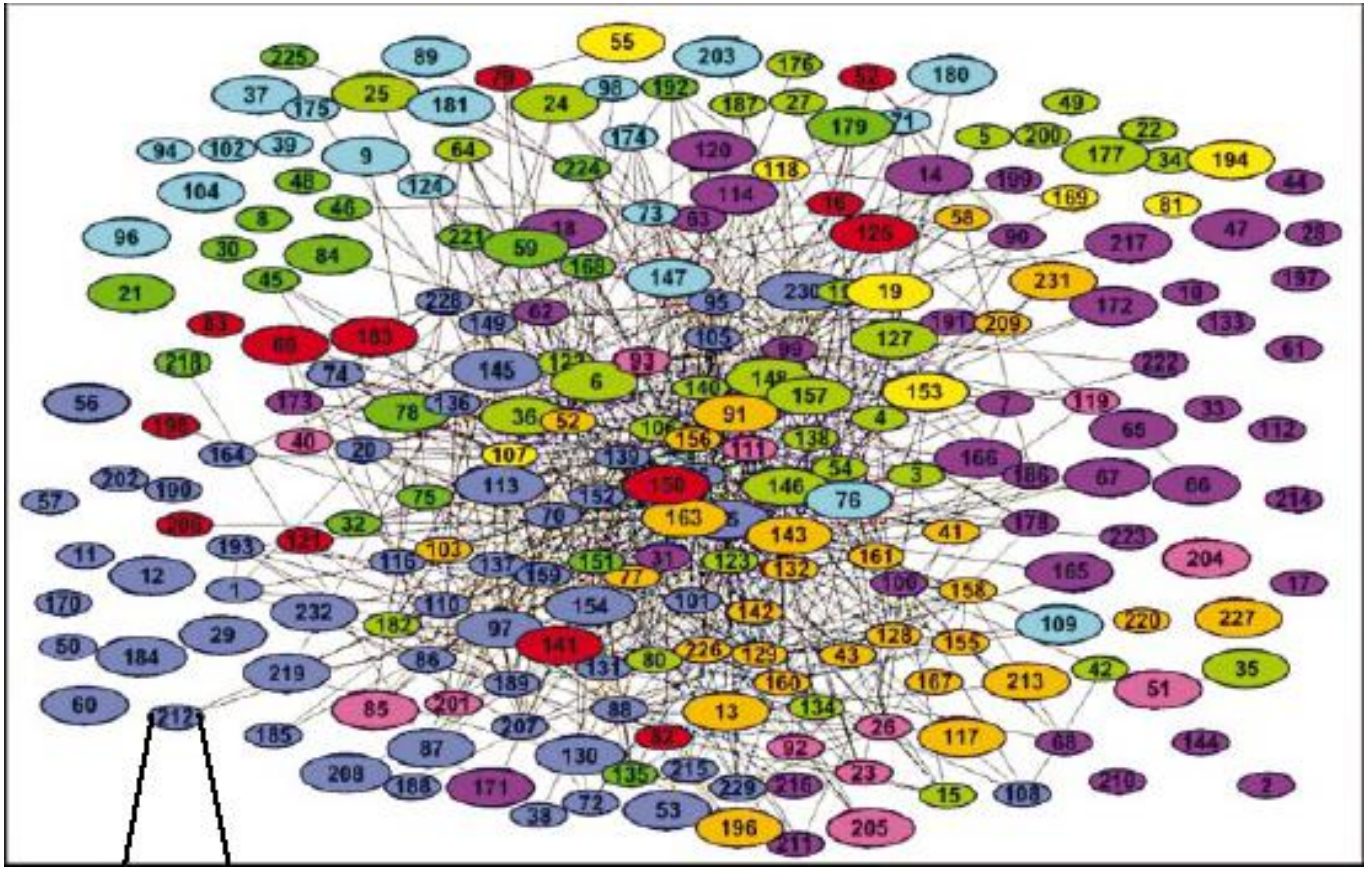
蛋白质相互作用网络分析

复杂的蛋白质相互作用网络





酵母: 2617蛋白, 11855连接



任务1：根据拓扑关系聚类

- 目标：将蛋白质分类，使得
  - 类内蛋白质连接紧密
  - 类间蛋白质连接稀疏

- 思路：
  - 第一步：先转化到欧式空间
  - 第二步：使用Ward法聚类

转换方法：

- 寻求方向Y，满足：
  - 每个结点都尽量得靠近其邻居
  - 即：min

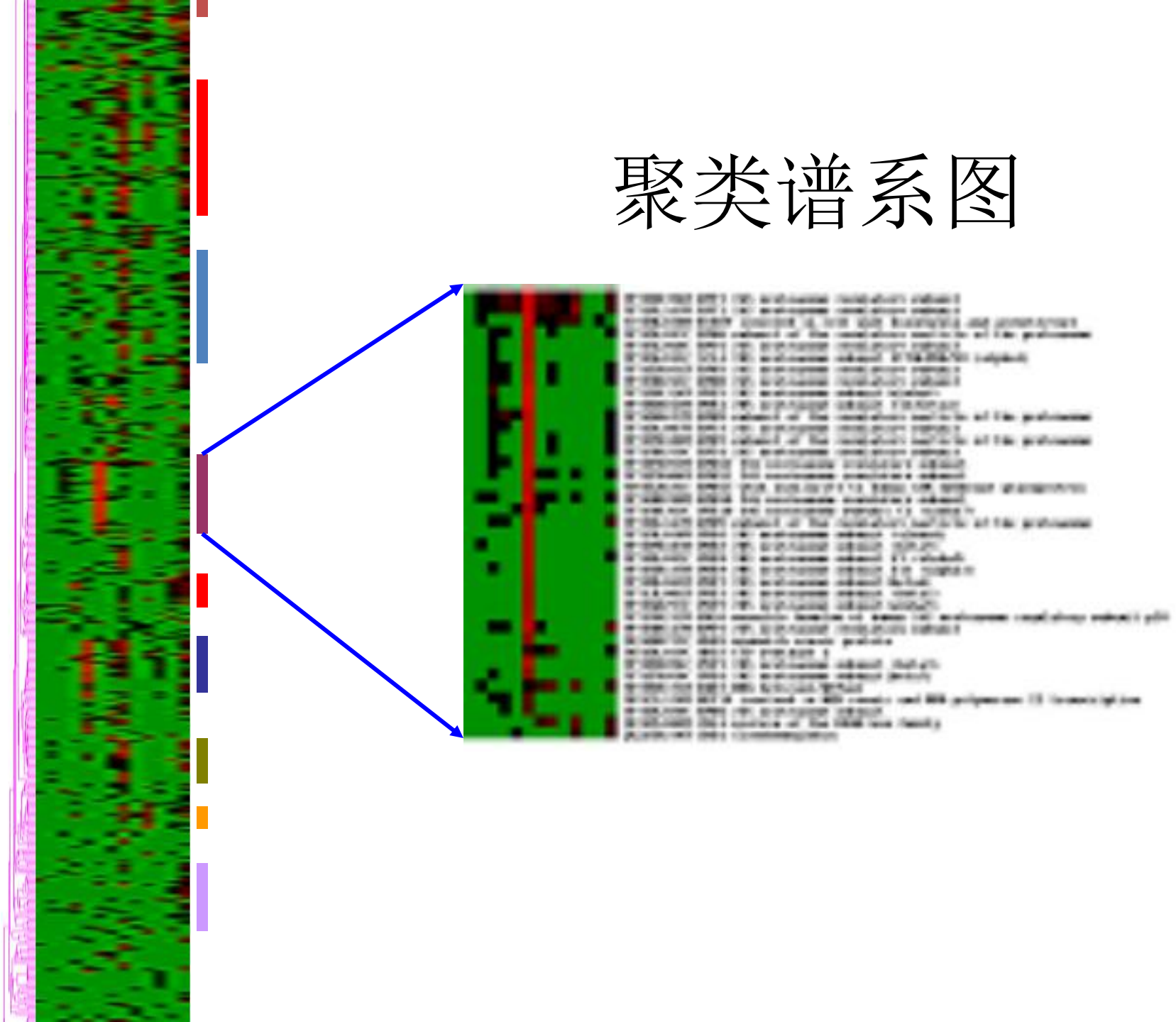
$$\sum_i \sum_j (y_i - y_j)^2 A_{ij}$$

- 定理：Y是对称阵A^T (L-A)A的最小特征值对应的特征向量
  - 其中L是Laplacian矩阵

聚类结果分析：

- 1。类内连接紧密，类间稀疏
- 2。同一类蛋白质功能类似

聚类谱系图



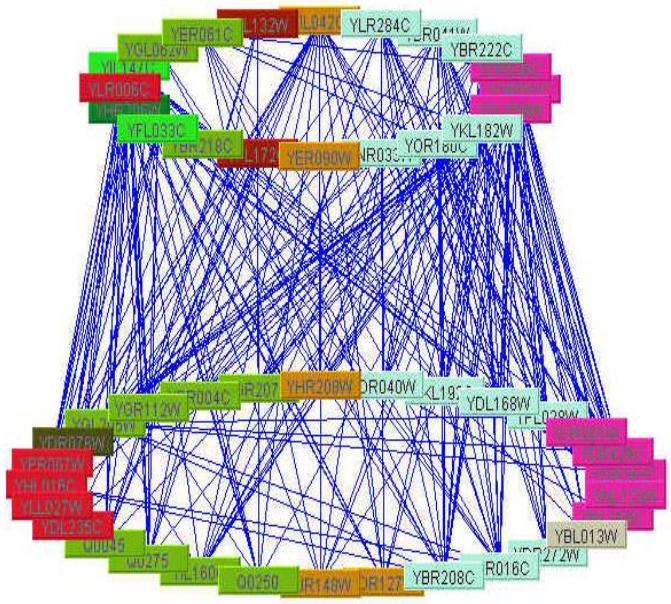
任务2：蛋白质相互作用网络的谱分析

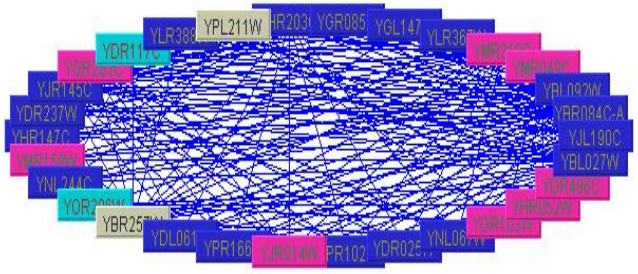
	1	2	3	.	.	.	**2617**
1	0	1	0	0	1	0	0
2	1	0	0	1	0	0	1
3	0	0	0	0	0	1	0
.	0	1	0	0	0	0	0
.	1	0	0	0	0	0	0
.	0	0	1	0	0	0	0
2617	0	1	0	0	0	0	0

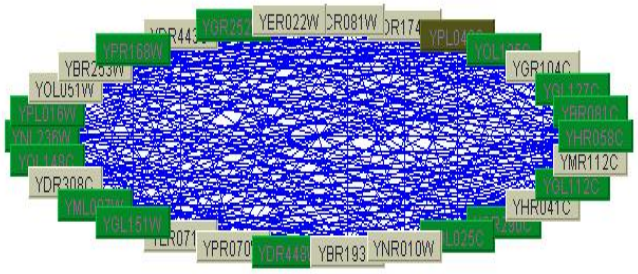
谱分析

- 1。正特征值相应的特征向量，绝对值较大分量相应蛋白质近似成团；
- 2。正特征值相应的特征向量，绝对值较大分量相应蛋白质近似成二部图；

团和二部图


YLR284C
YER061C
YBR222C
YFL033C
YKL182W
YHR208W
YDL168W
YBL013W
YBR208C


YPL211W
YGR085
YGL147
YDR117
YJR145C
YDR237W
YJL190C
YBL027W
YDL061
YPR166
YDR025


YER022W
YBR168W
YGR104C
YOL051W
YDR308C
YMR112C
YHR041C
YPR070
YDR448
YBR193
YNR010W

团：

- 分析：同一团的蛋白质生物学功能相似

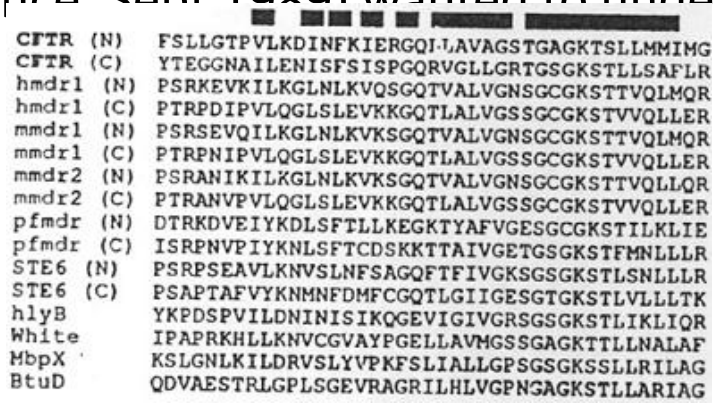
- 应用：预测未知蛋白质的功能
- 结果：
  - 分析了48个团，
  - 预测了100个未知蛋白质的功能
  - 部分结果Nature 5月份文章实验验证

二部图：

- 分析：
  - 同一复合物的不同亚基
  - 重要蛋白质的备份
  - 一个完整生化流程，不同阶段的反应物
- 应用：
  - 预测Pathway

Earliest Researches in Sequence Comparison

- Doolittle et al. (Science, July 1983) searched for platelet-derived growth factor (PDGF) in his own DB. He found that PDGF is similar to v-sis onc gene.
  - PDGF-2 1 SLGSLTIAEPAMIAECKTREEVFCICRRL?DR?? 34 p28sis 61
    LARGKRSLGSLSVAEPAMIAECKTRTEVFEISRRLIDRTN 100
- Riordan et al. (Science, Sept 1989) wanted to understand the cystic fibrosis gene:



Why we need to compare sequences?

- Biology has the following conjecture
  - Given two DNAs (or RNAs, or Proteins), high similarity
    →
    similar function or similar 3D structure
- Thus, in bioinformatics, we always compare the similarity of two biological sequences.

String edit problem

- Instead of minimizing the number of edge operations, we can associate a cost function to the operations and minimize the total cost. Such cost is called edit distance.
- For the previous example, the cost function is as follows:
  - A= _i nterestingly
    B= bioinformatics__ 10110111011
  - Edit distance = 11

	_	A	C	G	T
_	1	1	1	1	1
A	1	0	1	1	1
C	1	1	0	1	1
G	1	1	1	0	1
T	1	1	1	1	0

String alignment problem

- Instead of using string edit, in computational biology, people like to use string alignment.
- We use similarity function, instead of cost function, to evaluate the goodness of the alignment.
- E.g. of similarity function: match – 2, mismatch, insert, delete – -1.

$\delta(C,G) = -1$

–		A	C	G	T
–		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

String alignment

- Consider two strings ACAATCC and AGCATGC.
- One of their alignment is

- In the above alignment,
  - space ('_') is introduced to both strings
  - There are 5 matches, 1 mismatch, 1 insert, and 1 delete.

String alignment problem

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

- The alignment has similarity score 7

```
A_CAATCC   AGCA_TGC
```

- Note that the above alignment has the maximum score. Such alignment is called optimal alignment.
- String alignment problem tries to find the alignment with the maximum similarity score!
- String alignment problem is also called global alignment problem

Similarity vs. Distance

- String alignment problem and string edit distance are dual problems
  - This is your assignment!

- In this lecture, we only study string alignment!

Alignment:

Simple-Minded Probability & Score

Let p, q, r be respectively the probability of a match, a mismatch, and an indel. Then the probability of an alignment $A = (X, Y)$ is

$$prob(A) = p^m \cdot q^n \cdot r^h$$

where

$$\begin{aligned} m &= |\{i \mid x'_i = y'_i \neq -\}| \\ n &= |\{i \mid x'_i \neq y'_i, x'_i \neq -, y'_i \neq -\}| \\ q &= h\,|\{i \mid x'_i = -, y'_i \neq -\} \cup \{i \mid x'_i \neq -, y'_i = -\}| \end{aligned}$$

- The probability of a random alignment is 0.33^{m+n+h}.
- Define score S(A) by simple log odds ratio as $S(A) = \log \frac{p^m p^n p^h}{0.33^{m+n+h}}$.

How to align two sequences?

- In computer science, there is a technique called dynamic programming.
- We can apply dynamic programming to align two sequences.

Dynamic programming (DP)

- Dynamic programming aims to find optimal solution for some problem.

- It involves three steps:
  1. Characterize the structure of the problem
  2. Find the recursive formula
  3. Solve the recursive formula by bottom-up dynamic programming

- Below, we illustrate how to use DP to align two sequences.

Global alignment problem

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

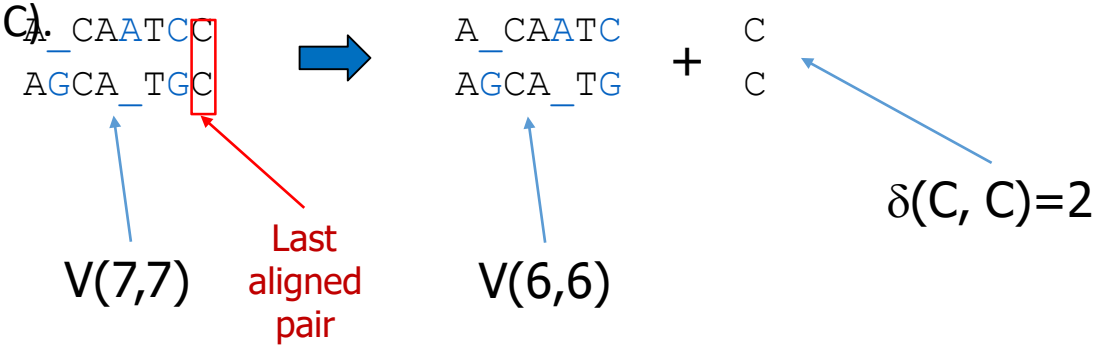
- Input:
  - A similarity score matrix $\delta[x,y]$
  - Two sequences S[1..m] and T[1..n]
- Output: An alignment of S and T that maximizes the alignment score Score(S, T).

- Example: S = ACAATCC and T=AGCATGC.
- The optimal alignment score is 7. The alignment is:

```
A_CAATCC
AGCA_TGC
```

Characterize the problem

- We aim to break the original problem Score(S[1..n], T[1..m]) into a smaller subproblem.
- Define **V(i, j)** be Score(S[1..i], T[1..j]), which is the score of the optimal alignment between S[1..i] and T[1..j].
- E.g. Score(ACAATCC, AGCATGC) can be transformed to Score(ACAATC, AGCATG) + δ(C,C).



Find recursive formula

- Identify the base case and recursive case of V(i, j)

- Basis (i=0 or j=0):
  - V(0, 0) = 0
  - V(0, j) = V(0, j-1) + δ(_, T[j])
    - Insert j times
  - V(i, 0) = V(i-1, 0) + δ(S[i], _)
    - Delete i times

Find recursive formula

- Recurrence: For i>0, j>0

$$V(i,j) = \max \begin{cases} V(i-1,j-1) + \delta(S[i], T[j]) \\ V(i-1,j) + \delta(S[i], \_) \\ V(i,j-1) + \delta(\_, T[j]) \end{cases}$$

Match/mismatch

Delete Insert

- In the alignment, the last pair must be either match/mismatch, delete, insert.

```
xxx…xx        xxx…xx    xxx…x_
      |             |         |
xxx…yy        yyy…y_    yyy…yy
```

match/mismatch delete insert

Solve the recursive formula by bottom-up dynamic programming

- We have the Needleman-Wunsch algorithm.

- V(0,0)=0;
- For i = 1 to n
  - V(i,0)=V(i-1,0)+δ(S[i],_);
- For j = 1 to m
  - V(0,j) = V(0,j-1)+δ(_,T[j]);
- For i = 1 to n
  - For j = 1 to m
    - V(i,j) = max{ V(i-1,j-1)+δ(S[i],T[j]), V(i-1,j)+δ(S[i],_), V(i,j-1)+δ(_,T[j]) };
- Return V(n,m);

Base case

Recursive case

Example (I)

Find alignment of
ACAATCC and AGCATGC

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

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

Example (II)

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

Example (III)

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

Recover the alignment

- To recover the alignment, we call back_tracing(V, n, m)

- back_tracing(V, i, j)
  - if (i=0 and j=0)
    - return []
  - else if (i=0)
    - return [back_tracing(V, i, j-1), (_, T[j])]
  - else if (j=0)
    - return [back_tracing(V, i-1, j), (S[i], _)]
  - else if (V(i,j) = V(i-1,j-1)+δ(S[i],T[j]))
    - return [back_tracing(V, i-1, j-1), (S[i], T[j])]
  - else if (V(i,j) = V(i-1,j)+δ(S[i],_))
    - return [back_tracing(V, i-1, j), (S[i], _)]
  - else
    - return [back_tracing(V, i, j-1), (_, T[j])]

Example (IV)

A_CAATCC
AGCA_TGC

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

Analysis

- We need to fill in all entries in the table with $n \times m$ matrix.
- Each entries can be computed in O(1) time.
- Time complexity = O(nm)
- Space complexity = O(nm)

Problem on Speed (I)

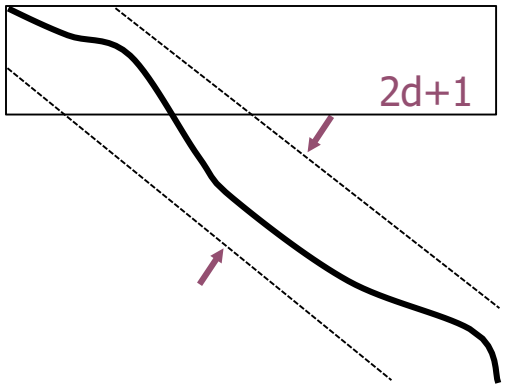
- Aho, Hirschberg, Ullman 1976
  - If we can only compare whether two symbols are equal or not, the string alignment problem can be solved in $\Omega(nm)$ time.
- Hirschberg 1978
  - If symbols are ordered and can be compared, the string alignment problem can be solved in $\Omega(n \log n)$ time.
- Masek and Paterson 1980
  - Based on Four-Russian's paradigm, the string alignment problem can be solved in $O(nm/\log^2 n)$ time.

Problem on Speed (II)

- Let d be the total number of inserts and deletes.
  - $0 \leq d \leq n+m$
- If d is smaller than n+m, can we get a better algorithm? Yes!

O(dn)-time algorithm

- Observe that the alignment should be inside the 2d+1 band.
- Thus, we don't need to fill-in the lower and upper triangle.
- Time complexity: O(dn).



Exam ple

- d=3

A_CAATC

C

AGCA_TG

C

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

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

Problem on Space

- Note that the dynamic programming requires a lot of space O(mn).
- When we compare two very long sequences, space may be the limiting factor.
- Can we solve the string alignment problem in linear space?

Suppose we don't need to recover the alignment

- In the pervious example, observe that the table can be filled in row by row.
- Thus, if we did not need to backtrack, space complexity = O(min(n, m))

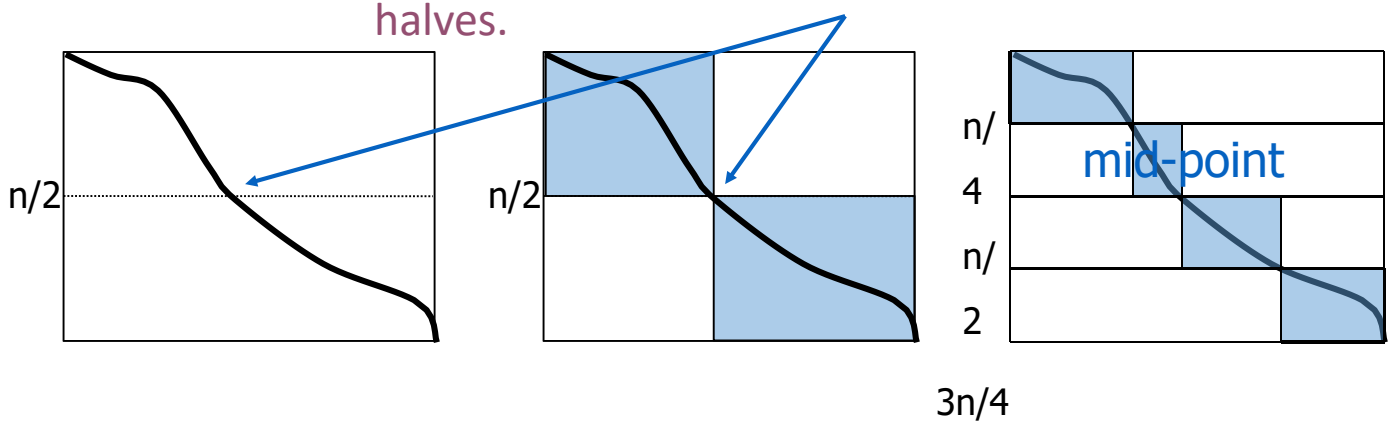
Example

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

- Note: when we fill in row 4, it only depends on row 3! So, we don't need to keep rows 1 and 2!
- Thus, we only need to keep two rows.
- Actually, keep one row is enough!

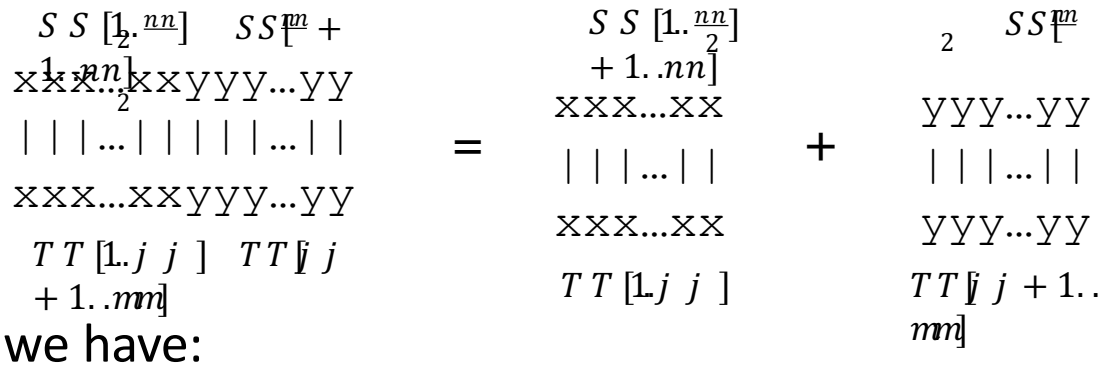
Can we recover the alignment given O(n+m) space?

- Yes. Idea: By recursion!
  1. Based on the cost-only algorithm, find the mid-point of the alignment!
  2. Divide the problem into two halves.
  3. Recursively deduce the alignments for the two halves.



Mid-point

- Mid-point is the position j that partitions an alignment in two halves.



- Hence, we have:

$$\max_{0 \le j \le m} \left\{ \begin{array}{l} V(S[1..\frac{n}{2}], T[1..j]) + V(S[\frac{n}{2}+1..n], T[j+1..m]) \\ = \max_{0 \le j \le m} \left\{ V_1(S[1..\frac{n}{2}], T[1..j]) + V_2(S[\frac{n}{2}+1..n], T[j+1..m]) \right\} \end{array} \right.$$

How to find the mid-point

Note:

$$V(S[1..n], T[1..m]) = \max_{0 \le j \le m} \{V(S[1..\tfrac{n}{2}], T[1..j]) + V(S[\tfrac{n}{2}+1..n], T[j+1..m])\}$$

1. Do cost-only dynamic programming for the first half.
   - Then, we find V(S[1..n/2], T[1..j]) for all j
2. Do cost-only dynamic programming for the reverse of the second half.
   - Then, we find V(S[n/2+1..n], T[j+1..m]) for all j
3. Determine j which maximizes the above sum!

Example (Step 1)

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

Example (Step 2)

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

Example (Step 3)

	_	A	G	C	A	T	G	C	_
_									
A									
C									
A									
A	-4	-1	-1	1	4	4	3	2	
T		-1	0	1	2	3	0	0	-3
C									
C									
_									

Example (Recursively solve the two subproblems)

	_	A	G	C	A	T	G	C	_
_									
A									
C									
A									
A									
T									
C									
C									
_									

Time Analysis

- Time for finding mid-point:
  - Step 1 takes O(n/2 m) time
  - Step 2 takes O(n/2 m) time
  - Step 3 takes O(m) time.
  - In total, O(nm) time.
- Let T(n, m) be the time needed to recover the alignment.
- T(n, m)
  = time for finding mid-point + time for solving the two subproblems
  = O(nm) + T(n/2, j) + T(n/2, m-j)
- Thus, time complexity = T(n, m) = O(nm)

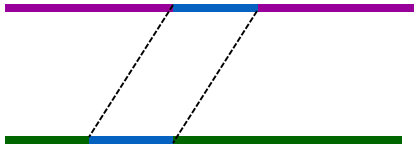
Space analysis

- Working memory for finding mid-point takes O(m) space
- Once we find the mid-point, we can free the working memory
- Thus, in each recursive call, we only need to store the alignment path
- Observe that the alignment subpaths are disjoint, the total space required is O(n+m).

More for string alignment problem

- Two special cases:
  - Longest common subsequence (LCS)
    - Score for mismatch is negative infinity
    - Score for insert/delete=0, Score for match=1
  - Hamming distance
    - Score for insert/delete is negative infinity
    - Score for match=1, Score for mismatch=0

Local alignment



- Given two long DNAs, both of them contain the same gene or closely related gene.
  - Can we identify the gene?
  - Local alignment problem:

Given two strings S[1..n] and T[1..m], among all substrings of S and T,
find substrings A of S and B of T whose global alignment has the highest score

Brute-force solution

- Algorithm:

  For every substring A=S[i'..i] of S,

  For every substring B=T[j'..j] of T,

  Compute the global alignment of A and B

  Return the pair (A, B) with the highest score

- Time:
  - There are $n^2/2$ choices of A and $m^2/2$ choices of B.
  - The global alignment of A and B can be computed in O(nm) time.
  - In total, time complexity = $O(n^3m^3)$
- Can we do better?

Some background

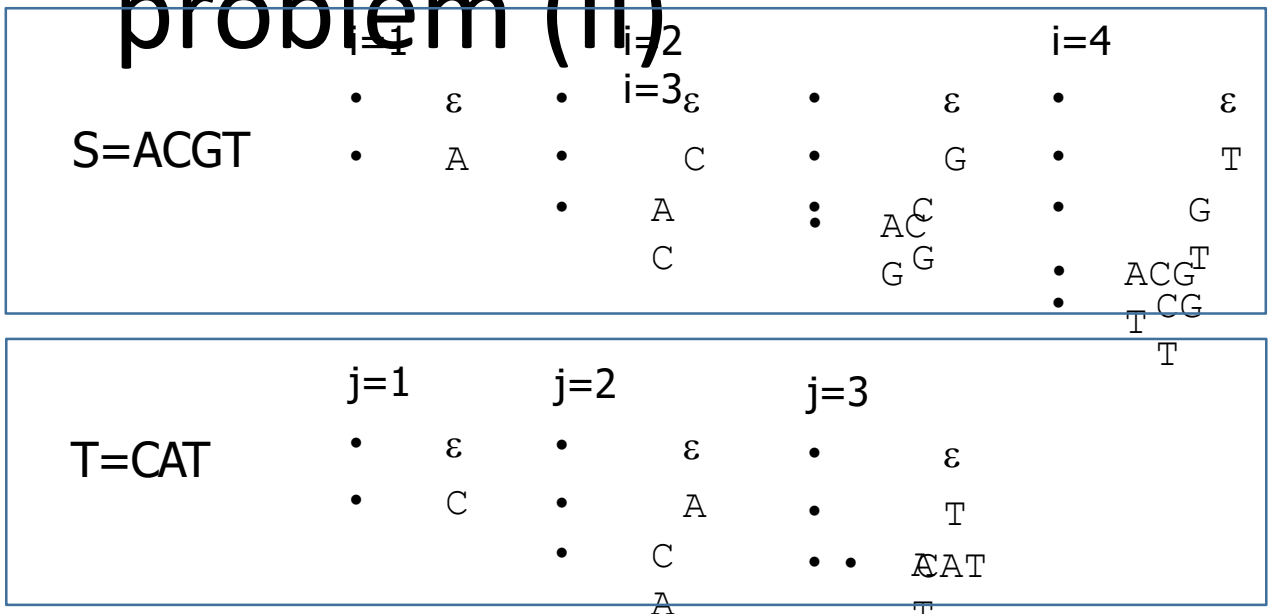
- X is a suffix of S[1..n] if X=S[k..n] for some $k \geq 1$
- X is a prefix of S[1..n] if X=S[1..k] for some $k \leq n$
- E.g.
  - Consider S[1..7] = ACCGATT
  - ACC is a prefix of S, GATT is a suffix of S
  - Empty string ε is both prefix and suffix of S

Local alignment problem (I)

	_	A	C	G	T
_		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

- Define V(i, j) be the maximum score of the global alignment between
  - any suffix of S[1..i] (i.e. S[i'..i] for i'=1,…,i) and
  - any suffix of T[1..j] (i.e. T[j'..j] for j'=1,…,j)
- Example: S=ACGT, T=CAT
  - Example 1 (i=3, j=2):
    - All suffixes of S[1..3] = {ε, G, CG, ACG}
    - All suffixes of T[1..2] = {ε, A, CA}
    - V(3,2)= max{ score(X,Y) | X∈{ε, G, CG, ACG}, Y∈{ε, A, CA} } = score(CG, CA)=1.
  - Example 2 (i=1, j=1):
    - All suffixes of S[1..1] = {ε, A}
    - All suffixes of T[1..1] = {ε, C}
    - V(3,2)= max{ score(X,Y) | X∈{ε, A}, Y∈{ε, C} } = score(ε, ε)=0.

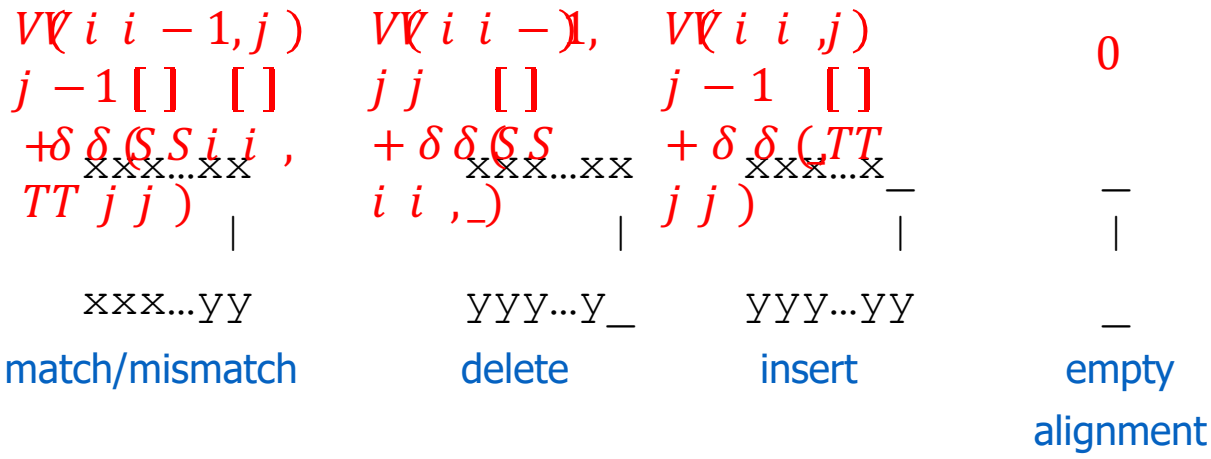
Local alignment problem (II)



- Note:
  - {all suffixes of S[1..i] | i=1,2,…,n} = all substrings of S
  - {all suffixes of T[1..j] | j=1,2,…,m} = all substrings of T
- Hence, score of local alignment is $\max_{i,j} V(i,j)$

Local alignment problem (III)

- V(i, j) is the score among the best pair of suffixes of (S[1..i] and T[1..j]).
- There are four cases: The best alignment ends at (1) match/mismatch, (2) insertion, (3) deletion or (4) no alignment



Smith-Waterman algorithm

- Basis:
  - V(i, 0) = V(0, j) = 0
- Recursion for i>0 and j>0:

$$V(i,j) = \max \begin{cases} 0 & \text{Align empty strings} \\ V(i-1,j-1) + \delta(S[i], T[j]) & \text{Match/mismatch} \\ V(i-1,j) + \delta(S[i], -) & \text{Delete} \\ V(i,j-1) + \delta(-, T[j]) & \text{Insert} \end{cases}$$

Example (I)

- Score for match = 2
- Score for insert, delete, mismatch = -1

	_	C	T	C	A	T	G	C
_	0	0	0	0	0	0	0	0
A	0							
C	0							
A	0							
A	0							
T	0							
C	0							
G	0							

Example (II)

- Score for match = 2
- Score for insert, delete, mismatch = -1

	_	C	T	C	A	T	G	C
_	0	0	0	0	0	0	0	0
A	0	0	0	0	2	1	0	0
C	0	2	1	2	1	1	0	2
A	0	0	1	1	4	3	2	1
A	0	0	0	0	3	3	2	1
T	0	0	2	1	2			
C								
G								

Example (III)

CAATCG

C_AT_G

	_	C	T	C	A	T	G	C
_	0	0	0	0	0	0	0	0
A	0	0	0	0	2	1	0	0
C	0	2	1	2	1	1	0	2
A	0	1	1	1	4	3	2	1
A	0	0	0	0	3	3	2	1
T	0	0	2	1	2	5	4	3
C	0	2	1	4	3	4	4	6
G	0	1	1	3	3	3	6	5

Analysis

- We need to fill in all entries in the table with $n \times m$ matrix.
- Each entries can be computed in O(1) time.
- Finally, finding the entry with the maximum value.
- Time complexity = O(nm)
- Space complexity = O(nm)

Semi-global alignment

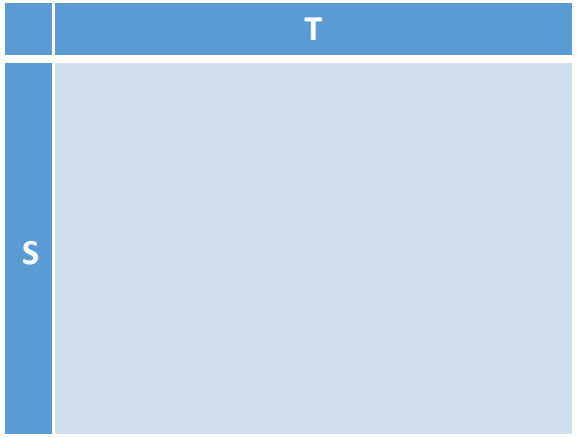
- We score alignments ignoring some of the end spaces
- Example 1: ignoring beginning and ending spaces of the second sequence.
  - ATCCGAA_CATCCAATCGAAGC

    _____AGCATGCAAT_____
  - The score of below alignment is 14
    - 8 matches (score=16), 1 delete (score=-1), 1 mismatch (score=-1)
  - This alignment can be used to locate gene in a prokaryotic genome

Semi-global alignment

- Example 2: ignoring beginning spaces of the 1st sequence and ending spaces of the 2nd sequence
  - _____ACCTCACGATCCGA TCAACGATCACCGCA_____
  - The score of above alignment is 9
    - 5 matches (score=10), 1 mismatch (score=-1)
  - This alignment can be used to find the common region of two overlapping sequences

How to compute semi-global alignment?

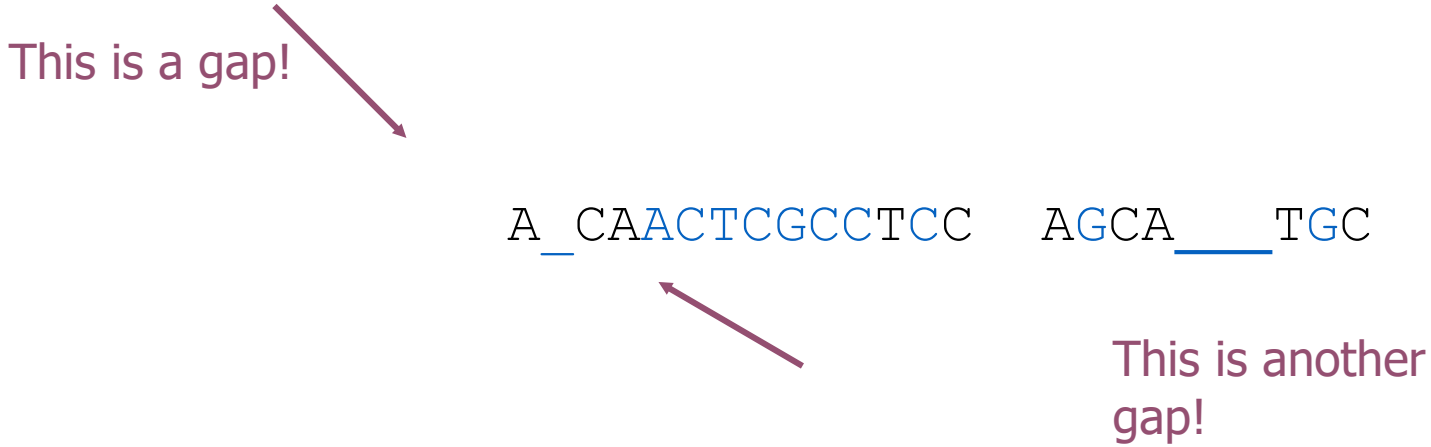


- In general, we can forgive spaces
  - in the beginning or ending of S[1..n]
  - in the beginning or ending of T[1..m]
- Semi-global alignment can be computed using the dynamic programming for global alignment with some small changes.
- Below table summaries the changes

Spaces that are not charged	Action
Spaces in the beginning of S[1..n]	Initialize first row with zeros
Spaces in the ending of S[1..n]	Look for maximum in the last row
Spaces in the beginning of T[1..m]	Initialize first column with zeros
Spaces in the ending of T[1..m]	Look for maximum in the last column

Gaps

- A gap in an alignment is a maximal substring of contiguous spaces in either sequence of the alignment



Penalty for gaps

- Previous discussion assumes the penalty for insert/delete is proportional to the length of a gap!
- This assumption may not be valid in some applications, for examples:
  - Mutation may cause insertion/deletion of a large substring. Such kind of mutation may be as likely as insertion/deletion of a single base.
  - Recall that mRNA misses the introns. When aligning mRNA with its gene, the penalty should not be proportional to the length of the gaps.

General gap penalty (I)

- Definition: g(q) is denoted as the penalty of a gap of length q
- Global alignment of S[1..n] and T[1..m]:
  - Denote V(i, j) be the score for global alignment between S[1..i] and T[1..j].
  - Base cases:
    - V(0, 0) = 0
    - V(0, j) = -g(j)
    - V(i, 0) = -g(i)

General gap penalty (II)

- Recurrence: for i>0 and j>0,

$$V(i,j) = \max \begin{cases} V(i-1,j-1) + \delta(S[i], T[j]) \\ \max_{0 \le k \le j-1} \{V(i,k) + g(j-k)\} \\ \max_{0 \le k \le i-1} \{V(k,j) - g(i-k)\} \end{cases}$$

Match/Insert T[k+1..j]

Delete S[k+1..i]

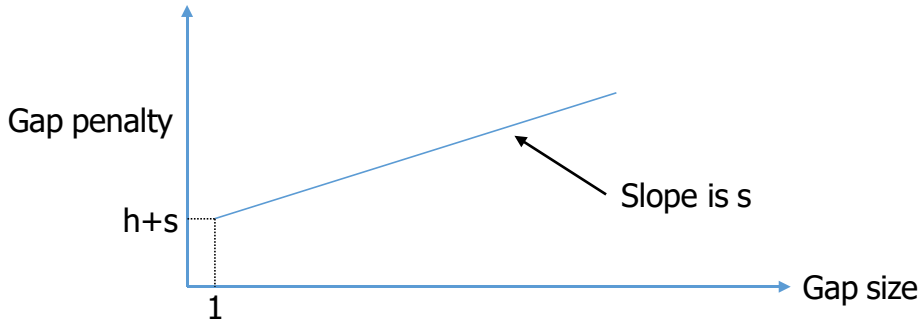
Analysis

- We need to fill in all entries in the $n \times m$ table.
- Each entry can be computed in $O(n+m)$ time.
- Time complexity = $O(n^2m + nm^2)$
- Space complexity = $O(nm)$

Affine gap model

- In this model, the penalty for a gap is divided into two parts:
  - A penalty (h) for initiating the gap
  - A penalty (s) depending on the length of the gap
- Consider a gap with q spaces,
  - The penalty g(q) = h+qs

Usually, h>s

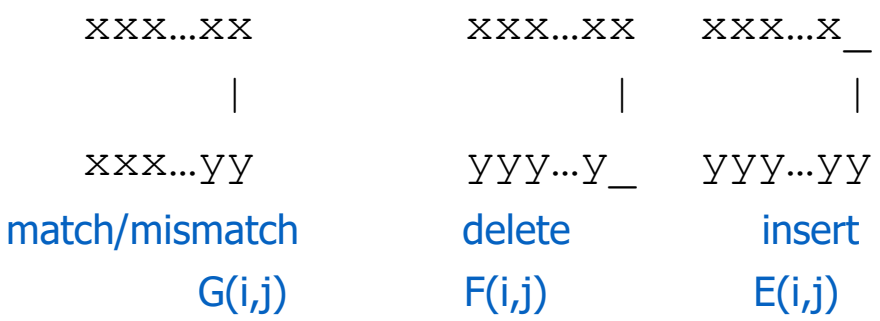


Dynamic programming solution (I)

- V(i, j) is the score of a global optimal alignment between S[1..i] and T[1..j]
- F(i, j) is the score of a global optimal alignment between S[1..i] and T[1..j] with S[i] matches with a space
- E(i, j) is the score of a global optimal alignment between S[1..i] and T[1..j] with a space matches with T[j]

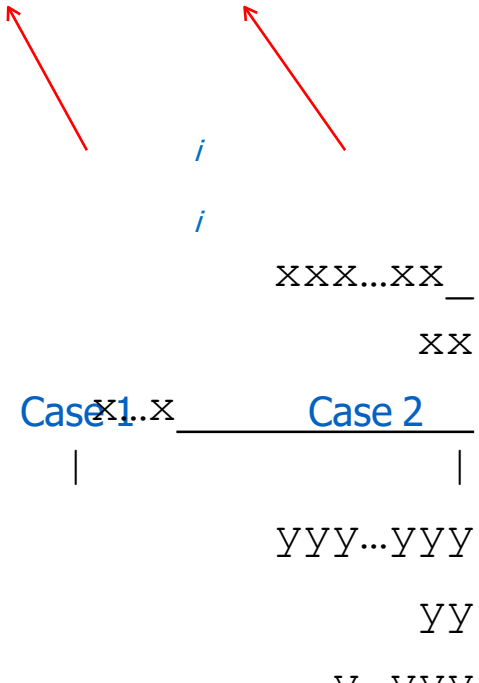
V(i, j)

- Basis:
  - V(0, 0) = 0
  - V(i, 0) = -h-is; V(0, j) = -h-js
- Recurrence:
  - V(i, j) = max { V(i-1, j-1) + δ(S[i], T[j]), F(i, j), E(i, j) }



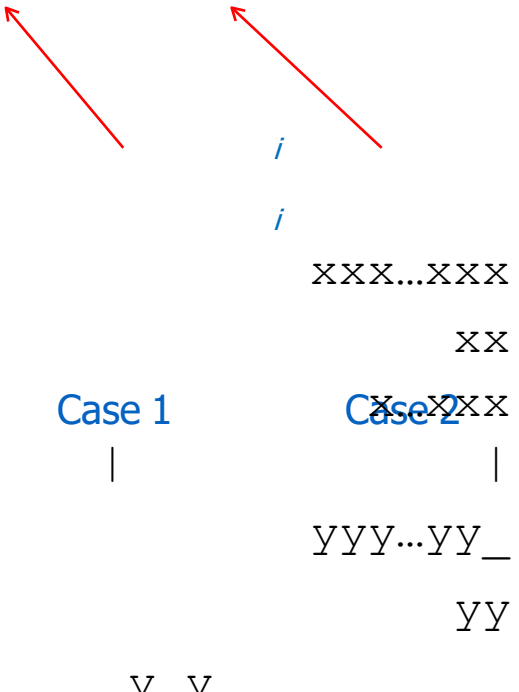
E(i, j)

- Basis:
  - E(i, 0) = -∞
- Recurrence:
  - E(i, j) = max { V(i, j-1)–h–s, E(i, j-1)–s}



F(i, j)

- Basis:
  - F(0, j) = -∞
- Recurrence:
  - F(i, j) = max { V(i-1, j)–h–s, F(i-1, j)–s }



Summary of the dynamic programming solution

- Basis:
  - V(0, 0) = 0
  - V(i, 0) = -h-is; V(0, j) = -h-js
  - E(i, 0) = $-\infty$
  - F(0, j) = $-\infty$
- Recurrence:
  - E(i, j) = max { E(i, j-1)–s, V(i, j-1)–h–s }
  - F(i, j) = max { F(i-1, j)–s, V(i-1, j)–h–s }
  - V(i, j) = max { V(i-1, j-1) + δ(S[i], T[j]), F(i, j), E(i, j) }

Exam ple

- S=ACCGA
- T=AGTTA
- Match=1, Mismatch=- 3, initial gap (h) =1, each space (s) = 1
- Fill-in base case.

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	0	-2	-3	-4	-5	-6
A	1	-2					
C	2	-3					
C	3	-4					
G	4	-5					
A	5	-6					

V table

V(0, 0) = 0

V(i, 0) = -h-is; V(0, j) = -h-js

E(i, 0) = -∞

F(0, j) = -∞

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0						
A	1	-∞					
C	2	-∞					
C	3	-∞					
G	4	-∞					
A	5	-∞					

E table

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0		-∞	-∞	-∞	-∞	-∞
A	1						
C	2						
C	3						
G	4						
A	5						

F table

Exam ple

$V(i, j) = \max \{ V(i-1, j-1) + \delta(S[i], T[j]), F(i, j), E(i, j) \}$

$F(i, j) = \max \{ F(i-1, j) - s, V(i-1, j) - h - s \}$

$E(i, j) = \max \{ E(i, j-1) - s, V(i, j-1) - h - s \}$

- S=ACCGA
- T=AGTTA
- Match=1, Mismatch=-3,
  initial gap (h) =-1,
  each space (s) = -1
- Fill-in row by row.
- For each entry,
  - fill-in E, F
  - Then, V

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0	0	-2	-3	-4	-5	-6
A	1	-2	1				
C	2	-3					
C	3	-4					
G	4	-5					
A	5	-6					

V table

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0						
A	1	-∞	-4				
C	2	-∞					
C	3	-∞					
G	4	-∞					
A	5	-∞					

E table

		_	A	G	T	T	A
		0	1	2	3	4	5
_	0		-∞	-∞	-∞	-∞	-∞
A	1		-4				
C	2						
C	3						
G	4						
A	5						

F table

Example

$$V(i, j) = \max \{ V(i-1, j-1) + \delta(S[i], T[j]), F(i, j), E(i, j) \}$$

$$F(i, j) = \max \{ F(i-1, j) - s, V(i-1, j) - h - s \}$$

$$E(i, j) = \max \{ E(i, j-1) - s, V(i, j-1) - h - s \}$$

- S=ACCGA
- T=AGTTA
- Match=1, Mismatch=-3,
  initial gap (h) =-1,
  each space (s) = -1
- Fill-in row by row.
- For each entry,
  - fill-in E, F
  - Then, V

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

V table

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

E table

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

F table

Analysis

- We need to fill in 4 tables, each is of size $n \times m$.
- Each entry can be computed in O(1) time.
- Time complexity = O(nm)
- Space complexity = O(nm)

Scoring function

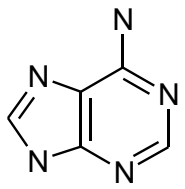
- In the rest of this lecture, we discuss the scoring function for both DNA and Protein

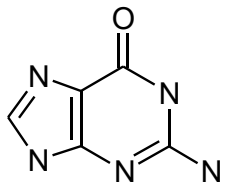
Scoring function for DNA

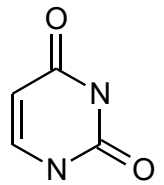
- For DNA, since we only have 4 nucleotides, the score function is a 4x4 matrix.

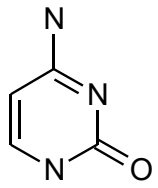
- Criteria for the scoring function for DNA:
  1. Positive for match
  2. Negative for mismatch
  3. The expected score is negative when an alignment is extended by random pairs.

Scoring function for DNA









Adenine

Guanine

Thymine

Cytosine

- Since DNA has 4 nucleotides only, the score function is simple.
  - BLAST matrix
  - Transition Transversion matrix:
    - Transition = replacing purine (A,G) by purine or replacing pyrimidine (C,T) by pyrimadine!
    - Transversion = replacing purine by pyrimadine, or vice versa
    - Give mild penalty to transition

	A	C	G	T
A	5	-4	-4	-4
C	-4	5	-4	-4
G	-4	-4	5	-4
T	-4	-4	-4	5

BLAST Matrix

	A	C	G	T
A	1	-5	-1	-5
C	-5	1	-5	-1
G	-1	-5	1	-5
T	-5	-1	-5	1

Transition Transversion Matrix

Scoring function for Protein

- Scoring function for amino acid is a 20x20 matrix.

- There are a few scoring matrix:
  - Identity matrix
  - Genetic code matrix
  - Physical property matrix: based on chemical/physical similarity
  - Statistical scoring matrices (PAM, BLOSUM): based on observed substitution frequencies

Identity matrix

- The simple amino-acid scoring matrix is the identity matrix
  - Match scores +1 while mismatch scores 0.

Genetic code matrix

- Score is defined by the number of shared nucleotides in their codons.

	A	B	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S
		TV	WY	Z	3.0	2.0	1.0	2.0	2.0	1.0	2.0	1.0	1.0	1.0	1.0	1.0	2.0
A	1.0	1.0	2.0	2.0	2.0	1.0	1.0	2.0	A								
	3.0	1.0	3.0	2.0	1.0	2.0	2.0	2.0	2.0	1.0	1.0	3.0	1.0	2.0	1.0	2.0	2.0
														2.0	0.0	2.0	2.0 B
	3.0	1.0	0.0	2.0	2.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	2.0	2.0	1.0	1.0
															2.0	2.0	0.0 C
	3.0	2.0	1.0	2.0	2.0	1.0	1.0	1.0	0.0	2.0	1.0	1.0	1.0	1.0	1.0	2.0	0.0
																2.0	2.0 D
	3.0	0.0	2.0	1.0	1.0	2.0	1.0	1.0	1.0	1.0	2.0	1.0	1.0	1.0	2.0	1.0	1.0
																	3.0 E
	3.0	1.0	1.0	2.0	0.0	2.0	1.0	1.0	1.0	0.0	1.0	2.0	1.0	2.0	1.0	2.0	0.0
																	F
	3.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	2.0	1.0	2.0	2.0	1.0	2.0	2.0 G
	3.0	1.0	3.0	1.0	1.0	2.0	0.0	2.0	2.0	2.0	2.0	1.0	1.0	1.0	0.0	2.0	2.0 H
	3.0	2.0	3.0	2.0	2.0	2.0	2.0	2.0	1.0	1.0	2.0	2.0	2.0	2.0	0.0	1.0	1.0 I
	3.0	2.0	3.0	2.0	2.0	1.0	2.0	2.0	1.0	2.0	2.0	1.0	2.0	1.0	1.0	1.0	2.0 K
	3.0	2.0	3.0	2.0	2.0	1.0	2.0	1.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	1.0	2.0 L
							3.0	1.0	1.0	1.0	2.0	1.0	2.0	2.0	1.0	0.0	1.0 M
								3.0	1.0	1.0	1.0	2.0	2.0	1.0	0.0	2.0	2.0 N
									3.0	2.0	2.0	2.0	2.0	1.0	1.0	1.0	2.0 P
										3.0	2.0	1.0	1.0	1.0	1.0	1.0	3.0 Q
											3.0	2.0	2.0	1.0	2.0	1.0	2.0 R
												3.0	2.0	1.0	2.0	2.0	1.0 S
													3.0	1.0	1.0	1.0	1.0 T
														3.0	1.0	1.0	2.0 V
															3.0	1.0	1.0 W
																3.0	1.0 Y
																	3.0 Z

	T	C	A	G	
T	TTT Phe [F]	TCT Ser [S]	TAT Tyr [Y]	TGT Cys [C]	T
	TTC Phe [F]	TCC Ser [S]	TAC Tyr [Y]	TGC Cys [C]	C
	TTA Leu [L]	TCA Ser [S]	TAA Ter [end]	TGA Ter [end]	A
	TTG Leu [L]	TCG Ser [S]	TAG Ter [end]	TGG Trp [W]	G
C	CTT Leu [L]	CCT Pro [P]	CAT His [H]	CGT Arg [R]	T
	CTC Leu [L]	CCC Pro [P]	CAC His [H]	CGC Arg [R]	C
	CTA Leu [L]	CCA Pro [P]	CAA Gln [Q]	CGA Arg [R]	A
	CTG Leu [L]	CCG Pro [P]	CAG Gln [Q]	CGG Arg [R]	G
A	ATT Il e [I]	ACT Thr [T]	AAT Asn [N]	AGT Ser [S]	T
	ATC Ile [I]	ACC Thr [T]	AAC Asn [N]	AGC Ser [S]	C
	ATA Ile [I]	ACA Thr [T]	AAA Lys [K]	AGA Arg [R]	A
	ATG Met [M]	ACG Thr [T]	AAG Lys [K]	AGG Arg [R]	G
G	GTT Val [V]	GCT Ala [A]	GAT Asp [D]	GGT Gly [G]	T
	GTC Val [V]	GCC Ala [A]	GAC Asp [D]	GGC Gly [G]	C
	GTA Val [V]	GCA Ala [A]	GAA Glu [E]	GGA Gly [G]	A
	GTG Val [V]	GCG Ala [A]	GAG Glu [E]	GGG Gly [G]	G

Scoring function for protein using physical/chemical properties

- Idea: an amino acid is more likely to be substituted by another if they have similar property
- See Karlin and Ghandour (1985, PNAS 82:8597)
- The score matrices can be derived based on hydrophobicity, charge, electronegativity, and size
- E.g. we give higher score for substituting nonpolar amino acid to another nonpolar amino acid

Scoring function for protein based on statistical model

- Most often used approaches
- Two popular matrices:
  - Point Accepted Mutation (PAM) matrix
  - BLOSUM
- Both methods define the score as the log-odds ratio between the observed substitution rate and the actual substitution rate

Point Accepted Mutation (PAM)

- PAM was developed by Dayhoff (1978).

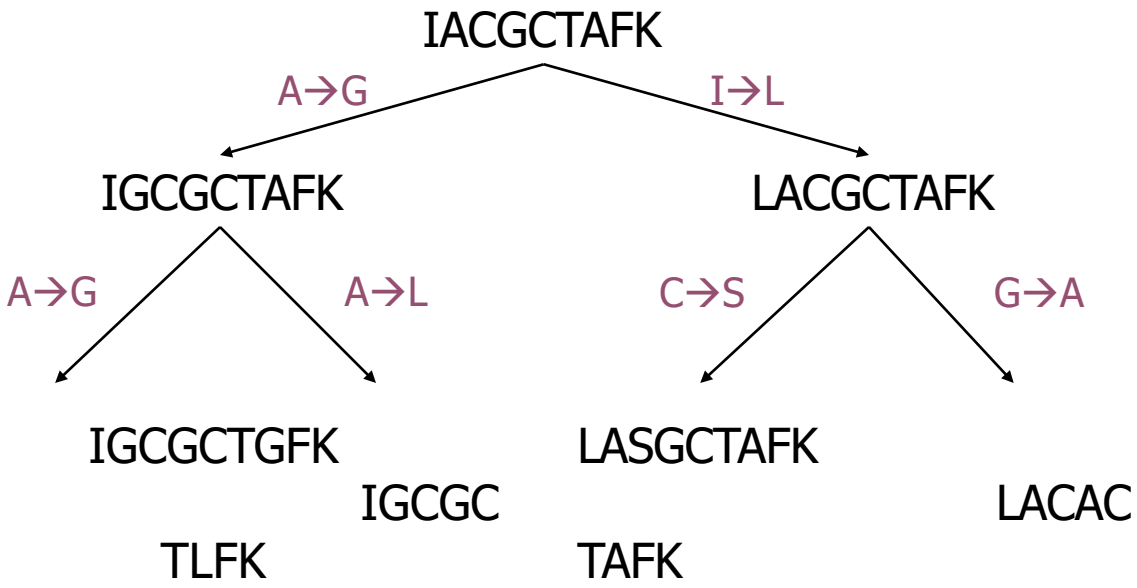
- A point mutation means substituting one residue by another.
- It is called an accepted point mutation if the mutation does not change the protein's function or is not fatal.
- Two sequence S_1 and S_2 are said to be 1 PAM diverged if a series of accepted point mutation can convert S_1 to S_2 with an average of 1 accepted point mutation per 100 residues

PAM matrix by example (I)

- Ungapped alignment is constructed for high similarity amino acid sequences (usually >85%)
- Below is a simplified global multiple alignment of some highly similar amino acid sequences (without gap):
  - `IACGCTAFK IGCGCTAFK LACGCTAFK IGCGCTGFK IGCGCTLFK LASGCTAFK LACACTAFK`

PAM matrix by example (II)

- Build the phylogenetic tree for the sequences



PAM-1 matrix

- $\delta(a,b) = \log_{10}\frac{O_{a,b}}{EE}$
  - where $O_{a,b}$ and $E_{a,b}$ are the observed frequency and the expected frequency.

- Since PAM-1 assume 1 mutation per 100 residues,
  - $O_{a,a} = 99/100$.
- For $a \neq b$,
  - $O_{a,b} = F_{a,b} / (100\ \Sigma_x \Sigma_y F_{x,y})$ where $F_{a,b}$ is the frequency $F_{a,b}$ of substituting a by b or b by a.
- $E_{a,b} = f_a * f_b$ where f_a is the no. of a divided by total residues

- E.g., $F_{A,G} = 3$, $F_{A,L} = 1$. $f_A = f_G = 10/63$.
- $O_{A,G} = 3/(100*2*6) = 0.0025$
  - $E_{A,G} = (10/63)(10/63) = 0.0252$
  - $\delta(A,G) = \log (0.0025 / 0.0252) = \log (0.09925) = -1.0034$

PAM-n matrix

- Let $M_{a,b}$ be the probability that a is mutated to b, which equals $O_{a,b}/f_a$.
- $M^n(a,b)$ is the probability that a is mutated to b after n mutation.
- PAM-n matrix is created by extrapolate PAM-1 matrix.
- PAM-n matrix is computed as follows.
  - At time t, suppose the residue is a!
  - At time t+1, probability that it becomes j is M(a,b)
  - At time t+2, probability that it becomes j is $M^2(a,b)$
  - …
  - At time t+n, probability that it becomes j is $M^n(a,b)$
- Therefore, (a,b) entry of the PAM-n matrix is

$$\log(f_a M^n(a,b)/f_a f_b) = \log(M^n(a,b)/f_b)$$

%difference versus PAM

- M^0 is the identity matrix.

- After n mutation steps,
  - %difference $= 1 - \sum_{i\ i \in AA} MM^{mn}(i\ \ i\ \ j\ \ i\ \ )f\ f_{ii}$
- The right table shows the relation.

- PAM250 corresponds to approximately 80% difference.

%Difference	PAM
1	1
5	5
10	11
15	17
20	23
25	30
30	38
35	47
40	56
45	67
50	80
55	94
60	112
65	133
70	159
75	195
80	246
85	328

History of PAM

- Dayhoff, M. O., Schwartz, R. M., and Orcutt, B. C. [1979] in Atlas of Protein Sequence and Structure
  - This is the matrix MDM78 PAM250
- Gribskov, M. & Burgess, R.R. (1986) *NAR* **14**:6745-6763
  - They renormalize MDM78 so that M(i,i)=1.5
  - The scores for non-identical residues have been adjusted to give a mean of -0.17 and a standard deviation of 0.364. The negative expectation value is necessary for local alignment routines
- Jones et al. (Jones, D.T., Taylor, W.R. & Thornton, J.M. (1992), *CABIOS* **8**:275-282) have automated the procedure of deriving scoring matrices from sequence databases.
  - Using 40 folds larger database, a better PAM is built.
- Gaston Gonnet and coworkers (Gonnet, G., Cohen, M. A.& Benner, S. (1992) *Science* **256**:1443-1445) have calculated an exhaustive all-against-all sequence matching of the entire protein database using their DARWIN system.
  - By using a large database, they can directly compile mutation matrix at different PAM distances.
  - They show that extrapolation introduces significant errors.

BLOSUM (BLOck SUbstition Matrix)

- PAM did not work well for aligning evolutionarily divergent sequences since the matrix is generated by extrapolation.
- Henikoff and Henikoff (1992) proposed BLOSUM.

- Unlike PAM, BLOSUM matrix is constructed directly from the observed alignment (instead of extrapolation)

Generating conserved blocks

- In BLOSUM, the input is the set of multiple alignments for nonredundant groups of protein families.
- Based on PROTOMAT, blocks of nongapped local aligments are derived.
- Each block represents a conserved region of a protein family.

Extract frequencies from blocks

- From all blocks, we count the frequency p_a for each amino acid residue a.
- For any two amino acid residues a and b, we count the frequency p_{ab} of the aligned pairs of a and b.
- For example,
  - IACGCTAFK IGCGCTAFK LACGCTAFK IGCGCTGFK IGCGCTLFK LASGCTAFK LACACTAFK
- There are 7*9=63 residues, including 10's A and 10's G. Hence, $p_A = p_G = 10/63$.
  - There are $9\binom{7}{2} = 189$ aligned residue pairs, including 23 (A, G) pairs. $p_{AG} = 23/198$.
  - Hence, p
    - 7

The scoring function of BLOSUM

- For each pair of aligned residues a and b, the alignment score $\delta(a,b) = 1/\lambda \ln p_{ab}/(p_a p_b)$
  - where p_{ab} is the probability that a and b are observed to align together. p_a and p_b are the frequency of residues a and b respectively. λ is a normalization constant.
- Example: $p_A=p_G=0.159$, $p_{AG} = 0.122$. With $\lambda=0.347$, $\delta(A,L)=4.538$.

What is BLOSUM 62?

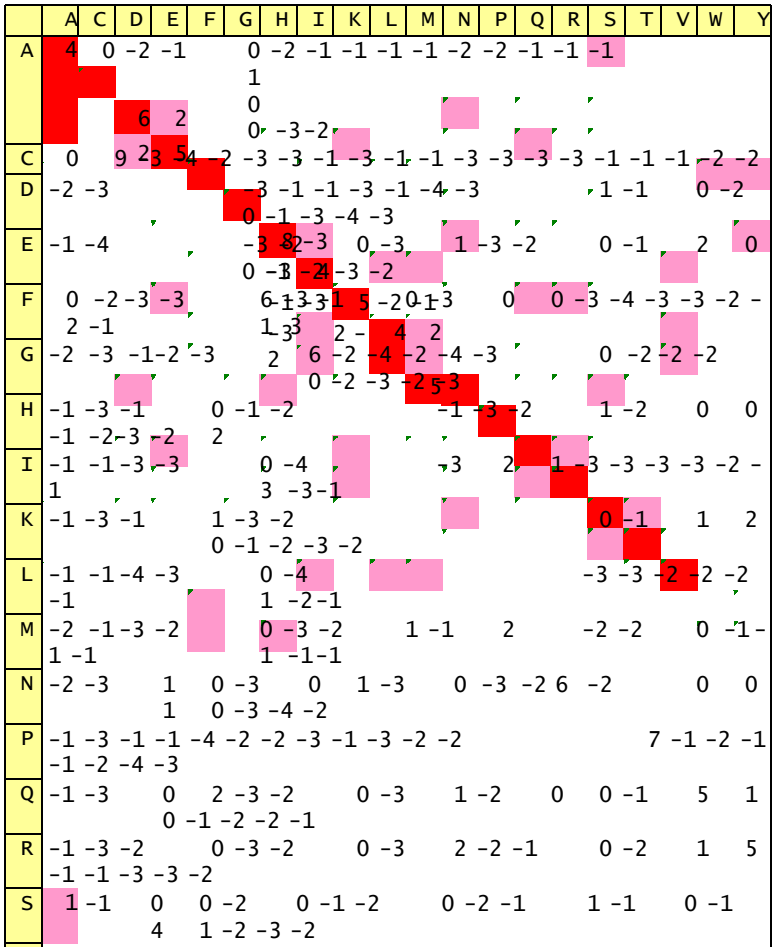
- To reduce multiple contributions to amino acid pair frequencies from the most closely related members of a family, similar sequences are merged within block.
- BLOSUM p matrix is created by merging sequences with no less than p% similarity.
- For example,
  - AVAAA AVAAA AVAAA AVLAA VVAAL
- Note that the first 4 sequences have at least 80% similarity. The similarity of the last sequence with the other 4 sequences is less than 62%.
- For BLOSUM 62, we group the first 4 sequeneces and we get
  - AV[$A_{0.75}L_{0.25}$]AA VVAAL
- Then, $p_{AV} = 1 / 5$ and $P_{AL} = (0.25 + 1)/5$.

Relationship between BLOSUM and PAM

- Relationship between BLOSUM and PAM
  - BLOSUM 80 ≈ PAM 1
  - BLOSUM 62 ≈ PAM 120
  - BLOSUM 45 ≈ PAM 250

- BLOSUM 62 is the default matrix for BLAST 2.0

BLOSUM 62


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

References

- R. F. Doolittle, M. Hunkapiller, L. E. hood, S. Devare, K. Robbins, S. Aaronson, and H. Antoniades. Simian sarcoma virus onc gene v-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor. Science, 221:275-277, 1983.
- S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48:443-453, 1970.
- T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. Journal of Molecular Biology, 147:195-197, 1981.

References

- M. Waterman, T. F. Smith, and W. A. Beyer. Some biological sequence metrics. Advances in Mathematics, 20:367-387, 1976.
- M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff (ed) Atlas of Protein Sequence and Structure, volume 5, supplement 3, pp. 345-352. National Biomedical Research Foundation, Washington, DC, 1978.
- Henikoff, S. and Henikoff, J. Amino acid substitution matrices from protein blocks. Proc. Natl. Acad. Sci. USA. 89(biochemistry): 10915 - 10919 (1992).