# BioInformatics 2009-2010

## 12 lectures -- Pietro Lio', pl219

Bioinformatics is focused on developing algorithms to be used in biological and medical researches. Molecular biologists generate massive amounts of information that can only be efficiently analyzed with computers.

Computer science could provide the abstraction needed for consolidating knowledge of biomolecular systems.

Both DNA sequence and protein structure research have adopted good abstractions: 'DNA-as-string' (a mathematical string is a finite sequence of symbols) and'protein-as-three-dimensional-labelled-graph', respectively.

1

# BioInformatics

## Content

- Working with sequence data
  - Algorithms focusing on strings (lect 1-4)
  - Algorithms on Trees (lect 5-7)
  - Information theory and DNA (lect 8)
  - Applications of Hidden Markov models (lect 9)
- working with microarray data
  - Algorithms for Clustering (lect 10)
  - Algorithms for Genetic Networks (lect 11)
  - Algorithms for System Biology (lect 12)

  - These algorithms are useful in life sciences and are also used in other fields such as economic and social sciences.

2

# DNA SEQUENCES AS STRINGS

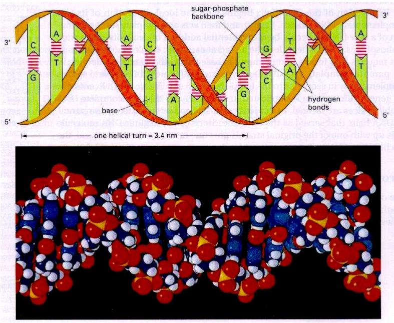
DNA: 4-letter alphabet, A (adenine), T (thymine), C (cytosine) and G (guanine). In the double helix A pairs with T, C with G ; RNA: same as DNA but T -> U (uracil)
3 letters (triplet – a codon) code for one amino acid in a protein.

```
5-CCTGAGCCAACTATTGATGAA-3
3-GGACTCGGTTGATAACTACTT-5
```

↓

RNA

CCUGAGCCAACUAUUGAUGAA



Gene: hereditary information located on the chromosomes and consisting of DNA (say 1000 bases);

Genome: an organism's genetic material; human genome= 46 pieces (chromosomes) with overall length 3 x $10^9$ base.

# Proteins as 3D labelled graphs

units are the 20 amino acids A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y.

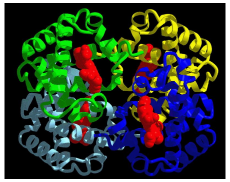
```
5-CCTGAGCCAACTATTGATGAA-3
3-GGACTCGGTTGATAACTACTT-5
```

↓

CCUGAGCCAACUAUUGAUGAA

↓

PEPTIDE



DNA

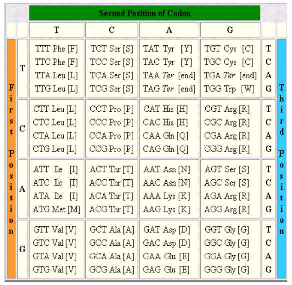
↓ transcription

RNA

↓ translation

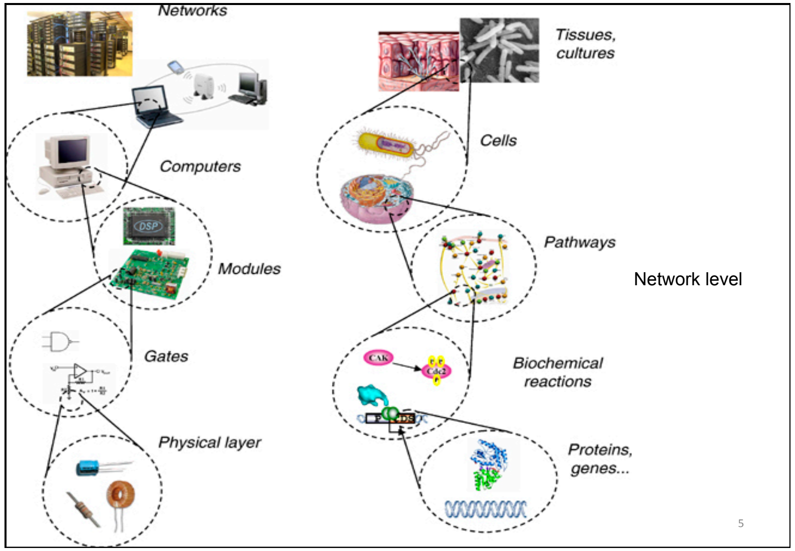
Protein

Genetic Code



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

Third Position



Networks

Computers

Modules

Gates

Physical layer

Tissues, cultures

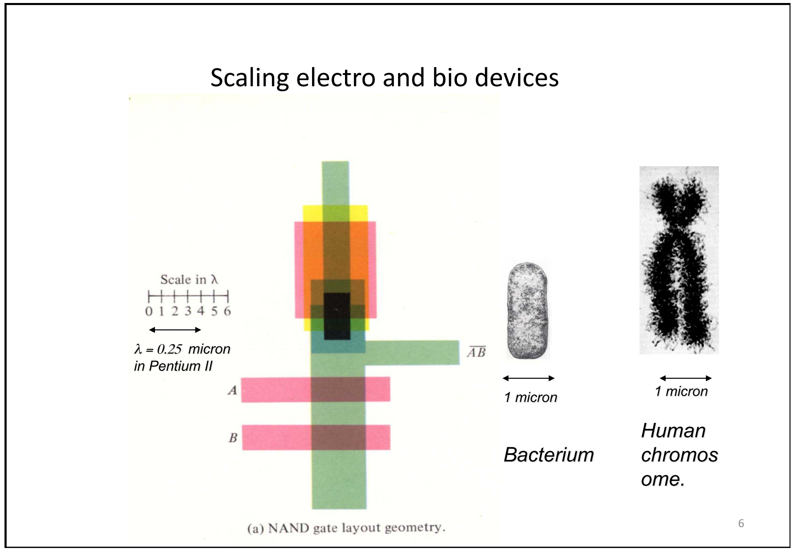
Cells

Pathways

Network level

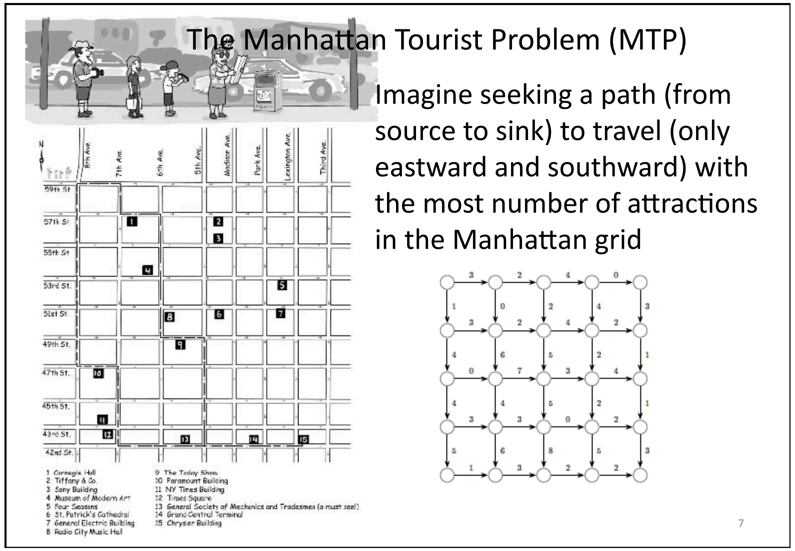
Biochemical reactions

Proteins, genes...



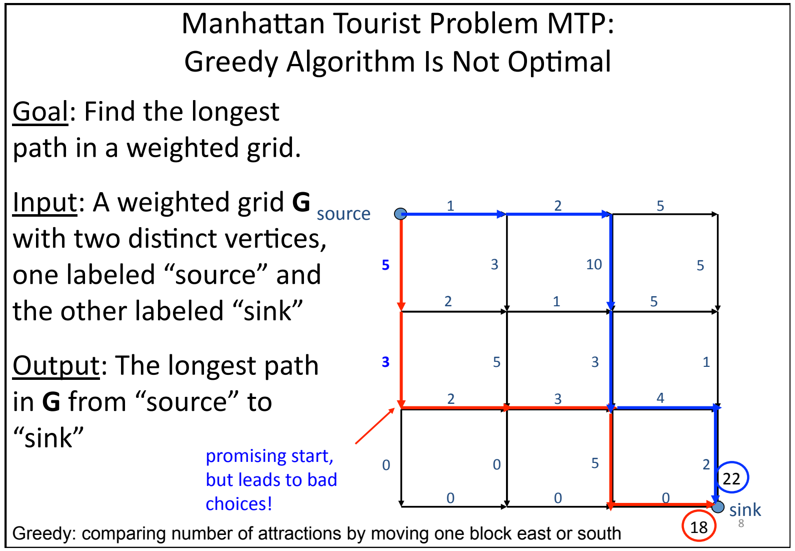
## Scaling electro and bio devices

(a) NAND gate layout geometry.



## The Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions in the Manhattan grid



## Manhattan Tourist Problem MTP: Greedy Algorithm Is Not Optimal

Goal: Find the longest path in a weighted grid.

Input: A weighted grid **G** with two distinct vertices, one labeled "source" and the other labeled "sink"

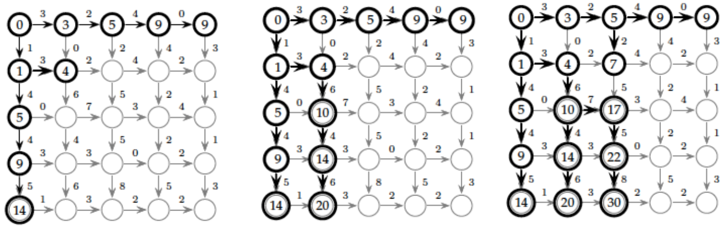
Output: The longest path in **G** from "source" to "sink"

promising start, but leads to bad choices!

Greedy: comparing number of attractions by moving one block east or south

## Computing the score for a point (i,j) by the recurrence relation:

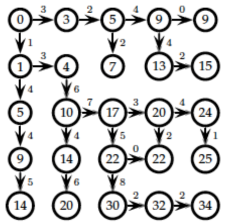
- Calculate optimal path score for each vertex in the graph
- Each vertex's score is the maximum of the prior vertices score plus the weight of the respective edge in between
- The only hitch is that one must decide on the order in which visit the vertices; By the time the vertex x is analyzed, the values sy for all its predecessors y should be computed.



$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of the edge between } (i-1, j) \text{ and } (i, j) \\ s_{i,j-1} + \text{weight of the edge between } (i, j-1) \text{ and } (i, j) \end{cases}$$

The running time is **n x m** for a **n** by **m** grid

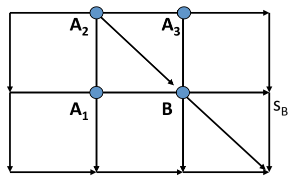
## MTP: Dynamic Programming



```
MANHATTANTOURIST($\overset{\downarrow}{\mathbf{w}}, \vec{\mathbf{w}}, n, m$)
1 $s_{0,0} \leftarrow 0$
2 for $i \leftarrow 1$ to $n$
3     $s_{i,0} \leftarrow s_{i-1,0} + \overset{\downarrow}{w}_{i,0}$
4 for $j \leftarrow 1$ to $m$
5     $s_{0,j} \leftarrow s_{0,j-1} + \vec{w}_{0,j}$
6 for $i \leftarrow 1$ to $n$
7     for $j \leftarrow 1$ to $m$
8         $s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} + \overset{\downarrow}{w}_{i,j} \\ s_{i,j-1} + \vec{w}_{i,j} \end{cases}$
9 return $s_{n,m}$
```

MANHATTANTOURIST computes the length of the longest path in the grid, but does not give the path itself. Lines 1 through 5 set up the initial conditions on the matrix s, and line 8 correspondsto the recurrence that allows us to fill in later table entries based on earlier ones. Most of the dynamic programming algorithms we will develop in the context of DNA sequence comparison will look just like MANHATTANTOURIST with only minor changes. Many problems in bioinformatics can be solved efficiently by the application of the dynamic programming technique, once they are cast as traveling in a Manhattan-like grid.

## Manhattan Is Not A Perfect Grid



- The score at point B is given by:

$$s_B = \max \text{ of } \begin{cases} s_{A1} + \text{weight of the edge } (A_1, B) \\ s_{A2} + \text{weight of the edge } (A_2, B) \\ s_{A3} + \text{weight of the edge } (A_3, B) \end{cases}$$

Computing the score for point **x** is given by the recurrence relation:

$$s_x = \max_{\text{of}} \begin{cases} s_y + \text{weight of vertex } (y, x) \text{ where} \\ \qquad y \in \text{Predecessors}(x) \end{cases}$$

- Predecessors (x) – set of vertices that have edges leading to x
- The running time for a graph G(**V**, **E**) (**V** is the set of all vertices and **E** is the set of all edges) is O(**E**) since each edge is evaluated once

## Alignment: 2 row representation

Given 2 DNA sequences **v** and **w**:

**v** : A T G T T A T **m** = 7
**w** : A T C G T A C **n** = 7

Alignment : 2 * **k** matrix ( **k** > **m**, **n** )

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| letters of **v** | A | T | -- | G | T | T | A | T | -- |
| letters of **w** | A | T | C | G | T | -- | A | -- | C |

4 matches 2 insertions 2 deletions

Longest Common Subsequence (LCS) –the simplest form of sequence alignment – allows only insertions and deletions (no mismatches). In the LCS Problem, we scored 1 for matches and 0 for indels; in real analysis we consider penalising indels and mismatches with negative scores.

- Given two sequences

  **v** = $v_1\ v_2 \ldots v_m$ and **w** = $w_1\ w_2 \ldots w_n$

- The LCS of **v** and **w** is a sequence of positions in

  **v**: $1 \le i_1 < i_2 < \ldots < i_t \le m$

and a sequence of positions in

**w**: $1 \le j_1 < j_2 < \ldots < j_t \le n$

such that $i_t$-th letter of **v** equals to $j_t$-th letter of **w** and **t** is maximal

# LCS: Example

| i coords: | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| elements of v | | A | T | -- | C | -- | T | G | A | T | C |
| elements of w | | -- | T | G | C | A | T | -- | A | -- | C |
| j coords: | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 6 | 6 | 7 |

(0,0)→ (1,0)→ (2,1)→ (2,2)→ (3,3)→ (3,4)→ (4,5)→ (5,5)→ (6,6)→ (7,6)→ (8,7)

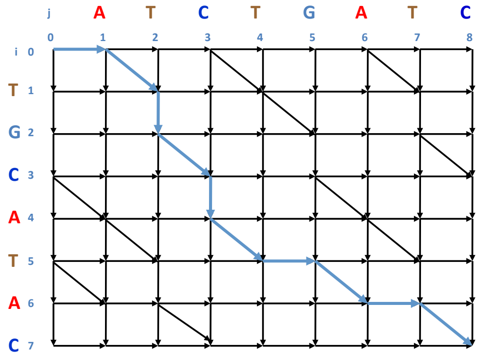
Matches shown in red

positions in v: 2 < 3 < 4 < 6 < 8

positions in w: 1 < 3 < 5 < 6 < 7

Every common subsequence is a path in 2-D grid

# LCS Problem as Manhattan Tourist Problem- Edit Graph for LCS Problem



Every path is a common subsequence.

Every diagonal edge adds an extra element to common subsequence

**LCS Problem:** Find a path with maximum number of diagonal edges

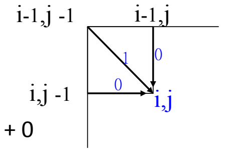
# Computing LCS

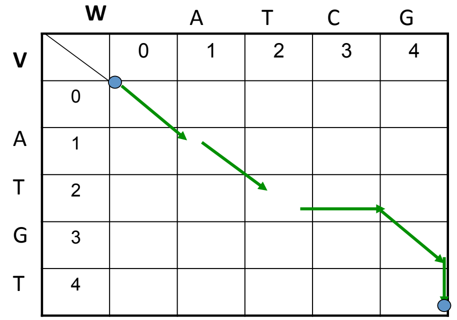
Let $\mathbf{v}_i$ = prefix of **v** of length i: $v_1 \ldots v_i$

and $\mathbf{w}_j$ = prefix of **w** of length j: $w_1 \ldots w_j$

The length of LCS($\mathbf{v}_i$,$\mathbf{w}_j$) is computed by:

$$s_{i,j} = \text{MAX} \begin{cases} s_{i-1,j} + 0 \\ s_{i,j-1} + 0 \\ s_{i-1,j-1} + 1, \quad \text{if } v_i = w_j \end{cases}$$





Every Path in the Grid Corresponds to an Alignment

↘↘→↘↓

```
    0 1 2  2 3 4
V =  A T -  G T
     | |    |
W=   A T C G –
     0 1 2  3 4 4
```

The Edit distance between two strings is the minimum number of operations (insertions, deletions, and substitutions) to transform one string into the other

Hamming distance always compares $i$-th letter of **v** with $i$-th letter of **w**

```
V  =  ATATATAT
      ||||||||
W  =  TATATATA
```

Just one shift - - - - → Make it all line up

Edit distance may compare $i$-th letter of **v** with $j$-th letter of **w**

```
V  =  -ATATATAT
       ||||||||
W  =  TATATATA-
```

Hamming distance:

d(**v**, **w**)=8

Computing Hamming distance is a trivial task

Edit distance:

d(**v**, **w**)=2

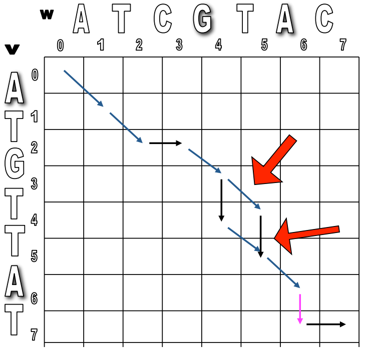
Computing edit distance is a non-trivial task

# Edit Distance: Example

TGCATAT → ATCCGAT in 4 steps

TGCATAT → (insert A at front)
ATGCATAT → (delete 6th T)
ATGCATA → (substitute G for 5th A)
ATGCGTA → (substitute C for 3rd G)
ATCCGAT (Done)

# Alignment as a Path in the Edit Graph



Old Alignment

```
     0122345677
v=   AT_GTTAT_
w=   ATCGT_A_C
     0123455667
```

New Alignment

```
     0122345677
v=   AT_GTTAT_
w=   ATCG_TA_C
     0123445667
```

Two similar alignments; the score is 5 for both the alignment paths.

# Alignment: Dynamic Programming

$$s_{i,j} = \max \begin{cases} s_{i-1,\, j-1}+1 \text{ if } v_i = w_j & \searrow \\ s_{i-1,\, j}+0 & \downarrow \\ s_{i,\, j-1}+0 & \rightarrow \end{cases}$$

This recurrence corresponds to the Manhattan Tourist problem (three incoming edges into a vertex) with all horizontal and vertical edges weighted by zero.

# LCS Algorithm

```
LCS(v, w)
for $i \leftarrow 0$ to $n$
    $s_{i,0} \leftarrow 0$
for $j \leftarrow 1$ to $m$
    $s_{0,j} \leftarrow 0$
for $i \leftarrow 1$ to $n$
    for $j \leftarrow 1$ to $m$
        $s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1}+1, & \text{if } v_i = w_j \end{cases}$
        $b_{i,j} \leftarrow \begin{cases} \text{"}\uparrow\text{"} & \text{if } s_{i,j} = s_{i-1,j} \\ \text{"}\leftarrow\text{"} & \text{if } s_{i,j} = s_{i,j-1} \\ \text{"}\nwarrow\text{"}, & \text{if } s_{i,j} = s_{i-1,j-1}+1 \end{cases}$
return $(s_{n,m}, \mathbf{b})$
```

```
PRINTLCS(b, v, i, j)
if $i = 0$ or $j = 0$
    return
if $b_{i,j} = \text{"}\nwarrow\text{"}$
    PRINTLCS(b, v, $i-1$, $j-1$)
    print $v_i$
else
    if $b_{i,j} = \text{"}\uparrow\text{"}$
        PRINTLCS(b, v, $i-1$, $j$)
    else
        PRINTLCS(b, v, $i$, $j-1$)
```

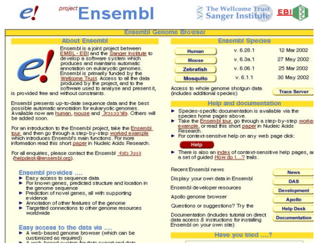
The above recursive program prints out the longest common subsequence using the information stored in b. The initial invocation that prints the solution to the problem is PRINTLCS(b, v, n,m).

**Human Genome Browser Gateway** http://genome.ucsc.edu/

**Ensembl** Genome Browser (EMBL-EBI-Sanger)
www.**ensembl**.org

http://www.ebi.ac.uk

Ensembl Human Database
http://www.ensembl.org/



e! project Ensembl

The Wellcome Trust Sanger Institute EBI

Ensembl Genome Browser

About Ensembl

Ensembl is a joint project between EMBL - EBI and the Sanger Institute to develop a software system which produces and maintains automatic annotation on eukaryotic genomes. Ensembl is primarily funded by the Wellcome Trust. Access to all the data produced by the project, and to the software used to analyse and present it, is provided free and without constraints.

Ensembl presents up-to-date sequence data and the best possible automatic annotation for eukaryotic genomes. Available now are human, mouse and Drosophila. Others will be added soon.

For an introduction to the Ensembl project, take the Ensembl tour, and then go through a step-by-step worked example which introduces Ensembl's main functions. For more information read this short paper in Nucleic Acids Research.

For all enquiries, please contact the Ensembl Help Desk (helpdesk@ensembl.org).

Ensembl provides ....
- Easy access to sequence data
- For known genes, predicted structure and location in the genome sequence
- Prediction of novel genes, all with supporting evidence
- Annotation of other features of the genome
- Targetted connections to other genome resources worldwide

Easy access to the data via ....
- A web-based genome browser (which can be customized as required)

Ensembl Species

| | | |
|---|---|---|
| Human | v. 6.28.1 | 12 Mar 2002 |
| Mouse | v. 6.3a.1 | 27 May 2002 |
| Zebrafish | v. 6.06.1 | 25 Mar 2002 |
| Mosquito | v. 6.1.1 | 30 May 2002 |

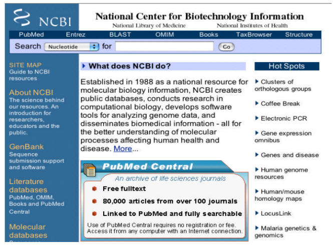
Access to whole genome shotgun data (includes additional species) Trace Server

Help and documentation
- Species-specific documentation is available via the species home pages above.
- Take the Ensembl tour, go through a step-by-step worked example, or read this short paper in Nucleic Acids Research.
- For context-sensitve help on any web page click: Help
- There is also an index of context-sensitive help pages, a set of guided How do I...? trails.

| | |
|---|---|
| Recent Ensembl news | News |
| Display your own data in Ensembl | DAS |
| Ensembl developer resources | Development |
| Apollo genome browser | Apollo |
| Questions or suggestions? Try the | Help Desk |
| Documentation (includes tutorial on direct data access & instructions for installing Ensembl on your own site) | Documentation |

Have you tried ....?

National Center for Biotechnology Information
http://www.ncbi.nlm.nih.gov/



NCBI National Center for Biotechnology Information
National Library of Medicine National Institutes of Health

SITE MAP
Guide to NCBI resources

About NCBI
The science behind our resources. An introduction for researchers, educators and the public.

GenBank
Sequence submission support and software

Literature databases
PubMed, OMIM, Books and PubMed Central

Molecular databases

What does NCBI do?

Established in 1988 as a national resource for molecular biology information, NCBI creates public databases, conducts research in computational biology, develops software tools for analyzing genome data, and disseminates biomedical information - all for the better understanding of molecular processes affecting human health and disease. More...

PubMed Central
An archive of life sciences journals
- Free fulltext
- 80,000 articles from over 100 journals
- Linked to PubMed and fully searchable

Use of PubMed Central requires no registration or fee. Access it from any computer with an Internet connection.

Hot Spots
- Clusters of orthologous groups
- Coffee Break
- Electronic PCR
- Gene expression omnibus
- Genes and disease
- Human genome resources
- Human/mouse homology maps
- LocusLink
- Malaria genetics & genomics

Fasta Format

```
>gi|18089116|gb|BC020718.1| Homo sapiens I factor
AAATTTCAAAAGAATACCTGGAGTGGAAAAGAGTTCTCAGCAGAGACAAAGACCCCGAACACCTCCAACA
TGAAGCTTCTTCATGTTTTCCTGTTATTTCTGTGCTTCCACTTAAGGTTTTGCAAGGTCACTTATACATC
TCAAGAGGATCTGGTGGAGAAAAAGTGCTTAGCAAAAAAATATACTCACCTCTCCTGCGATAAAGTCTTC
TGCCAGCCATGGCAGAGATGCATTGAGGGCACCTGTGTTTGTAAACTACCGTATCAGTGCCCAAAGAATG
GCACTGCAGTGTGTGCAACTAACAGGAGAAGCTTCCCAACATACTGTCAACAAAAGAGTTTGGAATGTCT
TCATCCAGGGACAAAGTTTTTAAATAACGGAACATGCACAGCCGAAGGAAAGTTTAGTGTTTCCTTGAAG
CATGGAAATACAGATTCAGAGGGAATAGTTGAAGTAAAACTTGTGGACCAAGATAAGACAATGTTCATAT
GCAAAAGCAGCTGGAGCATGAGGGAAGCCAACGTGGCCTGCCTTGACCTTGGGTTTCAACAAGGTGCTGA
TACTCAAAGAAGGTTTAAGTTGTCTGATCTCTCTATAAATTCCACTGAATGTCTACATGTGCATTGCCGA
GGATTAGAGACCAGTTTGGCTGAATGTACTTTTACTAAGAGAAGAACTATGGGTTACCAGGATTTCGCTG
ATGTGGTTTGTTATACACAGAAAGCAGATTCTCCAATGGATGACTTCTTTCAGTGTGTGAATGGGAAATA
CATTTCTCAGATGAAAGCCTGTGATGGTATCAATGATTGTGGAGACCAAAGTGATGAACTGTGTTGTAAA
GCATGCCAAGGCAAAGGCTTCCATTGCAAATCGGGTGTTTGCATTCCAAGCCAGTATCAATGCAATGGTG
AGGTGGACTGCATTACAGGGGAAGATGAAGTTGGCTGTGCAGGCTTTGCATCTGTGGCTCAAGAAGAAAC
AGAAATTTTGACTGCTGACATGGATGCAGAAAGAAGACGGATAAAATCATTATTACCTAAACTATCTTGT
GGAGTTAAAAACAGAATGCACATTCGAAGGAAACGAATTGTGGGAGGAAAGCGAGCACAACTGGGAAAAA
TGAAGCAAATCTCATTGGATATTTTTAAAGGTCTCCACAGAGTTTATGCCATATTGGAATTTTGTTGTAT
AATTCTCAAATAAATATTTTGGTGAAGCCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

GenBank Format

```
1:  BC020718. Reports  Homo sapiens I fa...[gi:18089116]  Links
LOCUS       BC020718               1249 bp    mRNA    linear   PRI 06-OCT-2003
DEFINITION  Homo sapiens I factor (complement), mRNA (cDNA clone MGC:22501
VERSION     BC020718.1  GI:18089116
KEYWORDS    MGC.
SOURCE      Homo sapiens (human)
FEATURES             Location/Qualifiers
     source          1..1249
                     /organism="Homo sapiens"
                     /mol_type="mRNA"
                     /db_xref="taxon:9606"
                     /clone="MGC:22501 IMAGE:4716122"
                     /tissue_type="Liver"
                     /clone_lib="NIH_MGC_76"
                     /lab_host="DH10B"
                     /note="Vector: pDNR-LIB"
     gene            1..1249
                     /gene="IF"
                     /note="synonym: FI"
                     /db_xref="GeneID:3426"
                     /db_xref="MIM:217030"
     CDS             70..1203
                     /gene="IF"
                     /codon_start=1
                     /product="IF protein"
                     /protein_id="AAH20718.1"
                     /db_xref="GI:18089117"
                     /db_xref="GeneID:3426"
                     /db_xref="MIM:217030"
                     /translation="MKLLHVFLLFLCFHLRFCKVTYTSQEDLVEKKCLAKKYTHLSCD
                     KVFCQPWQRCIEGTCVCKLPYQCPKNGTAVCATNRRSFPTYCQQKSLECLHPGTKFLN
                     NGTCTAEGKFSVSLKHGNTDSEGIVEVKLVDQDKTMFICKSSWSMREANVACLDLGFQ
                     QGADTQRRFKLSDLSINSTECLHVHCRGLETSLAECTFTKRRTMGYQDFADVVCYTQK
                     ADSPMDDFFQCVNGKYISQMKACDGINDCGDQSDELCCKACQGKGFHCKSGVCIPSQY
                     QCNGEVDCITGEDEVGCAGFASVAQEETEILTADMDAERRRIKSLLPKLSCGVKNRMH
                     IRRKRIVGGKRAQLGKMKQISLDIFKGLHRVYAILEFCCIILK"
```

- BioJava – www.biojava.org
- BioPerl – www.bioperl.org (till now the dominant language in bioinformatics; loosely typed)
- BioPython – www.biopython.org
- BioCorba – www.biocorba.org (can be used to tying it all together; strongly typed)
- C++ www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/

## BioInformatics 2: sequence alignment

```
    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC

-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

Definition

Given two strings $x = x_1x_2...x_M$, $y = y_1y_2...y_N$,

an alignment is an assignment of gaps to positions 0,…, N in x, and 0,…, N in y, so as to line up each letter in one sequence with either a letter, or a gap in the other sequence

| | |
|---|---|
| F[i-1,j-1] | F[i,j-1] |
| F[I-1,j] | F[i,j] |

Notice three possible cases:

1. $x_i$ aligns to $y_j$
   $x_1……x_{i-1}$ $x_i$
   $y_1……y_{j-1}$ $y_j$

$$F(i,j) = F(i-1, j-1) + \begin{cases} m, \text{ if } x_i = y_j \\ -s, \text{ if not} \end{cases}$$

2. $x_i$ aligns to a gap
   $x_1……x_{i-1}$ $x_i$
   $y_1……y_j$ -

$$F(i,j) = F(i-1, j) - d$$

3. $y_j$ aligns to a gap
   $x_1……x_i$ -
   $y_1……y_{j-1}$ $y_j$

$$F(i,j) = F(i, j-1) - d$$

- How do we know which case is correct?

| | |
|---|---|
| F[i-1,j-1] | F[i,j-1] |
| F[I-1,j] | F[i,j] |

Inductive assumption:

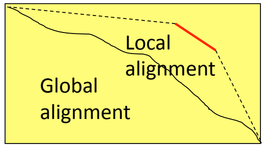
F(i, j-1), F(i-1, j), F(i-1, j-1) are optimal

Then,

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

Where $F(x_i, y_j) = m$, if $x_i = y_j$; $-s$, if not

- The Global Alignment Problem tries to find the longest path between vertices *(0,0)* and *(n,m)* in the edit graph.
- The Local Alignment Problem tries to find the longest path among paths between **arbitrary vertices** (*i,j*) and (*i', j'*) in the edit graph.



- Global Alignment

```
--T—-CC-C-AGT—-TATGT-CAGGGGACACG—A-GCATGCAGA-GAC
  |   || |   ||   |  |  |  |||      ||  |  |  |    |  ||||    |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG—T-CAGAT--C
```

- Local Alignment—better alignment to find conserved segment

```
                    tccCAGTTATGTCAGgggacacgagcatgcagagac
                       ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

## The Needleman-Wunsch Algorithm (Global alignment)

```
1.  Initialization.
    F(0, 0)   = 0
    F(0, j)   = - j × d
    F(i, 0)   = - i × d

3.  Main Iteration. Filling-in partial alignments
      For each   i = 1……M
        For each   j = 1……N
                                   ⎧ F(i-1,j) – d                 [case 1]
          F(i, j)   =   max        ⎨ F(i, j-1) – d                [case 2]
                                   ⎩ F(i-1, j-1) + s(x_i, y_j)    [case 3]

                                   ⎧ UP,     if [case 1]
          Ptr(i,j)  =              ⎨ LEFT    if [case 2]
                                   ⎩ DIAG    if [case 3]

3.  Termination. F(M, N) is the optimal score, and from Ptr(M, N) can trace back optimal
    alignment
```

Complexity: Space: O(mn); Time: O(mn)

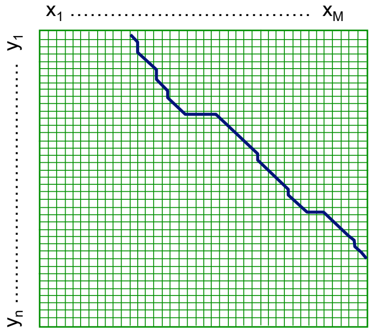
Filling the matrix O(mn)

Backtrace O(m+n)

## The Overlap Detection variant

Maybe it is OK to have an unlimited # of gaps in the beginning and end:

```
----------CTATCACCTGACCTCCAGGCCGATGCCCCTTCCGGC
GCGAGTTCATCTATCAC--GACCGC--GGTCG--------------
```



Changes:

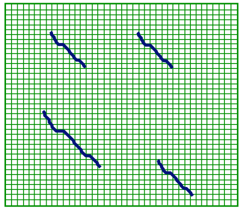
1. Initialization
   For all i, j,
   F(i, 0) = 0
   F(0, j) = 0

2. Termination

$$F_{OPT} = \max \begin{cases} \max_i F(i, N) \\ \max_j F(M, j) \end{cases}$$

## The local alignment: Smith-Waterman algorithm

**Idea**: Ignore badly aligning regions: Modifications to Needleman-Wunsch



e.g. x = aaaacc**cccgggg**

y = **cccggg**aaccaacc

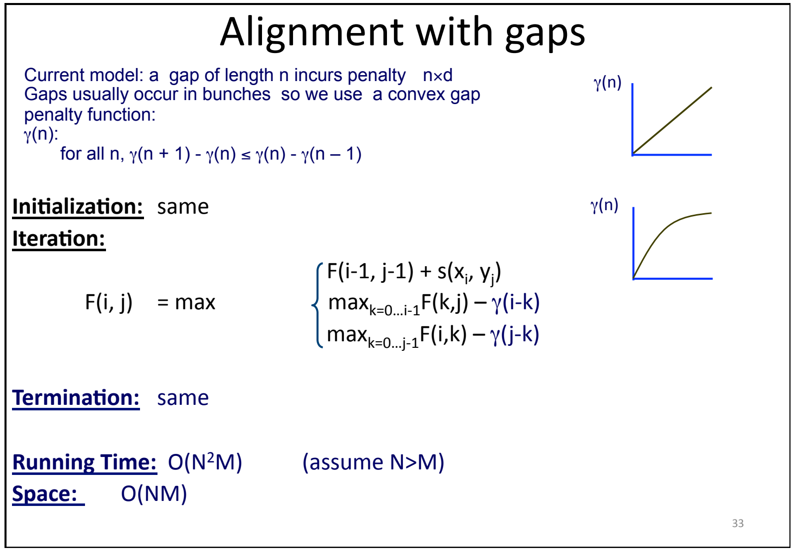
**Initialization**: F(0, j) = F(i, 0) = 0

**Iteration**:

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

**Termination**:

1. If we want the best local alignment…
   $F_{OPT} = \max_{i,j} F(i, j)$
2. If we want all local alignments scoring > t
   For all i, j find F(i, j) > t, and trace back



# Alignment with gaps

Current model: a gap of length n incurs penalty n×d
Gaps usually occur in bunches so we use a convex gap penalty function:
$\gamma(n)$:
for all n, $\gamma(n + 1) - \gamma(n) \leq \gamma(n) - \gamma(n - 1)$

**Initialization:** same

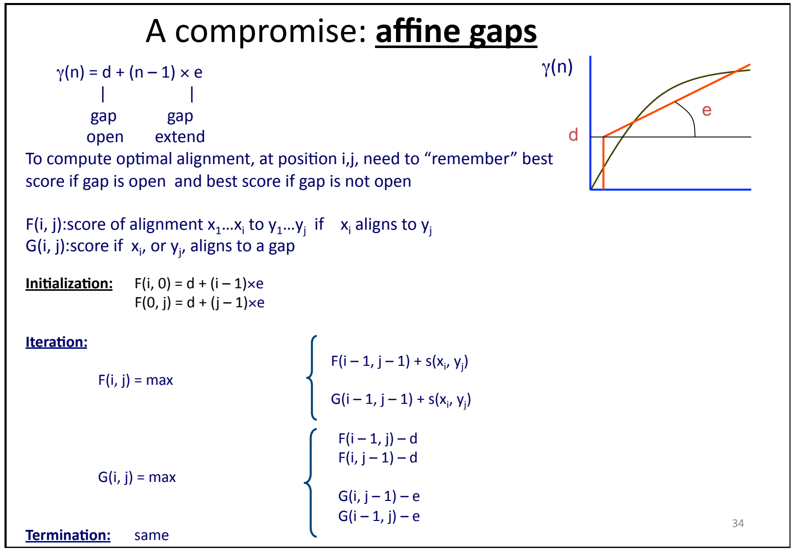
**Iteration:**

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

**Termination:** same

**Running Time:** $O(N^2M)$ (assume N>M)

**Space:** $O(NM)$



# A compromise: **affine gaps**

$\gamma(n) = d + (n - 1) \times e$

| | |
|---|---|
| gap open | gap extend |

To compute optimal alignment, at position i,j, need to “remember” best score if gap is open and best score if gap is not open

F(i, j):score of alignment $x_1 \ldots x_i$ to $y_1 \ldots y_j$ if $x_i$ aligns to $y_j$
G(i, j):score if $x_i$, or $y_j$, aligns to a gap

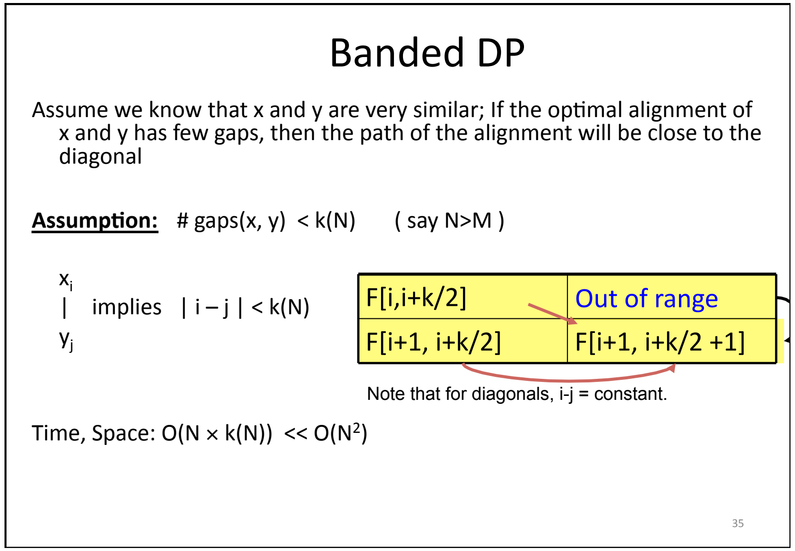
**Initialization:** $F(i, 0) = d + (i - 1) \times e$
$F(0, j) = d + (j - 1) \times e$

**Iteration:**

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

$$G(i, j) = \max \begin{cases} F(i - 1, j) - d \\ F(i, j - 1) - d \\ G(i, j - 1) - e \\ G(i - 1, j) - e \end{cases}$$

**Termination:** same



# Banded DP

Assume we know that x and y are very similar; If the optimal alignment of x and y has few gaps, then the path of the alignment will be close to the diagonal

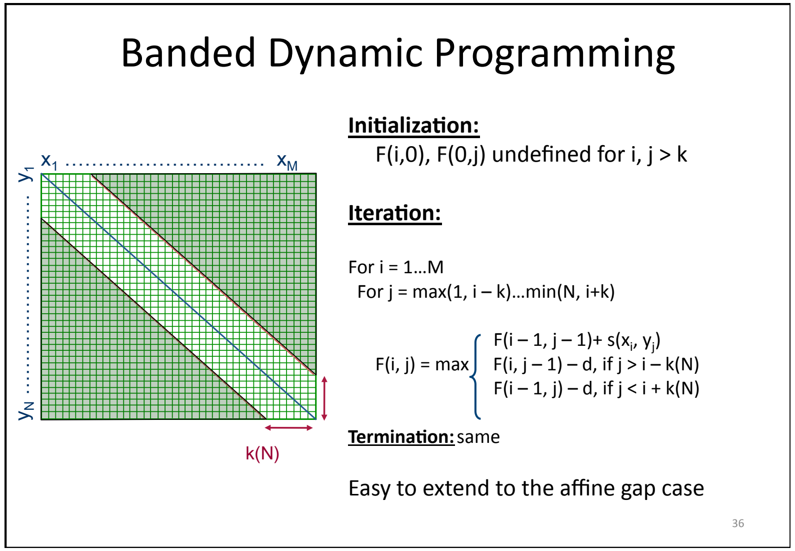
**Assumption:** # gaps(x, y) < k(N) ( say N>M )

$x_i$
|
$y_j$ implies $| i - j | < k(N)$

| F[i,i+k/2] | Out of range |
|---|---|
| F[i+1, i+k/2] | F[i+1, i+k/2 +1] |

Note that for diagonals, i-j = constant.

Time, Space: $O(N \times k(N)) \ll O(N^2)$



# Banded Dynamic Programming

**Initialization:**
F(i,0), F(0,j) undefined for i, j > k

**Iteration:**

```
For i = 1…M
  For j = max(1, i – k)…min(N, i+k)
```

$$F(i, j) = \max \begin{cases} F(i - 1, j - 1) + s(x_i, y_j) \\ F(i, j - 1) - d, \text{ if } j > i - k(N) \\ F(i - 1, j) - d, \text{ if } j < i + k(N) \end{cases}$$

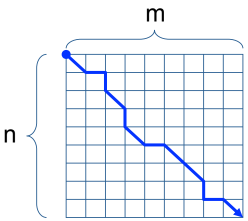
**Termination:** same

Easy to extend to the affine gap case

## Computing Alignment Path Requires Quadratic Memory

**Alignment Path**

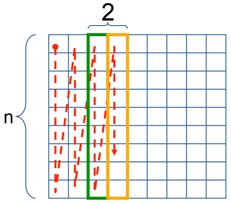
- Space complexity for computing alignment path for sequences of length *n* and *m* is O(*nm*)
- We need to keep all backtracking references in memory to reconstruct the path (backtracking)



## Computing Alignment Score with Linear Memory

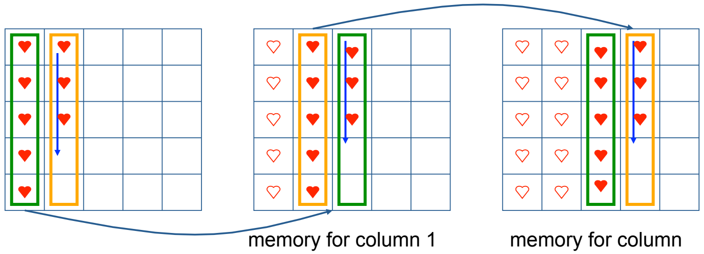
**Alignment Score**

- Space complexity of computing just the score itself is O(*n*)
- We only need the previous column to calculate the current column, and we can then throw away that previous column once we're done using it



## Computing Alignment Score: Recycling Columns

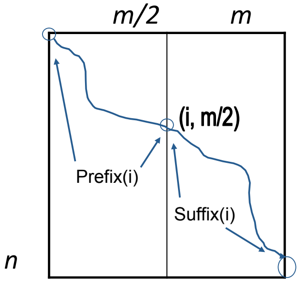
Only two columns of scores are saved at any given time



memory for column 1 is used to calculate column 3

memory for column 2 is used to calculate column 4

## Crossing the Middle Line



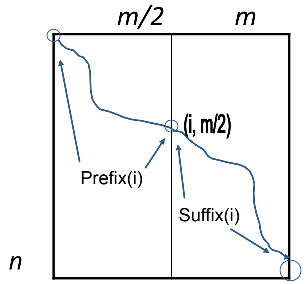
We want to calculate the longest path from (0,0) to (*n*,*m*) that passes through (*i*,*m*/2) where *i* ranges from 0 to *n* and represents the *i*-th row

Define

*length*(*i*)

as the length of the longest path from (0,0) to (*n*,*m*) that passes through vertex (*i*, *m*/2)

## Crossing the Middle Line

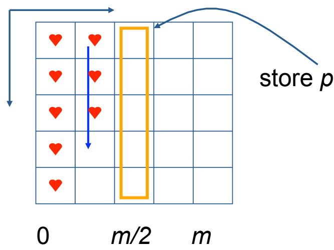


Define (*mid*,*m*/2) as the vertex where the longest path crosses the middle column.

$length(mid)$ = optimal length = $\max_{0 \le i \le n} length(i)$

## Computing Prefix(*i*)

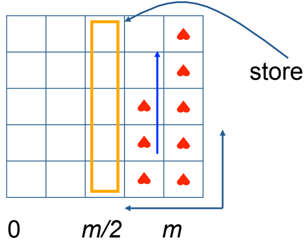
- *prefix*(*i*) is the length of the longest path from (0,0) to (*i*,*m*/2)
- Compute *prefix*(*i*) by dynamic programming in the left half of the matrix



store *prefix*(*i*) column

## Computing Suffix(*i*)

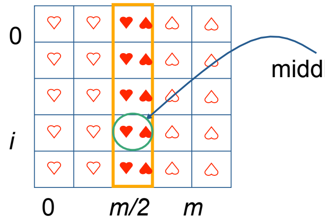
- *suffix*(*i*) is the length of the longest path from (*i*,*m*/2) to *(n,m)*
- *suffix*(*i*) is the length of the longest path from (*n,m*) to (*i*,*m*/2) with all edges reversed
- Compute *suffix*(*i*) by dynamic programming in the right half of the "reversed" matrix



store *suffix*(*i*) column

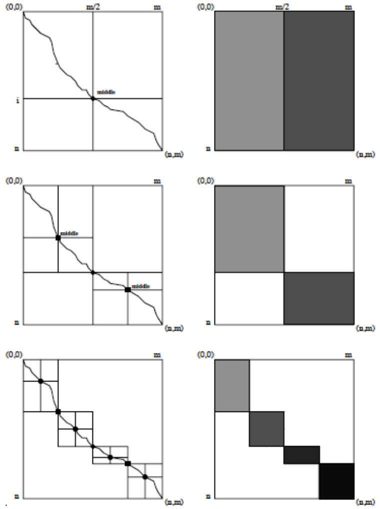
## *Length(i) = Prefix(i) + Suffix(i)*

- Add *prefix*(*i*) and *suffix*(*i*) to compute *length(i):*
  - *length*(*i*)=*prefix*(*i*) + *suffix*(*i*)
- You now have a middle vertex of the maximum path (*i*,*m*/2) as maximum of *length(i)*

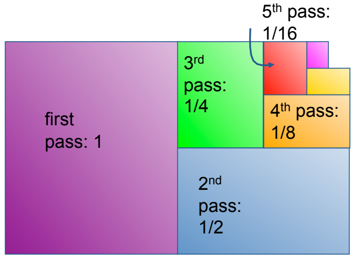


middle point found

### Linear-Space Sequence Alignment



Space-efficient sequence alignment. The computational time (area of solid rectangles) decreases by a factor of 2 at every iteration:
area + area/2 + area/4 + …≤2 x area
and therefore time is O(nm) and space is O(n)



## BioInformatics 3: can we align Sequences in Subquadratic Time?

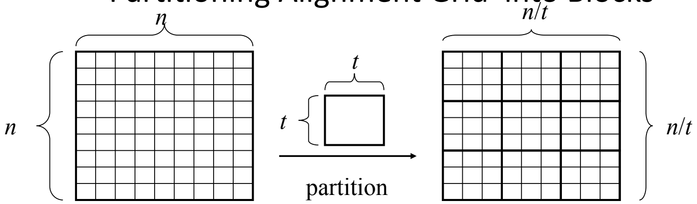
- Partition the $n$ x $n$ grid into blocks of size $t$ x $t$
- We are comparing two sequences, each of size $n$, and each sequence is sectioned off into chunks, each of length $t$
- Sequence $\boldsymbol{u} = u_1 \ldots u_n$ becomes

$$|u_1 \ldots u_t| \; |u_{t+1} \ldots u_{2t}| \; \ldots \; |u_{n-t+1} \ldots u_n|$$

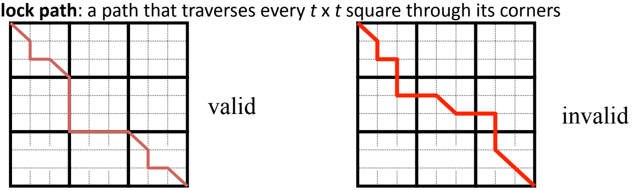
and sequence $\boldsymbol{v} = v_1 \ldots v_n$ becomes

$$|v_1 \ldots v_t| \; |v_{t+1} \ldots v_{2t}| \; \ldots \; |v_{n-t+1} \ldots v_n|$$

## Partitioning Alignment Grid into Blocks



- **Block alignment** of sequences ***u*** and ***v:***
  1. An entire block in ***u*** is aligned with an entire block in ***v***
  2. An entire block is inserted
  3. An entire block is deleted
- **Block path**: a path that traverses every $t$ x $t$ square through its corners



## Block Alignment Problem

- Goal: Find the longest block path through an edit graph
- Input: Two sequences, ***u*** and ***v*** partitioned into blocks of size $t$. This is equivalent to an $n$ x $n$ edit graph partitioned into $t$ x $t$ subgrids
- Output: The block alignment of ***u*** and ***v*** with the maximum score (longest block path through the edit graph)

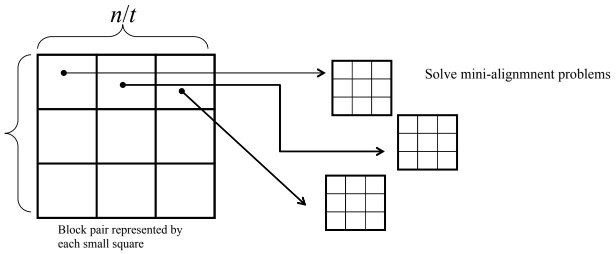
## Constructing Alignments within Blocks

- To solve: compute alignment score $ß_{i,j}$ for each pair of blocks $|u_{(i-1)*t+1}...u_{i*t}|$ and $|v_{(j-1)*t+1}...v_{j*t}|$
- How many blocks are there per sequence?

  (*n*/*t*) blocks of size *t*
- How many pairs of blocks for aligning the two sequences?

  (*n*/*t*) x (*n*/*t*)
- For each block pair, solve a mini-alignment problem of size *t* x *t*

## Constructing Alignments within Blocks


n/t
Solve mini-alignmnent problems
Block pair represented by each small square

## Block Alignment: Dynamic Programming

- Let $s_{i,j}$ denote the optimal block alignment score between the first *i* blocks of ***u*** and first *j* blocks of ***v***

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j} - \sigma_{\text{block}} \\ s_{i,j-1} - \sigma_{\text{block}} \\ s_{i-1,j-1} - \beta_{i,j} \end{array} \right\}$$

$\sigma_{\text{block}}$ is the penalty for inserting or deleting an entire block

$\beta_{i,j}$ is score of pair of blocks in row *i* and column *j*.

## Block Alignment Runtime

- Indices *i,j* range from *0* to *n*/*t*
- Running time of algorithm is

  $O( [n/t]*[n/t]) = O(n^2/t^2)$

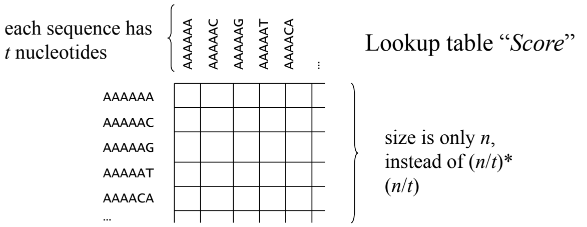
  if we don't count the time to compute each $\beta_{i,j}$
- Computing all $\beta_{i,j}$ requires solving (*n*/*t*)*(*n*/*t*) mini block alignments, each of size (*t***t*)
- Computing all $\beta_{i,j}$ takes time $O([n/t]*[n/t]*t*t) = O(n^2)$
- This is the same as dynamic programming
- How do we speed this up?

## Four Russians Technique

(Arlazarov, Dinic, Kronrod, Faradzev)

- Let $t$ = log($n$), where $t$ is block size, $n$ is sequence size.
- Instead of having $(n/t)*(n/t)$ mini-alignments, construct $4^t$ x $4^t$ mini-alignments for all pairs of strings of $t$ nucleotides (huge size), and put in a lookup table.
- However, size of lookup table is not really that huge if $t$ is small. Let $t$ = (log$n$)/4. Then $4^t$ x $4^t$ = $n$

## Look-up Table for Four Russians Technique



The new lookup table *Score* is indexed by a pair of $t$-nucleotide strings, so

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma_{\text{block}} \\ s_{i,j-1} - \sigma_{\text{block}} \\ s_{i-1,j-1} - Score(i^{\text{th}} \text{ block of } \boldsymbol{v}, j^{\text{th}} \text{ block of } \boldsymbol{u}) \end{cases}$$

## Four Russians Speedup Runtime

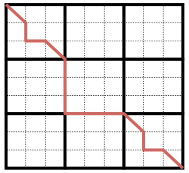
- Since computing the lookup table *Score* of size $n$ takes O($n$) time, the running time is mainly limited by the $(n/t)*(n/t)$ accesses to the lookup table
- Each access takes O(log$n$) time
- Overall running time: O( $[n^2/t^2]$*log$n$ )
- Since $t$ = log$n$, substitute in:
- O( $[n^2/\{\log n\}^2]$*log$n$) $\geq$ O( $n^2$/log$n$ )

## So Far…

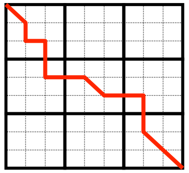
- We can divide up the grid into blocks and run dynamic programming only on the corners of these blocks
- In order to speed up the mini-alignment calculations to under $n^2$, we create a lookup table of size $n$, which consists of all scores for all $t$-nucleotide pairs
- Running time goes from quadratic, O($n^2$), to subquadratic: O($n^2$/log$n$)

## Four Russians Speedup for LCS

- Unlike the block partitioned graph, the LCS path does not have to pass through the vertices of the blocks.



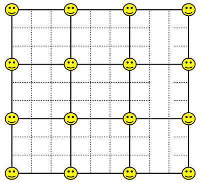
block alignment



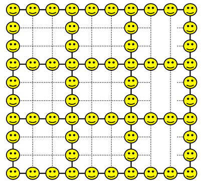
longest common subsequence

## Block Alignment vs. LCS

- In block alignment, we only care about the corners of the blocks.
- In LCS, we care about all points on the edges of the blocks, because those are points that the path can traverse.
- Recall, each sequence is of length *n*, each block is of size *t*, so each sequence has (*n*/*t*) blocks.



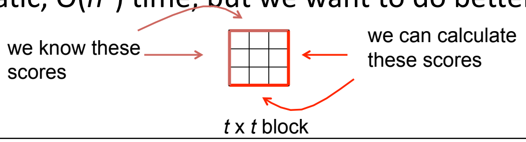
block alignment has $(n/t)*(n/t) = (n^2/t^2)$ points of interest



LCS alignment has $O(n^2/t)$ points of interest

## Traversing Blocks for LCS

- Given alignment scores $s_{i,*}$ in the first row and scores $s_{*,j}$ in the first column of a *t* x *t* mini square, compute alignment scores in the last row and column of the minisquare.
- To compute the last row and the last column score, we use these 4 variables:
  - alignment scores $s_{i}$,* in the first row
  - alignment scores s*$_{,j}$ in the first column
  - substring of sequence u in this block ($4^t$ possibilities)
  - substring of sequence v in this block ($4^t$ possibilities
- If we used this to compute the grid, it would take quadratic, O($n^2$) time, but we want to do better.



we know these scores

we can calculate these scores

*t* x *t* block

## Four Russians Speedup

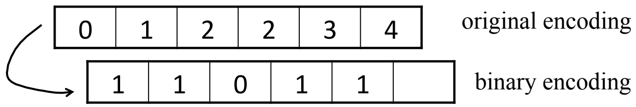
- Build a lookup table for all possible values of the four variables:
  1. all possible scores for the first row $s_{*,j}$
  2. all possible scores for the first column $s_{*,j}$
  3. substring of sequence *u* in this block ($4^t$ possibilities)
  4. substring of sequence *v* in this block ($4^t$ possibilities)
- For each quadruple we store the value of the score for the last row and last column.
- This will be a huge table, but we can eliminate alignments scores that don't make sense

## Reducing Table Size

- Alignment scores in LCS are monotonically increasing, and adjacent elements can't differ by more than 1
- Example: 0,1,2,2,3,4 is ok; 0,1,**2,4**,5,8, is not because 2 and 4 differ by more than 1 (and so do 5 and 8)
- Therefore, we only need to store quadruples whose scores are monotonically increasing and differ by at most 1

## Efficient Encoding of Alignment Scores

- Instead of recording numbers that correspond to the index in the sequences *u* and *v*, we can use binary to encode the differences between the alignment scores



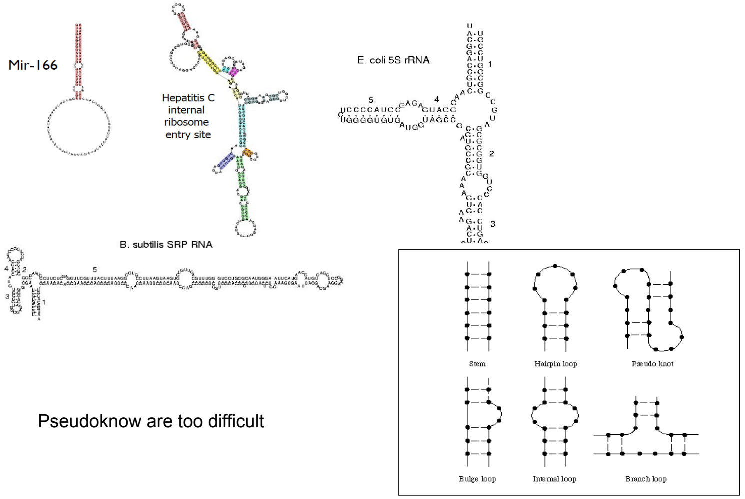
| 0 | 1 | 2 | 2 | 3 | 4 | original encoding |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 |  | binary encoding |

## Reducing Lookup Table Size

- $2^t$ possible scores ($t$ = size of blocks)
- $4^t$ possible strings
  - Lookup table size is $(2^t * 2^t)*(4^t * 4^t) = 2^{6t}$
- Let $t = (\log n)/4$;
  - Table size is: $2^{6((\log n)/4)} = n^{(6/4)} = n^{(3/2)}$
- Time = $O( [n^2/t^2]*\log n )$
- $O( [n^2/\{\log n\}^2]*\log n) \geq O( n^2/\log n )$

Summary: We take advantage of the fact that for each block of t = log(n), we can pre-compute all possible scores and store them in a lookup table of size $n^{(3/2)}$. We used the Four Russian speedup to go from a quadratic running time for LCS to subquadratic running time: $O(n^2/\log n)$.

## RNA structure: great variety!



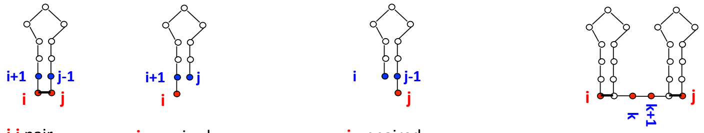
Pseudoknow are too difficult

# RNA Secondary Structure

- Secondary Structure :
  - Set of paired positions on interval $[i,j]$
  - This tells which bases are paired in the subsequence from $x_i$ to $x_j$
- Every optimal structure can be built by extending optimal substructures.
- Suppose we know all optimal substructures of length less than $j-i+1$.

  The optimal substructure for $[i,j]$ must be formed in one of four ways:
  1. *i,j* paired
  2. *i* unpaired
  3. *j* unpaired
  4. combining two substructures

  Note that each of these consists of extending or joining substructures of length less than $j-i+1$.



i,j pair

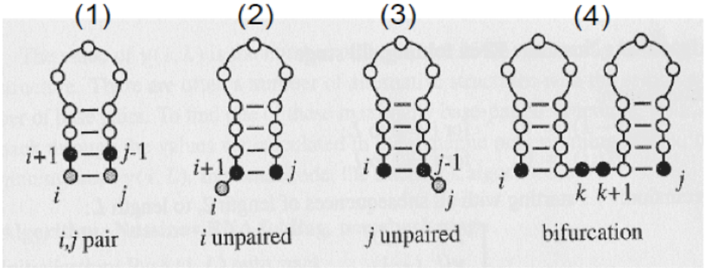
i unpaired

j unpaired

bifurcation

65

## Nussinov algorithm



Objective: To find the secondary structure with the maximal number of base pairs under the pseudo-knot exclusion constraint.

Principle: Recursive procedure (dynamic programming algorithm). Scoring function: sum of base-pair scores, no penalties for loops Optimal score computed from the optimal scores of subsequences.

Filling-stage. Scores for subsequences are recursively computed from and recorded in a quadratic table.

Trace-back: Reconstruction of filling steps indicates optimal structure

Time-complexity: O(N$^3$)

## The Nussinov Folding Algorithm

**Example: GGGAAAUCC**

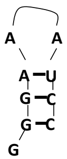
**$\gamma(i,j)$ is the maximum number of base pairs in segment $[i,j]$**

*Initialisation* $\gamma(i,i-1)=0 \;\&\; \gamma(i,i)=0$

**Starting with all subsequences of length 2, to length *L*:**

$$\gamma(i,j)=\max\begin{cases}\gamma(i+1,j)\\ \gamma(i,j-1)\\ \gamma(i+1,j-1)+\delta(i,j)\\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)]\end{cases}$$

**Where $\delta(i,j)=1$ if $x_i$ and $x_j$ are a complementary base pair, and $\delta(i,j)=0$, otherwise.**



j →

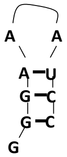
i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | | | | | | | | |
| G | 0 | 0 | | | | | | | |
| G | | 0 | 0 | | | | | | |
| A | | | 0 | 0 | | | | | |
| A | | | | 0 | 0 | | | | |
| A | | | | | 0 | 0 | | | |
| U | | | | | | 0 | 0 | | |
| C | | | | | | | 0 | 0 | |
| | | | | | | | | 0 | 0 |

j →

i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | **0** | | | | | | | |
| G | 0 | 0 | **0** | | | | | | |
| G | | 0 | 0 | **0** | | | | | |
| A | | | 0 | 0 | **0** | | | | |
| A | | | | 0 | 0 | **0** | | | |
| A | | | | | 0 | 0 | **1** | | |
| U | | | | | | 0 | 0 | **0** | |
| C | | | | | | | 0 | 0 | **0** |
| C | | | | | | | | 0 | 0 |



$$\gamma(i,j)=\max\begin{cases}\gamma(i+1,j)\\ \gamma(i,j-1)\\ \gamma(i+1,j-1)+\delta(i,j)\\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)]\end{cases}$$

After scores for subsequences of length 2 (left) and 3 (right)

j →

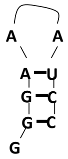
i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | 0 | **0** | | | | | | |
| G | 0 | 0 | 0 | **0** | | | | | |
| G | | 0 | 0 | 0 | **0** | | | | |
| A | | | 0 | 0 | 0 | **0** | | | |
| A | | | | 0 | 0 | 0 | **1** | | |
| A | | | | | 0 | 0 | 1 | **0** | |
| U | | | | | | 0 | 0 | 0 | **0** |
| C | | | | | | | 0 | 0 | 0 |
| | | | | | | | | 0 | 0 |

68

## Nussinov Folding Algorithm
## After scores for subsequences of length 4

$$\gamma(i,j) = \max\begin{cases}\gamma(i+1,j)\\ \gamma(i,j-1)\\ \gamma(i+1,j-1)+\delta(i,j)\\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)]\end{cases}$$



j →

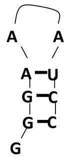
i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | 0 | 0 | **0** | | | | | |
| G | 0 | 0 | 0 | 0 | **0** | | | | |
| G | | 0 | 0 | 0 | 0 | **0** | | | |
| A | | | 0 | 0 | 0 | 0 | **1** | | |
| A | | | | 0 | 0 | **0** | **1** | **1** | |
| A | | | | | 0 | 0 | 1 | 1 | **1** |
| U | | | | | | 0 | 0 | 0 | 0 |
| C | | | | | | | 0 | 0 | 0 |
| C | | | | | | | | 0 | 0 |

**Two optimal substructures for same subsequence**

## Nussinov Folding Algorithm
## After scores for subsequences of length 5

$$\gamma(i,j) = \max\begin{cases}\gamma(i+1,j)\\ \gamma(i,j-1)\\ \gamma(i+1,j-1)+\delta(i,j)\\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)]\end{cases}$$



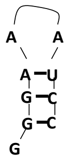
j →

i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | 0 | 0 | 0 | **0** | | | | |
| G | 0 | 0 | 0 | 0 | 0 | **0** | | | |
| G | | 0 | 0 | 0 | 0 | 0 | **1** | | |
| A | | | 0 | 0 | 0 | 0 | 1 | **1** | |
| A | | | | 0 | 0 | 0 | 1 | 1 | **1** |
| A | | | | | 0 | 0 | 1 | 1 | 1 |
| U | | | | | | 0 | 0 | 0 | 0 |
| C | | | | | | | 0 | 0 | 0 |
| C | | | | | | | | 0 | 0 |

## Nussinov Folding Algorithm
## After scores for subsequences of length 6

$$\gamma(i,j) = \max\begin{cases}\gamma(i+1,j)\\ \gamma(i,j-1)\\ \gamma(i+1,j-1)+\delta(i,j)\\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)]\end{cases}$$



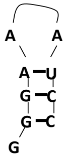
j →

i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | 0 | 0 | 0 | 0 | **0** | | | |
| G | 0 | 0 | 0 | 0 | 0 | 0 | **1** | | |
| G | | 0 | 0 | 0 | 0 | 0 | 1 | **2** | |
| A | | | 0 | 0 | 0 | 0 | 1 | 1 | **1** |
| A | | | | 0 | 0 | 0 | 1 | 1 | 1 |
| A | | | | | 0 | 0 | 1 | 1 | 1 |
| U | | | | | | 0 | 0 | 0 | 0 |
| C | | | | | | | 0 | 0 | 0 |
| C | | | | | | | | 0 | 0 |

## Nussinov Folding Algorithm
## After scores for subsequences of length 7

$$\gamma(i,j) = \max\begin{cases}\gamma(i+1,j)\\ \gamma(i,j-1)\\ \gamma(i+1,j-1)+\delta(i,j)\\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)]\end{cases}$$



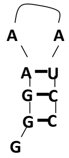
j →

i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | 0 | 0 | 0 | 0 | 0 | **1** | | |
| G | 0 | 0 | 0 | 0 | 0 | 0 | 1 | **2** | |
| G | | 0 | 0 | 0 | 0 | 0 | 1 | 2 | **2** |
| A | | | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| A | | | | 0 | 0 | 0 | 1 | 1 | 1 |
| A | | | | | 0 | 0 | 1 | 1 | 1 |
| U | | | | | | 0 | 0 | 0 | 0 |
| C | | | | | | | 0 | 0 | 0 |
| C | | | | | | | | 0 | 0 |

## Nussinov Folding Algorithm
## After scores for subsequences of length 8

$$\gamma(i,j) = \max\begin{cases}\gamma(i+1,j)\\ \gamma(i,j-1)\\ \gamma(i+1,j-1)+\delta(i,j)\\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)]\end{cases}$$



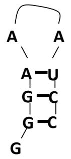
j →

i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | 0 | 0 | 0 | 0 | 0 | 1 | **2** | |
| G | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | **3** |
| G | | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 2 |
| A | | | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| A | | | | 0 | 0 | 0 | 1 | 1 | 1 |
| A | | | | | 0 | 0 | 1 | 1 | 1 |
| U | | | | | | 0 | 0 | 0 | 0 |
| C | | | | | | | 0 | 0 | 0 |
| C | | | | | | | | 0 | 0 |

## Nussinov Folding Algorithm
## After scores for subsequences of length 9

$$\gamma(i,j) = \max\begin{cases}\gamma(i+1,j)\\ \gamma(i,j-1)\\ \gamma(i+1,j-1)+\delta(i,j)\\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)]\end{cases}$$

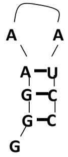


j →

i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | **3** |
| G | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 |
| G | | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 2 |
| A | | | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| A | | | | 0 | 0 | 0 | 1 | 1 | 1 |
| A | | | | | 0 | 0 | 1 | 1 | 1 |
| U | | | | | | 0 | 0 | 0 | 0 |
| C | | | | | | | 0 | 0 | 0 |
| C | | | | | | | | 0 | 0 |

## Nussinov Folding Algorithm
## Traceback



j →

i ↓

| | G | G | G | A | A | A | U | C | C |
|---|---|---|---|---|---|---|---|---|---|
| G | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | **3** |
| G | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 |
| G | | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 2 |
| A | | | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| A | | | | 0 | 0 | 0 | 1 | 1 | 1 |
| A | | | | | 0 | 0 | 1 | 1 | 1 |
| U | | | | | | 0 | 0 | 0 | 0 |
| C | | | | | | | 0 | 0 | 0 |
| C | | | | | | | | 0 | 0 |

## Nussinov algorithm: fill-stage

| | | G | G | C | C | A | G | U | U | C |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| G | 1 | 0 | 0 | 1 | 2 | 2 | 2 | 3 | 4 | 4 |
| G | 2 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 3 |
| C | 3 | | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 |
| C | 4 | | | 0 | 0 | 0 | 1 | 1 | 2 | 2 |
| A | 5 | | | | 0 | 0 | 0 | 1 | 2 | 2 |
| G | 6 | | | | | 0 | 0 | 1 | 1 | 1 |
| U | 7 | | | | | | 0 | 0 | 0 | 0 |
| U | 8 | | | | | | | 0 | 0 | 0 |
| C | 9 | | | | | | | | 0 | 0 |

**Algorithm: Nussinov RNA folding, fill stage**

Initialisation:

$$\gamma(i,i-1) = 0 \quad \text{for } i = 2 \text{ to } L;$$
$$\gamma(i,i) = 0 \quad \text{for } i = 1 \text{ to } L.$$

Recursion: starting with all subsequences of length 2, to length $L$:

$$\gamma(i,j) = \max\begin{cases}\gamma(i+1,j),\\ \gamma(i,j-1),\\ \gamma(i+1,j-1)+\delta(i,j),\\ \max_{i<k<j}[\gamma(i,k)+\gamma(k+1,j)].\end{cases}$$

Scoring system:
$\delta(i,j)$ = 1 for all RNA Watson-Crick base-pairs including G-U else $\delta(i,j)$ = 0.

Blue: addition of unpaired base 3 or 7

Green: addition of paired bases 1,7

Pink: joining of substructures 1..4 and 5..8

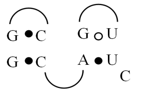
## Nussinov algorithm: trace-back

| | | G | G | C | C | A | G | U | U | C |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| G | 1 | 0 | 0 | 1 | 2 | 2 | 2 | 3 | 4 | 4 |
| G | 2 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 3 |
| C | 3 | | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 |
| C | 4 | | | 0 | 0 | 0 | 1 | 1 | 2 | 2 |
| A | 5 | | | | 0 | 0 | 0 | 1 | 2 | 2 |
| G | 6 | | | | | 0 | 0 | 1 | 1 | 1 |
| U | 7 | | | | | | 0 | 0 | 0 | 0 |
| U | 8 | | | | | | | 0 | 0 | 0 |
| C | 9 | | | | | | | | 0 | 0 |

**Algorithm: Nussinov RNA folding, traceback stage**

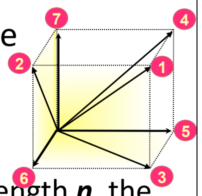
```
Initialisation: Push $(1, L)$ onto stack.
Recursion: Repeat until stack is empty:
    - pop $(i, j)$.
    - if $i >= j$ continue;
      else if $\gamma(i+1, j) = \gamma(i, j)$ push $(i+1, j)$;
      else if $\gamma(i, j-1) = \gamma(i, j)$ push $(i, j-1)$;
      else if $\gamma(i+1, j-1) + \delta_{i,j} = \gamma(i, j)$:
          - record $i, j$ base pair.
          - push $(i+1, j-1)$.
      else for $k = i+1$ to $j-1$: if $\gamma(i,k) + \gamma(k+1, j) = \gamma(i, j)$:
          - push $(k+1, j)$.
          - push $(i, k)$.
          - break.
```

```
current record stack
                1,9
1,9             1,8
1,8             1,4 5,8
1,4     1,4     2,3 5,8
2,3     2,3     3,2 5,8
3,2             5,8
5,8     5,8     6,7
6,7     6,7     7,6
7,6
```



# BioInformatics 4

## Multiple Alignment: Running Time



- For N sequences, there are $2^N$-1 ways

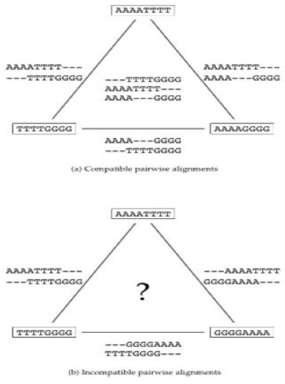
to extend an alignment; for 3 sequences of length ***n***, the run time is *7**n**$^3$*; O(***n**$^3$)*

- For ***k*** sequences, build a ***k***-dimensional Manhattan, with run time (*2$^k$*-1)(***n**$^k$*); O(*2$^k$**n**$^k$*)
- Conclusion: dynamic programming approach for alignment between two sequences is easily extended to ***k*** sequences but it is impractical due to exponential running time

## Combining Optimal Pairwise Alignments into Multiple Alignment

Can combine pairwise alignments into multiple alignment

Can *not* combine pairwise alignments into multiple alignment



(a) Compatible pairwise alignments

(b) Incompatible pairwise alignments

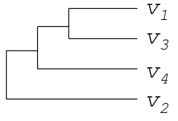
## Progressive Alignment

1) Align each sequence against each other giving a similarity matrix; Similarity = exact matches / sequence length (percent identity)

2) Create Guide Tree using the similarity matrix; Guide tree roughly reflects evolutionary relations

3) Progressive Alignment guided by the tree

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $v_1$ | - | | | |
| $v_2$ | .17 | - | | |
| $v_3$ | .87 | .28 | - | |
| $v_4$ | .59 | .33 | .62 | - |

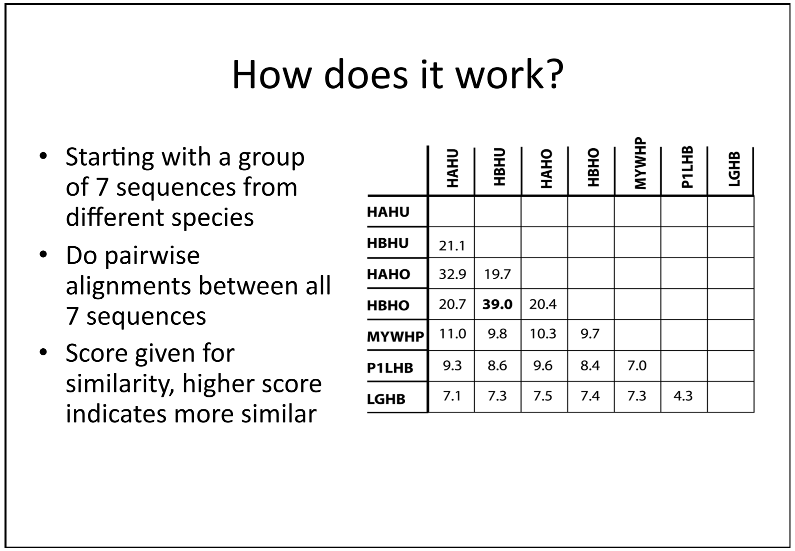


Calculate:

$v_{1,3}$ = alignment $(v_1, v_3)$

$v_{1,3,4}$ = alignment$((v_{1,3}), v_4)$

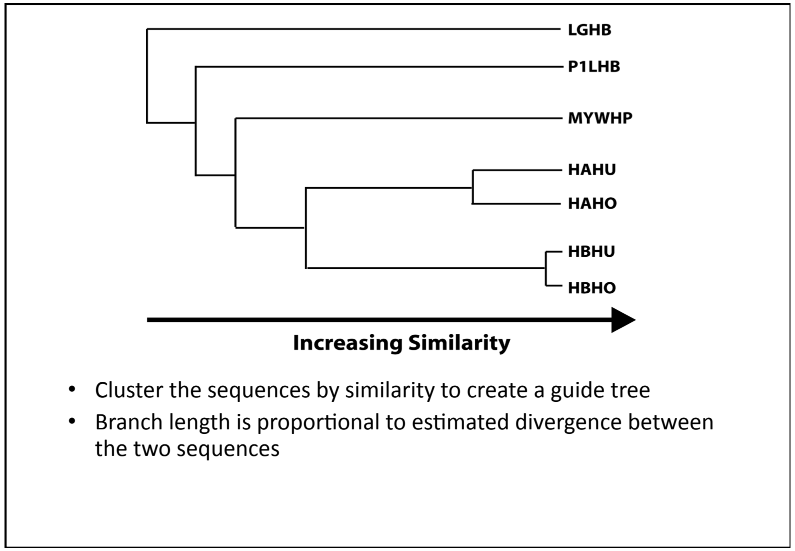
$v_{1,2,3,4}$ = alignment$((v_{1,3,4}), v_2)$



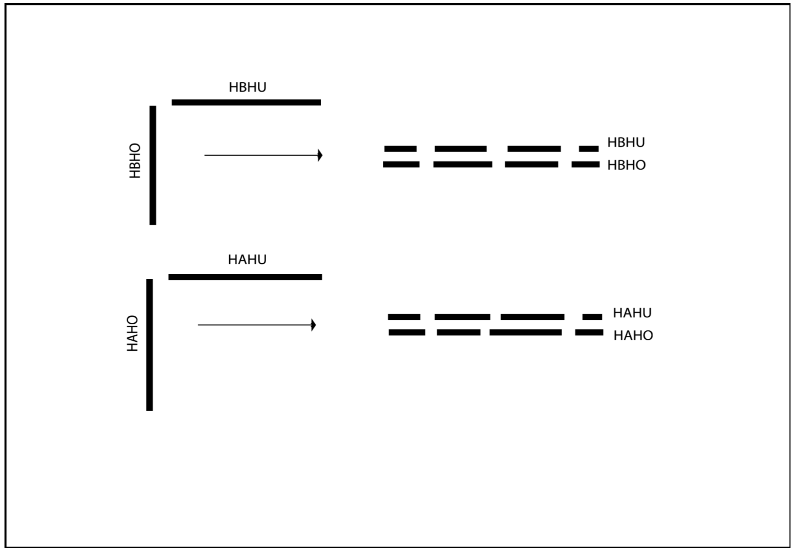
## How does it work?

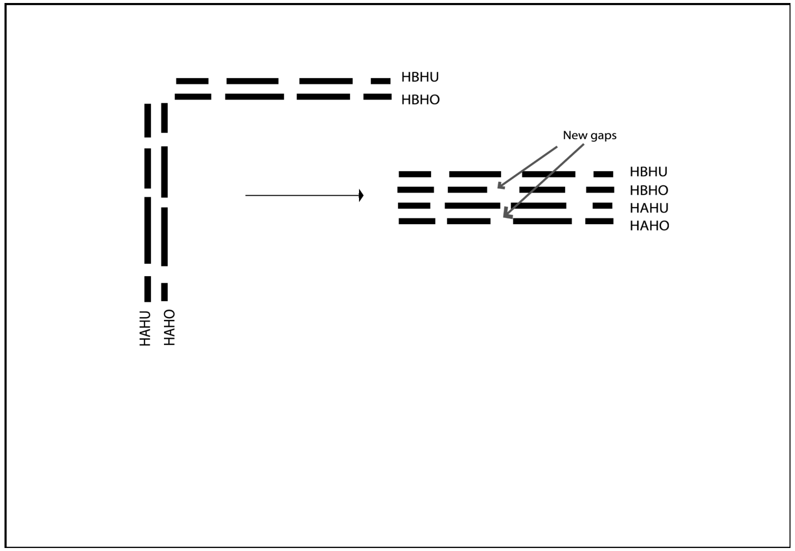
- Starting with a group of 7 sequences from different species
- Do pairwise alignments between all 7 sequences
- Score given for similarity, higher score indicates more similar

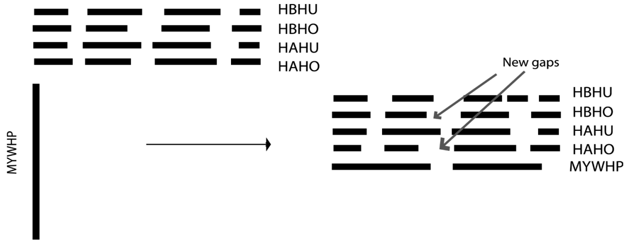
| | HAHU | HBHU | HAHO | HBHO | MYWHP | P1LHB | LGHB |
|---|---|---|---|---|---|---|---|
| **HAHU** | | | | | | | |
| **HBHU** | 21.1 | | | | | | |
| **HAHO** | 32.9 | 19.7 | | | | | |
| **HBHO** | 20.7 | **39.0** | 20.4 | | | | |
| **MYWHP** | 11.0 | 9.8 | 10.3 | 9.7 | | | |
| **P1LHB** | 9.3 | 8.6 | 9.6 | 8.4 | 7.0 | | |
| **LGHB** | 7.1 | 7.3 | 7.5 | 7.4 | 7.3 | 4.3 | |



- Cluster the sequences by similarity to create a guide tree
- Branch length is proportional to estimated divergence between the two sequences







## Sum of Pairs Score(SP-Score)

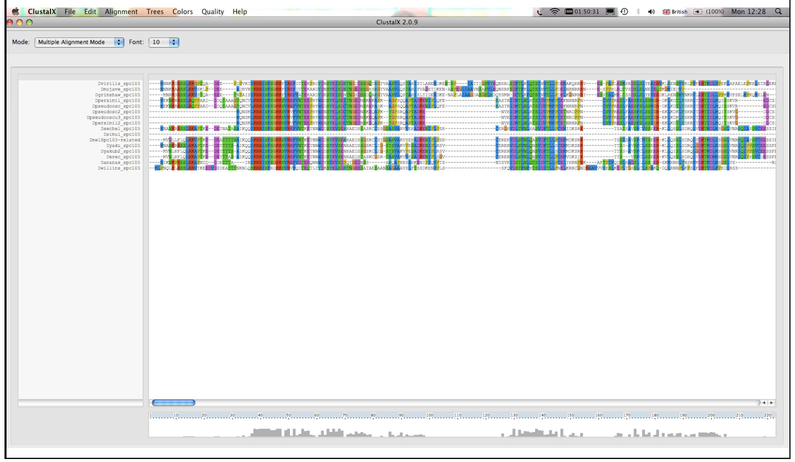
- Consider pairwise alignment of sequences $a_i$ and $a_j$ imposed by a multiple alignment of $k$ sequences
- Denote the score of this suboptimal (not necessarily optimal) pairwise alignment as $s^*(a_i, a_j)$
- Sum up the pairwise scores for a multiple alignment:

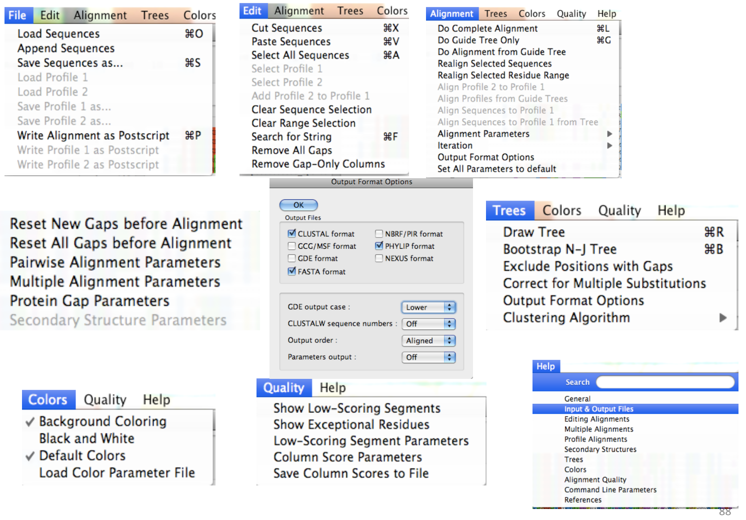
$$s(a_1,\ldots,a_k) = \Sigma_{i,j}\, s^*(a_i, a_j)$$

*Given a1,a2,a3,a4:*

$$\begin{aligned} s(a1\ldots a4) = \Sigma s^*(ai,aj) &= s^*(a1,a2) + s^*(a1,a3) \\ &+ s^*(a1,a4) + s^*(a2,a3) \\ &+ s^*(a2,a4) + s^*(a3,a4) \end{aligned}$$

CLUSTAL


ClustalX File Edit Alignment Trees Colors Quality Help
ClustalX 2.0.9
Mode: Multiple Alignment Mode
Font: 10


File Edit Alignment Trees Colors
Load Sequences ⌘O
Append Sequences
Save Sequences as... ⌘S
Load Profile 1
Load Profile 2
Save Profile 1 as...
Save Profile 2 as...
Write Alignment as Postscript ⌘P
Write Profile 1 as Postscript
Write Profile 2 as Postscript

Edit Alignment Trees Colors
Cut Sequences ⌘X
Paste Sequences ⌘V
Select All Sequences ⌘A
Select Profile 1
Select Profile 2
Add Profile 2 to Profile 1
Clear Sequence Selection
Clear Range Selection
Search for String ⌘F
Remove All Gaps
Remove Gap-Only Columns

Alignment Trees Colors Quality Help
Do Complete Alignment ⌘L
Do Guide Tree Only ⌘G
Do Alignment from Guide Tree
Realign Selected Sequences
Realign Selected Residue Range
Align Profile 2 to Profile 1
Align Profiles from Guide Trees
Align Sequences to Profile 1
Align Sequences to Profile 1 from Tree
Alignment Parameters ▶
Iteration ▶
Output Format Options
Set All Parameters to default

Reset New Gaps before Alignment
Reset All Gaps before Alignment
Pairwise Alignment Parameters
Multiple Alignment Parameters
Protein Gap Parameters
Secondary Structure Parameters

Output Format Options
OK
Output Files
☑ CLUSTAL format ☐ NBRF/PIR format
☐ GCG/MSF format ☑ PHYLIP format
☐ GDE format ☐ NEXUS format
☑ FASTA format
GDE output case : Lower
CLUSTALW sequence numbers : Off
Output order : Aligned
Parameters output : Off

Trees Colors Quality Help
Draw Tree ⌘R
Bootstrap N-J Tree ⌘B
Exclude Positions with Gaps
Correct for Multiple Substitutions
Output Format Options
Clustering Algorithm ▶

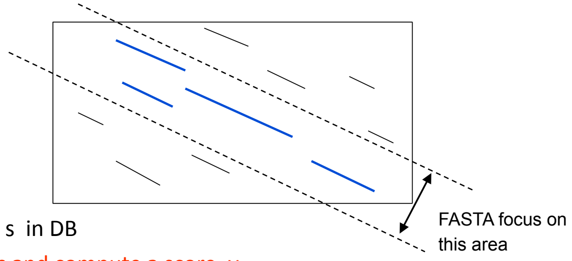
Colors Quality Help
✓ Background Coloring
Black and White
✓ Default Colors
Load Color Parameter File

Quality Help
Show Low-Scoring Segments
Show Exceptional Residues
Low-Scoring Segment Parameters
Column Score Parameters
Save Column Scores to File

Help
Search
General
Input & Output Files
Editing Alignments
Multiple Alignments
Profile Alignments
Secondary Structures
Trees
Colors
Alignment Quality
Command Line Parameters
References

## FASTA - Heuristic -

- Problem of Dynamic Programming: D.P. computes the score in a lot of useless areas for optimal sequence
- Heuristic": Good local alignment should have some exact match subsequence.


FASTA focus on this area

```
Hi level algorithm
Let q be a query
max ← 0
For each sequence, s in DB
    compare q with s and compute a score, y
  if max < y
        max ← y;
bestSequence ← s ;
Return bestSequence
```

## FASTA - Algorithm -

- Step 1

Find all hot-spots

// Hot spots are pairs of words of length k that exactly match

- Step 1 in detail

Use look-up Table

Query : GAATTCAGTTA

Sequence: GGATCGA

Look-up Table

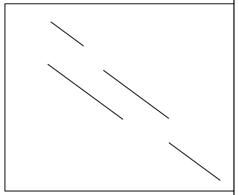
| Q | Location |
|---|---|
| A | 2,3,7,11 |
| C | 6 |
| G | 1,8 |
| T | 4,5,9,10 |

Dot—Matrix

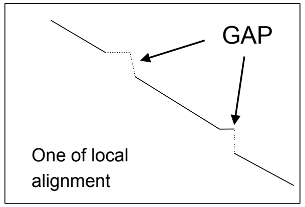
| | G | A | A | T | T | C | A | G | T | T | A |
|---|---|---|---|---|---|---|---|---|---|---|---|
| G | * | | | | | | | * | | | |
| G | * | | | | | | | * | | | |
| A | | * | * | | | | * | | | | * |
| T | | | | * | * | | | | * | * | |
| C | | | | | | * | | | | | |
| G | * | | | | | | | * | | | |
| A | | * | * | | | | * | | | | * |

## FASTA - Algorithm -

- Step 2: Score the Hot-spot and locate the ten best diagonal runs



- Step 3: Combine sub-alignments into one alignment with GAP



## FASTA - Algorithm

- Step 4

# Consider weighted direct graph.

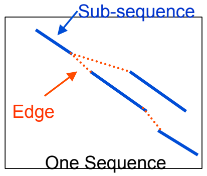
# Let node be a sub-alignment found in step 1

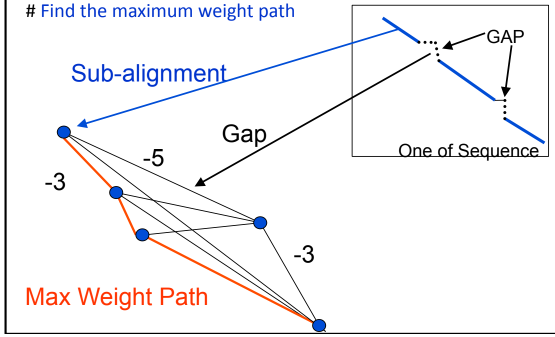
# Let u and v be nodes

# Edge (u,v) exists if alignment u is before in the sequence.

# Each edge has gap penalty (negative)

# Find the maximum weight path

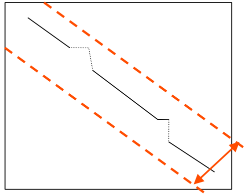




## FASTA - Algorithm -

- Step 5

Use the dynamic programming in restricted area around the best-score alignment to find out the higher-score alignment larger than the best-score alignment



Width of this band is a parameter

Summary of the algorithm

1: Find all hot-spots

// Hot spots is pairs of words of length k that exactly match.

2: Score each Hot-spot and locate the ten best diagonal runs.

3: Combine sub-alignments into one alignment.

4: Score each alignment with gap penalty and pick up the best-score alignment.

5: Use the dynamic programming in restricted area around the best-score alignment to find out the alignment greater than the best-score alignment.

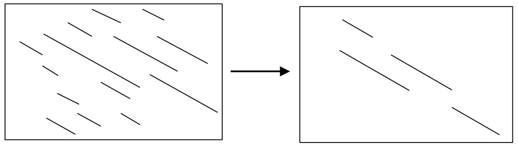
## FASTA - Complexity -

# Step 1 and 2 // select the best 10 diagonal runs

Let n be a sequence from DB

O(n) because Step 1 just uses look up table

O(n) << O(mn) m,n = 100 to 200



# Step 3 and 4 // compute the MAX Weight Path

Let r be the number of sub-alignments. (r = 10)

$O(r^2)$ < O(m*n)

# Step 5 // compute partial D.P.

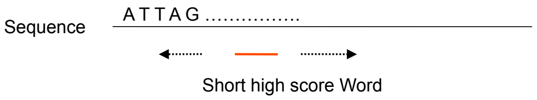
Depends on the restricted area < O(mn)

## BLAST
## Basic Local Alignment Search Tool

- Heuristic but evaluating the result statistically.

Homologous sequence are likely to contain a short high scoring word pair, a hit.

BLAST tries to extend it on the both sides to get optimal sequence.



hit

```
GCNTACACGTCACCATCTGTGCCACCACNCATGTCTCTAGTGATCCCTCATAAGTTCCAACAAAGTTTGC
|| |||||  | |||  ||||   ||    |||||||||||||||||| | |||||||| |  | |||||
GCCTACACACCGCCAGTTGTG-TTCCTGCTATGTCTCTAGTGATCCCTGAAAAGTTCCAGCGTATTTTGC

GAGTACTCAACACCAACATTGATGGGCAATGGAAAATAGCCTTCGCCATCACACCATTAAGGGTGA----
|| ||||||||| ||||||  | |||||   |||||||| ||| ||||||||  |  | | ||
GAATACTCAACAGCAACATCAACGGGCAGCAGAAAATAGGCTTTGCCATCACTGCCATTAAGGATGTGGG

------------------TGTTGAGGAAAGCAGACATTGACCTCACCGAGAGGGCAGGCGAGCTCAGGTA
                  ||||||||||||| ||| ||||||||||| || ||||||| || ||||   |
TTGACAGTACACTCATAGTGTTGAGGAAAGCTGACGTTGACCTCACCAAGTGGGCAGGAGAACTCACTGA

GGATGAGGTGGAGCATATGATCACCATCATACAGAACTCAC-------CAAGATTCCAGACTGGTTCTTG
||||||| |||| | | |||| ||||| || |||||  ||        |||||| |||||||||||||||
GGATGAGATGGAACGTGTGATGACCATTATGCAGAATCCATGCCAGTACAAGATCCCAGACTGGTTCTTG
```

Human-Mouse genome **homology**

# BLAST - Algorithm -

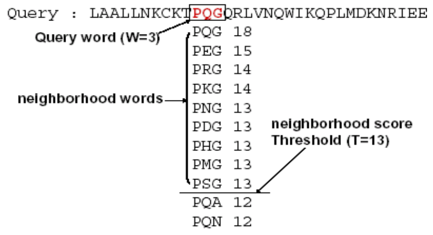
- Step 1: preprocessing Query

Compile the short-hit scoring word list from query.

The length of query word,w, is 3 for protein search, 11 for DNA.

Threshold T is 13

Neighborhood Word



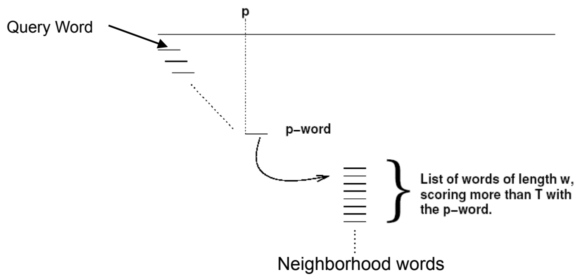
Query : LAALLNKCKTPQGQRLVNQWIKQPLMDKNRIEE

| | |
|---|---|
| PQG | 18 |
| PEG | 15 |
| PRG | 14 |
| PKG | 14 |
| PNG | 13 |
| PDG | 13 |
| PHG | 13 |
| PMG | 13 |
| PSG | 13 |
| PQA | 12 |
| PQN | 12 |

# BLAST - Algorithm -

- Step 1 – 2

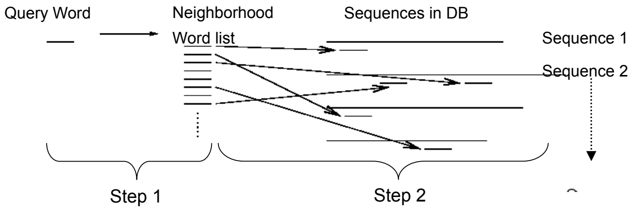
Create neighborhood words for each query word


Query Word
p
p–word
List of words of length w, scoring more than T with the p–word.
Neighborhood words

# BLAST - Algorithm -

- Step 2: Scanning DB

For each words list, identify all exact matches with DB sequences



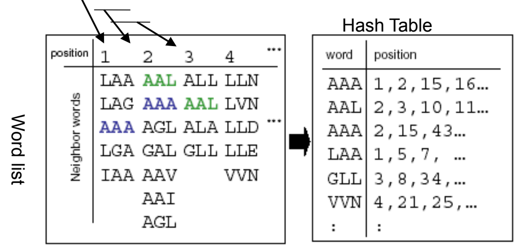
The purpose of Step 1 and 2 is as same as FASTA

# BLAST - Algorithm -

- Step 2-2

Method 1: Hash Table

Query: LAALLNKCKTPQGQRLVNQWIKQPLMD

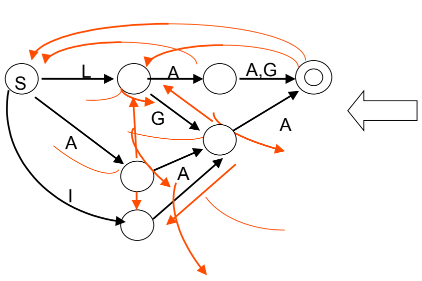


| position | 1 | 2 | 3 | 4 | … |
|---|---|---|---|---|---|
| | LAA | AAL | ALL | LLN | |
| | LAG | AAA | AAL | LVN | |
| | AAA | AGL | ALA | LLD | … |
| | LGA | GAL | GLL | LLE | |
| | IAA | AAV | | VVN | |
| | | AAI | | | |
| | | AGL | | | |

| word | position |
|---|---|
| AAA | 1,2,15,16… |
| AAL | 2,3,10,11… |
| AAA | 2,15,43… |
| LAA | 1,5,7, … |
| GLL | 3,8,34,… |
| VVN | 4,21,25,… |
| : | : |

# BLAST - Algorithm -

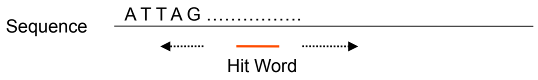
- Step 2-3
  Method 2: Finite Automata



| position | 1 | 2 | 3 | 4 | … |
|---|---|---|---|---|---|
| Neighbor words | LAA | AAL | ALL | LLN | |
| | LAG | AAA | AAL | LVN | |
| | AAA | AGL | ALA | LLD | … |
| | LGA | GAL | GLL | LLE | |
| | IAA | AAV | | VVN | |
| | | AAI | | | |
| | | AGL | | | |

# BLAST – Algorithm -

- Step 3 (Search optimal alignment)
  Let S be a score of hit-word
  For each hit-word, extend ungapped alignment in both directions.
- Step 4 (Evaluate the alignment statistically)
  Stop extension when E-value (depending on score S) become less than threshold. The hit-word is called High Scoring Segment Pair. BLAST return it



E-value = the number of HSPs having score S (or higher) expected to occur only by chance.

→ Smaller E-value, more significant in statistics
Bigger E-value , by chance

# BLAST - Algorithm -

- Step 3 -2
  Definition of E-Value

The expected number of HSP with the score at least S is :

$$E = K*n*m*e^{-\lambda S}$$

$K, \lambda$ is constant depending on model

$n, m$ are the length of query and sequence

The probability of finding at least one such HSP is:

$$P = 1 - e^{E}$$

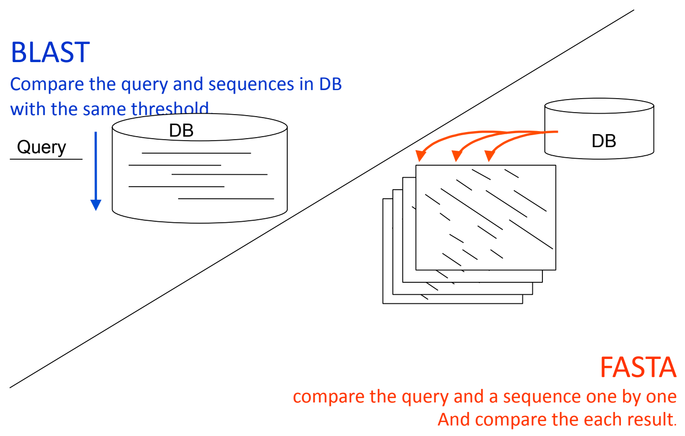
→ If a word is hit by chance (E-value is bigger),
P become smaller.

# BLAST - Running Time -

- Running Time on a Pentium 4
  The length of Query : 153
  DB size : 5997 sequences

| Algorithm | Running Time |
|---|---|
| D.P | 16.989 [s] |
| FASTA | 0.618 [s] |
| BLAST | 0.118 [s] |

# FASTA vs BLAST



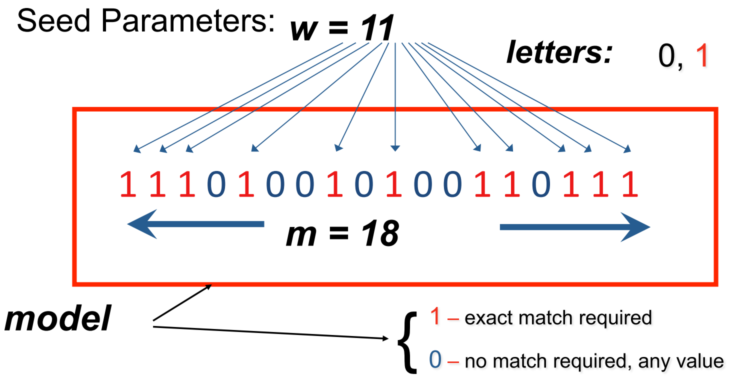
BLAST

Compare the query and sequences in DB with the same threshold

FASTA

compare the query and a sequence one by one
And compare the each result.

Hits indicate a similarity that may indicate a homology; Seeds determine how an algorithm looks for hits; w-> weight or number of positions to match; model -> relative position of letters for each w



*Patternhunter most sensitive model*

**Blastn seed is all "1"s**

A spaced seed is formed by two words, one from each input sequence, that match at positions specified by a fixed *pattern* – a word over symbols # and _ interpreted as a match and a don't care symbol respectively. For example, pattern ##_# specifies that the first, second and fourth positions must match and the third one may contain a mismatch.

PatternHunter was the first method that used carefully designed spaced seeds to improve the sensitivity of DNA local alignment. Spaced seeds have been shown to improve the efficiency of *lossless* filtration for approximate pattern matching, namely for the problem of detecting all matches of a string of length *m* with *q* possible substitution errors (an (*m*, *q*)-problem). Other software use some specific spaced seeds and random spaced seeds

107

# BLAST uses "consecutive seeds"

- In BLAST, we often use the consecutive model with weight 11.

```
GAGTACTCAACACCAACATCAGTGGGCAATGGAAAAT
|| ||||||||| |||||||| ||||||    ||||||
GAATACTCAACAGCAACATCAATGGGCAGCAGAAAAT
→11111111111 → … →… … → 11111111111←
```

- However, it fails to find the alignment in the two sequence.

# Dilemma: Sensitivity vs Speed

**Similarity**

How similar it is between two sequences?

Usually mean that the probability of the same symbol appear in anywhere of two sequences.

**Sensitivity**

The probability to find a local alignment.

needs shorter seeds

too many random hits, slow computation

**Speed** – needs longer seeds, lose distant homologies

**Specificity**

In all local alignments, how many alignments are homologous

# PatternHunter uses “non-consecutive seed”

- In PatternHunter, we often use the spaced model with weight 11 and length 18.

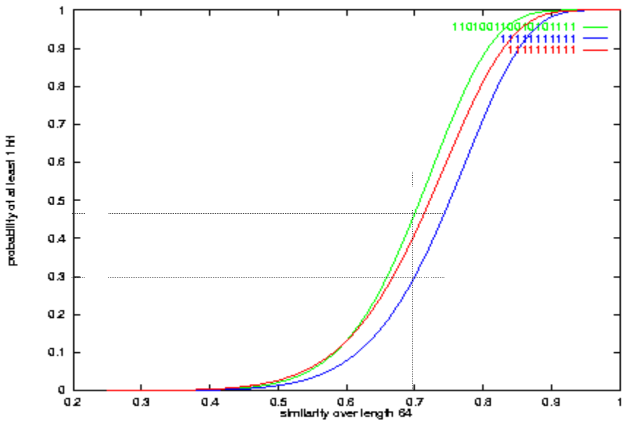
```
GAGTACTCAACACCAACATCAGTGGGCAATGGAAAAT
|| ||||||||| |||||||| ||||||   ||||||
GAATACTCAACAGCAACATCAATGGGCAGCAGAAAAT
        111010010100110111
```

- Higher hit probability
- Lower expected number of random hits

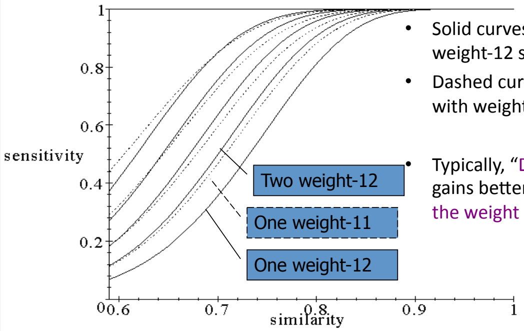
# A trivial comparison between spaced and consecutive seed

- Consider 111 and 1101.
- To fail seed 111, we can use
  - 110110110110…
  - 66.66% similarity
- But we can prove, seed 1101 will hit every region with 61% similarity for sufficient long region.

# Simulated sensitivity curves

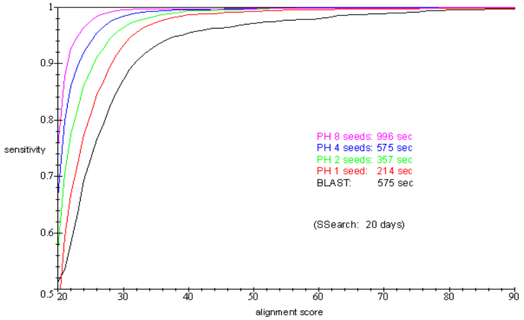


# Simulated sensitivity curves:



- Solid curves: Multiple (1, 2, 4, 8, 16) weight-12 spaced seeds.
- Dashed curves: Optimal spaced seeds with weight = 11, 10, 9, 8.
- Typically, “Doubling the seed number” gains better sensitivity than “decreasing the weight by 1”.

# Sensitivity curves:



# Proof

- Suppose there is a length 100 region which is not hit by 1101.
- We can break the region into blocks of $1^a0^b$. Besides the last block, the other blocks have the following few cases:
  - $10^b$ for b>=1
  - $110^b$ for b>=2
  - $1110^b$ for b>=2
- In each block, similarity <= 3/5.
- The last block has at most 3 matches.
- So, in total there are at most 61 matches in 100 positions. The similarity is <=61%.

# Formalize

- Given i.i.d. sequence (homology region) with Pr(1) =p and Pr(0)=1-p for each bit:

```
1100111011101101011101101011111011101
                   111*1**1*1**11*111
```

- Which seed is more likely to hit this region:
  - BLAST seed: 11111111111
  - Spaced seed: 111*1**1*1**11*111

## Expect Less, Get More

- Lemma: The expected number of hits of a weight W length M seed model within a length L region with homology level p is

  $(L-M+1)p^W$

  Proof. $E(\#hits) = \sum_{i=1 \dots L-M+1} p^W$ ■

- Example: In a region of length 64 with p=0.7
  - Pr(BLAST seed hits)=0.3

    E(# of hits by BLAST seed)=1.07
  - Pr(optimal spaced seed hits)=0.466, 50% more

    E(# of hits by spaced seed)=0.93, 14% less

## Why Is Spaced Seed Better?

A wrong, but intuitive, proof: seed s, interval I, similarity p

$E(\#hits) = Pr(s\ hits)\ E(\#hits \mid s\ hits)$

Thus:

$Pr(s\ hits) = Lp^w / E(\#hits \mid s\ hits)$

For optimized spaced seed, E(#hits | s hits)

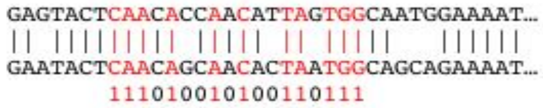
| 111*1**1*1**11*111 | Non overlap | Prob |
|---|---|---|
| 111*1**1*1**11*111 | 6 | $p^6$ |
| 111*1**1*1**11*111 | 6 | $p^6$ |
| 111*1**1*1**11*111 | 6 | $p^6$ |
| 111*1**1*1**11*111 | 7 | $p^7$ |
| ….. | | |

- For spaced seed: the divisor is $1+p^6+p^6+p^6+p^7+ \dots$
- For BLAST seed: the divisor is bigger: $1+ p + p^2 + p^3 + \dots$

## Observations of spaced seeds

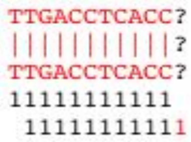
- Seed models with different shapes can detect different homologies.
- Two consequences:
  - Some models *may* detect more homologies than others
    - More sensitive homology search
    - PatternHunter I
  - Can use several seed models simultaneously to hit more homologies
    - Approaching 100% sensitive homology search
    - PatternHunter II

Example of a hit using a spaced seed:



```
GAGTACTCAACACCAACATTAGTGGCAATGGAAAAT…
|| ||||||||| ||||| || |||||   ||||||
GAATACTCAACAGCAACACTAATGGCAGCAGAAAAT…
       111010010100110111
```

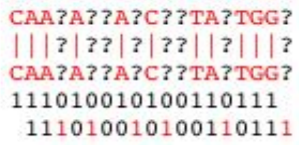
- BLAST: redundant hits



```
TTGACCTCACC?
|||||||||||?
TTGACCTCACC?
11111111111
 11111111111
```

This results in > 1 hit and creates clusters of redundant hits

- PatternHunter



```
CAA?A??A?C??TA?TGG?
|||?|??|?|??||?|||?
CAA?A??A?C??TA?TGG?
111010010100110111
 111010010100110111
```

This results in very few redundant hits

120

# Why is PH better?

**BLAST may also miss a hit**

```
GAGTACTCAACACCAACATTAGTGGGCAATGGAAAAT
|| ||||||||| |||||| | ||||||   ||||||
GAATACTCAACAGCAACATCAATGGGCAGCAGAAAAT
```

9 matches

In this example, despite a clear homology, there is no sequence of continuous matches longer than length 9. BLAST uses a length 11 and because of this, BLAST does not recognize this as a hit!

Resolving this would require reducing the seed length to 9, which would have a damaging effect on speed

# PatternHunter II:

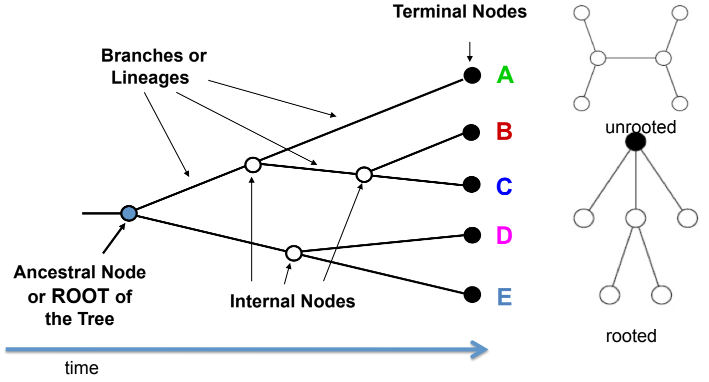
-- Smith-Waterman Sensitivity, BLAST Speed

(Li, Ma, Kisman, Tromp, *J. Bioinfo Comput. Biol.* 2004)

- The biggest problem for BLAST was low sensitivity (and low speed). Massive parallel machines are built to do S-W exhaustive dynamic programming.
- Spaced seeds give PH a *unique* opportunity of using several optimal seeds to achieve optimal sensitivity, this was not possible by BLAST technology.
- PH II has with multiple optimal seeds.
- PH II approaches Smith-Waterman sensitivity, and 3000 times faster.

# BioInformatics 5

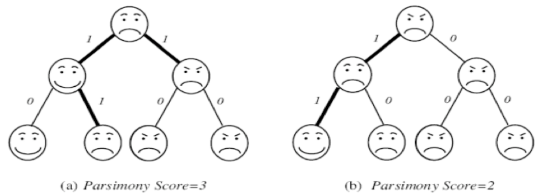
## Molecular Evolution: Fitch and Sankoff Algorithms



((A,(B,C)),(D,E)) = The above phylogeny as nested parentheses

# Parsimony Approach

- Applies Occam's razor principle to identify the simplest explanation for the data
- Assumes observed character differences resulted from the fewest possible mutations
- Seeks the tree that yields lowest possible **parsimony score -** sum of cost of all mutations found in the tree



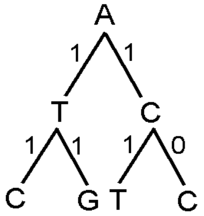
## Small Parsimony

- Input: Tree *T* with each leaf labeled by an *m*-character string.
- Output: Labeling of internal vertices of the tree *T* minimizing the parsimony score.
- We can assume that every leaf is labeled by a single character, because the characters in the string are independent.

## Weighted Small Parsimony Problem

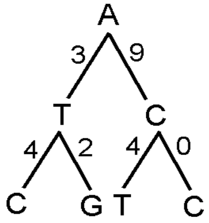
- Input: Tree *T* with each leaf labeled by elements of a *k*-letter alphabet and a *k* x *k* scoring matrix ($\delta_{ij}$)
- Output: Labeling of internal vertices of the tree *T* minimizing the weighted parsimony score.
- For Small Parsimony problem, the scoring matrix is based on Hamming distance dH(v, w) = 0 if v=w ; dH(v, w) = 1 otherwise

## Unweighted vs. Weighted



| | A | T | G | C |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| T | 1 | 0 | 1 | 1 |
| G | 1 | 1 | 0 | 1 |
| C | 1 | 1 | 1 | 0 |

Small Parsimony Score: 5



| | A | T | G | C |
|---|---|---|---|---|
| A | 0 | 3 | 4 | 9 |
| T | 3 | 0 | 2 | 4 |
| G | 4 | 2 | 0 | 4 |
| C | 9 | 4 | 4 | 0 |

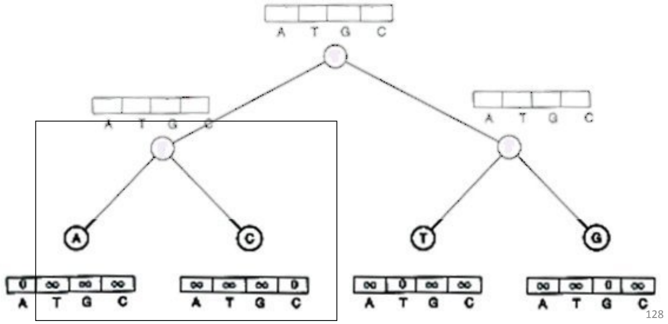
Weighted Parsimony Score: 22

## Sankoff Algorithm: Dynamic Programming

- Calculate and keep track of a score for every possible label at each vertex
  - $s_t(v)$ = minimum parsimony score of the **subtree** rooted at vertex *v* if *v* has character *t*
- The score at each vertex is based on scores of its children:
  - $s_t(\textbf{\textit{parent}}) = \min_i \{s_i(\textbf{\textit{ left child }}) + \delta_{i,\,t}\} + \min_j \{s_j(\textbf{\textit{ right child }}) + \delta_{j,\,t}\}$

## Sankoff Algorithm (cont.)

- Begin at leaves:
  - If leaf has the character in question, score is 0
  - Else, score is ∞

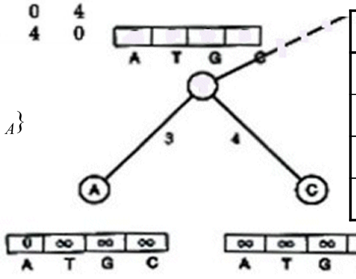


## Sankoff Algorithm (cont.)

| δ | A | T | G | C |
|---|---|---|---|---|
| A | 0 | 3 | 4 | 9 |
| T | 3 | 0 | 2 | 4 |
| G | 4 | 2 | 0 | 4 |
| C | 9 | 4 | 4 | 0 |

$s_t(v) = \min_i \{s_i(u) + \delta_{i,\,t}\} + \min_j\{s_j(w) + \delta_{j,\,t}\}$

$s_A(v) = 0$
$\min_j\{s_j(w) + \delta_{j,\,A}\}$



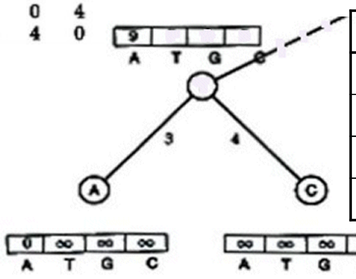
| | $s_i(u)$ | $\delta_{i,\,A}$ | sum |
|---|---|---|---|
| A | 0 | 0 | **0** |
| T | ∞ | 3 | ∞ |
| G | ∞ | 4 | ∞ |
| C | ∞ | 9 | ∞ |

## Sankoff Algorithm (cont.)

| δ | A | T | G | C |
|---|---|---|---|---|
| A | 0 | 3 | 4 | 9 |
| T | 3 | 0 | 2 | 4 |
| G | 4 | 2 | 0 | 4 |
| C | 9 | 4 | 4 | 0 |

$s_t(v) = \min_i \{s_i(u) + \delta_{i,\,t}\} + \min_j\{s_j(w) + \delta_{j,\,t}\}$

$s_A(v) = 0$
$+ 9 = 9$



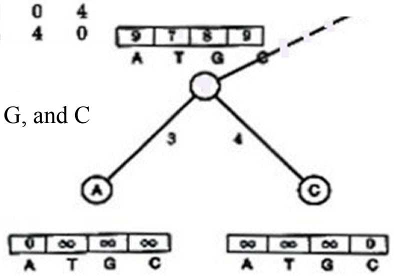
| | $s_j(u)$ | $\delta_{j,\,A}$ | sum |
|---|---|---|---|
| A | ∞ | 0 | ∞ |
| T | ∞ | 3 | ∞ |
| G | ∞ | 4 | ∞ |
| C | 0 | 9 | **9** |

## Sankoff Algorithm (cont.)

| δ | A | T | G | C |
|---|---|---|---|---|
| A | 0 | 3 | 4 | 9 |
| T | 3 | 0 | 2 | 4 |
| G | 4 | 2 | 0 | 4 |
| C | 9 | 4 | 4 | 0 |

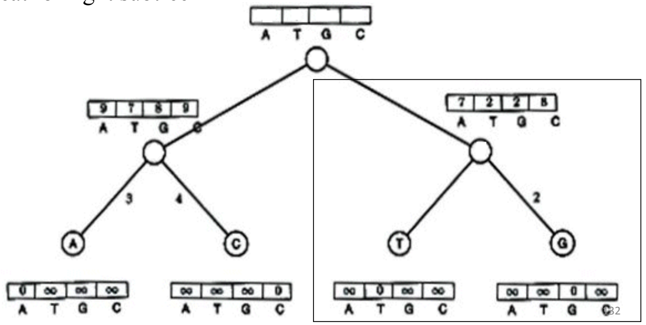
$s_t(v) = \min_i \{s_i(u) + \delta_{i,\,t}\} + \min_j\{s_j(w) + \delta_{j,\,t}\}$

Repeat for T, G, and C



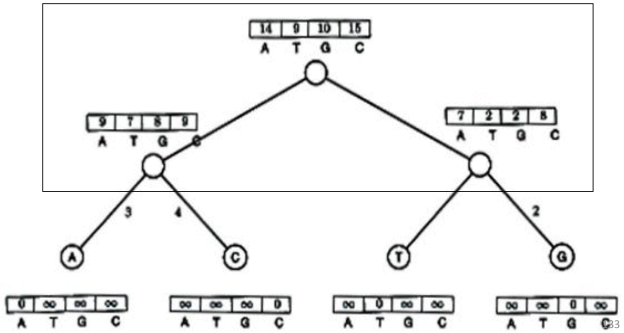
## Sankoff Algorithm (cont.)

Repeat for right subtree



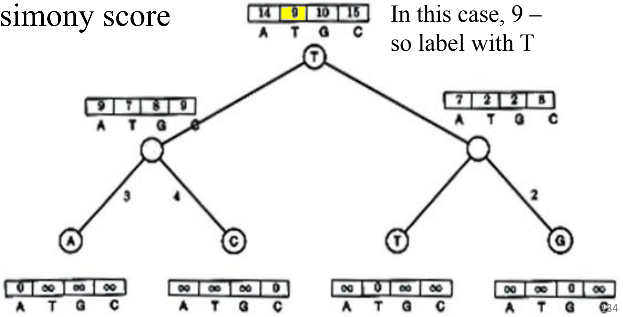
## Sankoff Algorithm (cont.)

Repeat for root



## Sankoff Algorithm (cont.)

Smallest score at root is minimum weighted parsimony score



In this case, 9 – so label with T

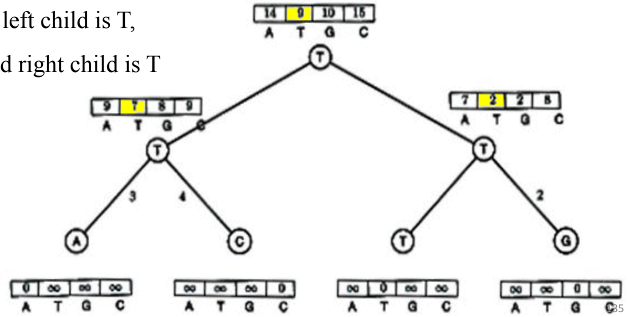
## Sankoff Algorithm: Traveling down the Tree

The scores at the root vertex have been computed by going up the tree
After the scores at root vertex are computed the Sankoff algorithm moves down the tree and assign each vertex with optimal character.

9 is derived from 7 + 2

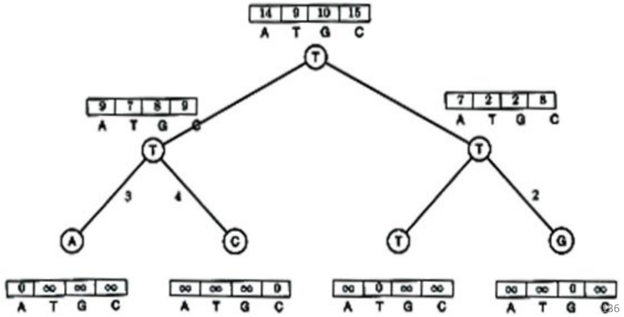
So left child is T,

And right child is T



## Sankoff Algorithm (cont.)

And the tree is thus labeled…



# Fitch Algorithm

- Solves Small Parsimony problem;
- Dynamic programming in essence;

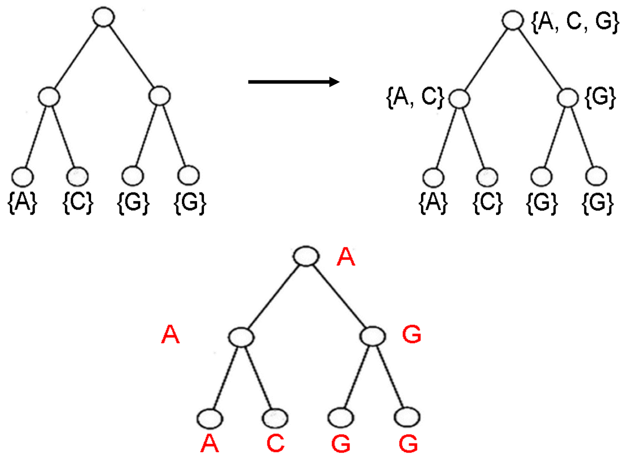
*1)* Assign a **set of possible letters** to every vertex, traversing the tree from leaves to root

- Each node's set is the combination of its children's sets (leaves contain their label)
  - E.g. if the node we are looking at has a left child labeled {A, C} and a right child labeled {A, T}, the node will be given the set {A}

*2)* Assign **labels** to each vertex, traversing the tree from root to leaves

- Assign root arbitrarily from its set of letters
- For all other vertices, if its parent's label is in its set of letters, assign it its parent's label
- Else, choose an arbitrary letter from its set as its label

# Fitch



# Parsimony Example

Say we have an alignment of 4 DNA sequences of 3 bases each

1 G G A

2 G G G

3 A C A

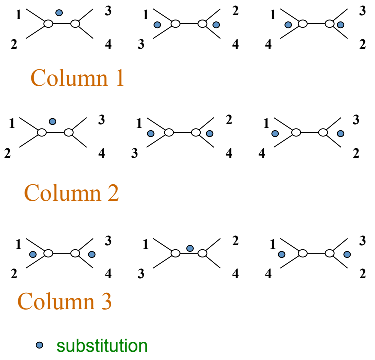
4 A C G

**Total number of substitutions**

**Tree 1: 4**

**Tree 2: 5**

**Tree 3: 6**



Column 1

Column 2

Column 3

# Sankoff vs. Fitch

- The Sankoff algorithm gives the **same** set of **optimal** labels as the Fitch algorithm
- For Sankoff algorithm, character *t* is *optimal* for vertex *v* if $s_t(v) = \min_{1\underline{<}i\underline{<}k} s_i(v)$
  - Denote the set of optimal letters at vertex *v* as *S*(*v*)
    - If *S*(*left child*) and *S*(*right child*) overlap, *S*(*parent*) is the intersection
    - Else it's the union of *S*(*left child*) and *S*(*right child*)
    - This is also the Fitch recurrence
- Complexity:
- Fitch: O(mnk); Sankoff: O(mnk$^2$)
- m characters, n leaves, k possible values for a character

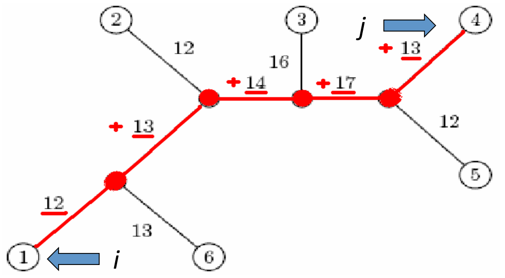
## Large Parsimony Problem

- Input: An $n$ x $m$ matrix $M$ describing $n$ species, each represented by an $m$-character string
- Output: A tree $T$ with $n$ leaves labeled by the $n$ rows of matrix $M$, and a labeling of the internal vertices such that the parsimony score is minimized over all possible trees and all possible labelings of internal vertices
- Possible search space is huge, especially as n increases
- (2n – 3)!! possible rooted trees
- (2n – 5)!! possible unrooted trees
- Problem is NP-complete; Exhaustive search only possible w/ small n(< 10)

**BioInformatics 6**

## Distance in Trees

*$d_{ij}(T)$ – **tree distance between i and j***



$d_{1,4} = 12 + 13 + 14 + 17 + 13 = 69$

## Edit Distance vs. Tree Distance

- Given $n$ sequences, we can compute the $n$ x $n$ ***distance matrix*** $D_{ij}$
- $D_{ij}$ may be defined as the edit distance between a gene in species $i$ and species $j$, where the gene of interest is sequenced for all $n$ species.

  $D_{ij}$ – ***edit distance between i and j***
- Note the difference with

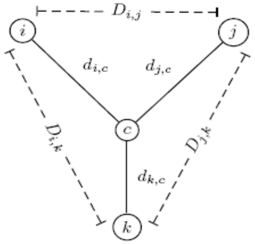
  $d_{ij}(T)$ – ***tree distance between i and j***

## Fitting Distance Matrix

- Given $n$ sequences, we can compute the $n$ x $n$ ***distance matrix*** $D_{ij}$
- Evolution of these sequences is described by a tree that ***we don't know***.
- We need an algorithm to construct a tree that best ***fits*** the distance matrix $D_{ij}$
- Fitting means $\underbrace{D_{ij}}_{\text{Edit distance between species (}\textit{known}\text{)}} = \overbrace{d_{ij}(T)}^{\text{Lengths of path in an (}\textit{unknown}\text{) tree } T}$

## Reconstructing a 3 Leaved Tree

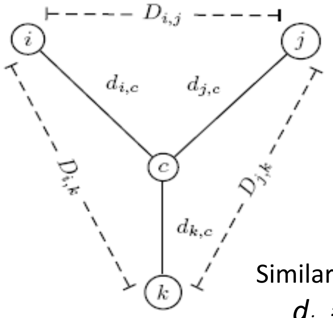
- Tree reconstruction for any 3x3 matrix is straightforward
- We have 3 leaves *i, j, k* and a center vertex *c*



Observe:

$$d_{ic} + d_{jc} = D_{ij}$$
$$d_{ic} + d_{kc} = D_{ik}$$
$$d_{jc} + d_{kc} = D_{jk}$$

## Reconstructing a 3 Leaved Tree (cont'd)



$$d_{ic} + d_{jc} = D_{ij}$$
$$+ \quad \underline{d_{ic} + d_{kc} = D_{ik}}$$
$$2d_{ic} + \underbrace{d_{jc} + d_{kc}} = D_{ij} + D_{ik}$$
$$2d_{ic} + \quad D_{jk} \quad = D_{ij} + D_{ik}$$
$$d_{ic} = (D_{ij} + D_{ik} - D_{jk})/2$$

Similarly,

$$d_{jc} = (D_{ij} + D_{jk} - D_{ik})/2$$
$$d_{kc} = (D_{ki} + D_{kj} - D_{ij})/2$$

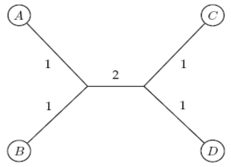
## Trees with > 3 Leaves

- A tree with *n* leaves has *2n-3* edges
- This means fitting a given tree to a distance matrix *D* requires solving a system of "n choose 2" equations with *2n-3* variables
- This is not always possible to solve for *n* > 3

## Additive Distance Matrices

Matrix *D* is ADDITIVE if there exists a tree *T* with $d_{ij}(T) = D_{ij}$

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 2 | 4 | 4 |
| B | 2 | 0 | 4 | 4 |
| C | 4 | 4 | 0 | 2 |
| D | 4 | 4 | 2 | 0 |



NON-ADDITIVE otherwise

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 2 | 2 | 2 |
| B | 2 | 0 | 3 | 2 |
| C | 2 | 3 | 0 | 2 |
| D | 2 | 2 | 2 | 0 |

?

## Distance Based Phylogeny Problem

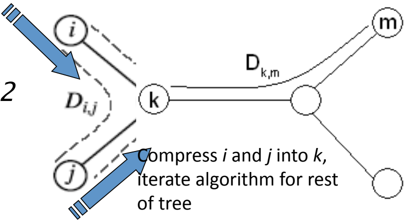
- Goal: Reconstruct an evolutionary tree from a distance matrix
- Input: $n$ x $n$ distance matrix $D_{ij}$
- Output: weighted tree $T$ with $n$ leaves fitting $D$

- If $D$ is additive, this problem has a solution and there is a simple algorithm to solve it

## Using Neighboring Leaves to Construct the Tree

- Find ***neighboring leaves*** *i* and *j* with parent *k*
- Remove the rows and columns of *i* and j
- Add a new row and column corresponding to *k*, where the distance from *k* to any other leaf *m* can be computed as:

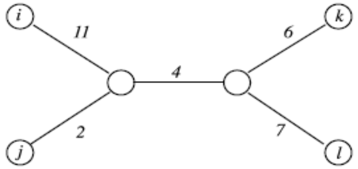
$$D_{km} = (D_{im} + D_{jm} - D_{ij})/2$$



Compress *i* and *j* into *k*, iterate algorithm for rest of tree

## Finding Neighboring Leaves

- Closest leaves aren't necessarily neighbors
- *i* and *j* are neighbors, but ($d_{ij}$ = 13) > ($d_{jk}$ = 12)



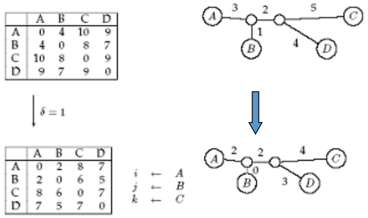
- Finding a pair of neighboring leaves is a nontrivial problem!

## Degenerate Triples

- A degenerate triple is a set of three distinct elements *1<=i,j,k<=n* where $D_{ij} + D_{jk} = D_{ik}$
- Element *j* in a degenerate triple *i,j,k* lies on the path from *i* to *k* (or is attached to this path by an edge of length 0).
- If distance matrix D has a degenerate triple i,j,k then j can be "removed" from D thus reducing the size of the problem.
- If distance matrix D does not have a degenerate triple i,j,k, one can "create" a degenerative triple in D by shortening all hanging edges (in the tree).

## Shortening Hanging Edges to Produce Degenerate Triples

- Shorten all “hanging” edges (edges that connect leaves) until a degenerate triple is found



|   | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 4 | 10 | 9 |
| B | 4 | 0 | 8 | 7 |
| C | 10 | 8 | 0 | 9 |
| D | 9 | 7 | 9 | 0 |

$\delta = 1$

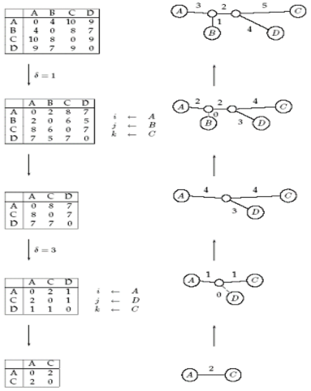
|   | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 2 | 8 | 7 |
| B | 2 | 0 | 6 | 5 |
| C | 8 | 6 | 0 | 7 |
| D | 7 | 5 | 7 | 0 |

$i \leftarrow A$
$j \leftarrow B$
$k \leftarrow C$

## Finding Degenerate Triples

- If there is no degenerate triple, all hanging edges are reduced by the same amount δ, so that all pair-wise distances in the matrix are reduced by 2δ.
- Eventually this process collapses one of the leaves (when δ = length of shortest hanging edge), forming a degenerate triple *i,j,k* and reducing the size of the distance matrix *D.*
- The attachment point for *j* can be recovered in the reverse transformations by saving $D_{ij}$ for each collapsed leaf.

## Reconstructing Trees for Additive Distance Matrices



|   | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 4 | 10 | 9 |
| B | 4 | 0 | 8 | 7 |
| C | 10 | 8 | 0 | 9 |
| D | 9 | 7 | 9 | 0 |

$\delta = 1$

|   | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 2 | 8 | 7 |
| B | 2 | 0 | 6 | 5 |
| C | 8 | 6 | 0 | 7 |
| D | 7 | 5 | 7 | 0 |

$i \leftarrow A$
$j \leftarrow B$
$k \leftarrow C$

|   | A | C | D |
|---|---|---|---|
| A | 0 | 8 | 7 |
| C | 8 | 0 | 7 |
| D | 7 | 7 | 0 |

$\delta = 3$

|   | A | C | D |
|---|---|---|---|
| A | 0 | 2 | 1 |
| C | 2 | 0 | 1 |
| D | 1 | 1 | 0 |

$i \leftarrow A$
$j \leftarrow D$
$k \leftarrow C$

|   | A | C |
|---|---|---|
| A | 0 | 2 |
| C | 2 | 0 |

## AdditivePhylogeny Algorithm

```
AdditivePhylogeny($D$)
  if $D$ is a $2$ x $2$ matrix
    $T$ = tree of a single edge of length $D_{1,2}$
    return $T$
  if $D$ is non-degenerate
    $\delta$ = trimming parameter of matrix $D$
    for all $1 \le i \ne j \le n$
      $D_{ij} = D_{ij} - 2\delta$
  else
    $\delta = 0$
```

## AdditivePhylogeny (cont'd)

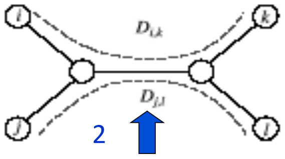
```
Find a triple $i, j, k$ in $D$ such that $D_{ij} + D_{jk} = D_{ik}$
$x = D_{ij}$
Remove $j^{th}$ row and $j^{th}$ column from $D$
$T$ = AdditivePhylogeny($D$)
Add a new vertex $v$ to $T$ at distance $x$ from $i$ to $k$
Add $j$ back to $T$ by creating an edge $(v,j)$ of length 0
for every leaf $l$ in $T$
    if distance from $l$ to $v$ in the tree ≠ $D_{l,j}$
        output "matrix is not additive"
        return
Extend all "hanging" edges by length $\delta$
return $T$
```

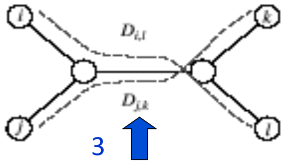
## The Four Point Condition

- AdditivePhylogeny provides a way to check if distance matrix *D* is additive
- **An even more efficient additivity check is the "four-point condition"**
- Let $1 \le i,j,k,l \le n$ be four distinct leaves in a tree

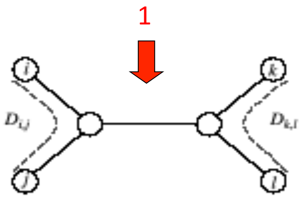
## The Four Point Condition (cont'd)

Compute: 1. $D_{ij} + D_{kl}$, 2. $D_{ik} + D_{jl}$, 3. $D_{il} + D_{jk}$





2 and 3 represent the same number: the length of all edges + the middle edge (it is counted twice)



1 represents a smaller number: the length of all edges – the middle edge

## The Four Point Condition: Theorem

- The four point condition for the quartet *i,j,k,l* is satisfied if two of these sums are the same, with the third sum smaller than these first two
- ***Theorem*** : An *n* x *n* matrix *D* is additive if and only if the four point condition holds for ***every*** quartet $1 \le i,j,k,l \le n$

## Least Squares Distance Phylogeny Problem

- If the distance matrix *D* is NOT additive, then we look for a tree *T* that approximates *D* the best:

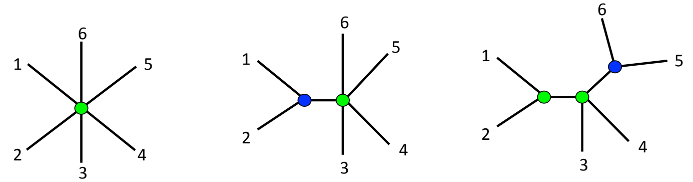
***Squared Error*** : $\sum_{i,j} (d_{ij}(T) - D_{ij})^2$

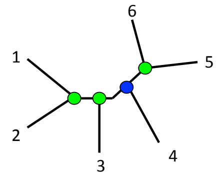
- Squared Error is a measure of the quality of the fit between distance matrix and the tree: we want to minimize it.
- **Least Squares Distance Phylogeny Problem**: finding the best approximation tree *T* for a non-additive matrix *D* (NP-hard).

## Neighbor Joining Algorithm

- In 1987 Naruya Saitou and Masatoshi Nei developed a neighbor joining algorithm for phylogenetic tree reconstruction
- **Finds a pair of leaves that are close to each other but far from other leaves:** implicitly finds a pair of neighboring leaves
- Advantages: works well for additive and other non-additive matrices, it does not have the flawed molecular clock assumption (see UPGMA).

## Neighbor Joining Algorithm





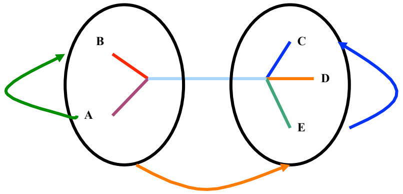
- Sequences chosen to give best least-squares estimate of branch length
- Begin with star topology – no neighbors have been joined

## Neighbor Joining

- Tree modified by joining pairs of sequences
- Pair is chosen by calculating sum of branch lengths, S, for the corresponding tree (joining m and n; i are the other nodes); $d_{ij}$ are the distance matrix values.

$$S_{mn} = \frac{\sum d_{im} + d_{in}}{2(N-2)} + \frac{d_{mn}}{2} + \frac{\sum d_{ij}}{N-2}$$

(If A and B are joined):



## Neighbour Joining Algorithm

- Identify i,j as neighbours if their “distance” is the shortest.
- Combine i,j into a new node u.
- Update the distance matrix.
- Distance of *u* from the rest of the tree is calculated
- If only 3 nodes are left – finish.

**Why does using $S_{ij}$ give us O($n^5$) complexity?**

1. If N represents the number of leaves at each stage, we compute $S_{12}$, $S_{13}$, $S_{14}$, …$S_{23}$, …$S_{(N-1,N),}$ which about $N^2$ computations.
2. We have N stages (we start off with a matrix of N x N, and at each stage the matrix is reduced by 1) → so we've reached N x $N^2 = N^3$.
3. Each $S_{ij}$ we compute, requires us to sum over all of the elements in the matrix – once again, $N^2$ computations, so now we've reached N x $N^2$ X $N^2 = N^5$.

Let’s define a new parameter called r. This r is computed for each node represented in the current matrix.

$$r_i = \sum_{k=1}^{N} d_{ik}$$

(i represents the node for which we are computing r now)

Next, we define a rate corrected matrix (M), in which the elements are defined by:

$M_{ij} = d_{ij} – (r_i+r_j) / (N-2)$

And this is now our new parameter; i and j represent numbers of nodes. At each stage, we look for the i and j which give us the minimal $M_{ij}$.

**Why does $M_{ij}$ give us complexity of O($N^3$)?**

In $M_{ij}$ we only have to evaluate $r_i$ and $r_j$ each round. This can be achieved in O(1), if we compute these terms *once* at the beginning of the round.

Thus, if we return to the list that built the complexity of $S_{ij}$,

Stage 1 and 2 remain with the same complexity → O($N^3$).

Stage 3 is reduced to O(1), and thus we get a total of O($N^3$).

## UPGMA: Unweighted Pair Group Method with Arithmetic Mean

- UPGMA is a clustering algorithm that:
  - computes the distance between clusters using average pairwise distance
  - assigns a *height* to every vertex in the tree, effectively assuming the presence of a molecular clock and dating every vertex
  - The algorithm produces an ultrametric tree : the distance from the root to any leaf is the same (this corresponds to a constant molecular clock: leaves in the tree are assumed to accumulate mutations (and thus evolve) at the same rate.

## Clustering in UPGMA

Given two disjoint clusters $C_i$, $C_j$ of sequences,

$$d_{ij} = \frac{1}{|C_i| \times |C_j|} \Sigma_{\{p \in Ci,\, q \in Cj\}} d_{pq}$$

Note that if $C_k = C_i \cup C_j$, then distance to another cluster $C_l$ is:

$$d_{kl} = \frac{d_{il}\, |C_i| + d_{jl}\, |C_j|}{|C_i| + |C_j|}$$

## UPGMA Algorithm

**Initialization:**

Assign each $x_i$ to its own cluster $C_i$

Define one leaf per sequence, each at height 0

**Iteration:**

Find two clusters $C_i$ *and* $C_j$ such that $d_{ij}$ is min

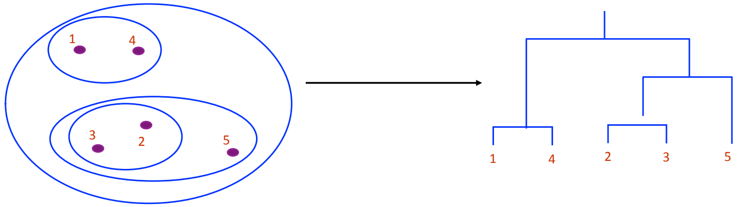
Let $C_k = C_i \cup C_j$

Add a vertex connecting $C_i$, $C_j$ *and* place it at height $d_{ij}/2$

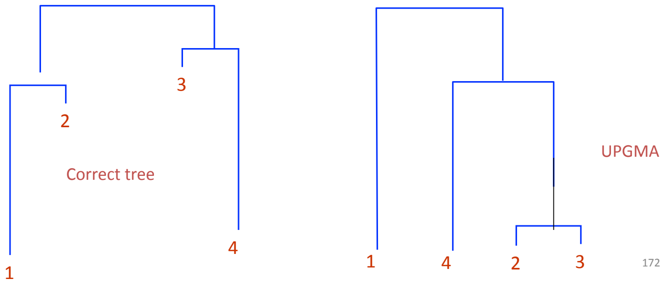
Delete $C_i$ *and* $C_j$

**Termination:**

When a single cluster remains

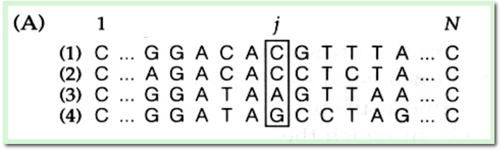


Weakness

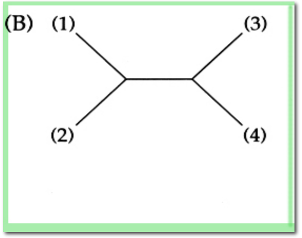


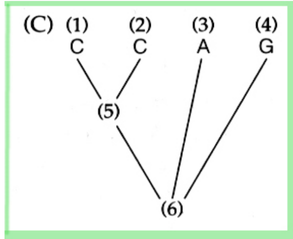
## BioInformatics 7: Likelihood for a tree

Aligned sequences for 4 taxa; What is the prob that this tree generated the data?



```
(A)      1                      j                    N
    (1)  C ... G G A C A  C  G T T T A ... C
    (2)  C ... A G A C A  C  C T C T A ... C
    (3)  C ... G G A T A  A  G T T A A ... C
    (4)  C ... G G A T A  G  C C T A G ... C
```



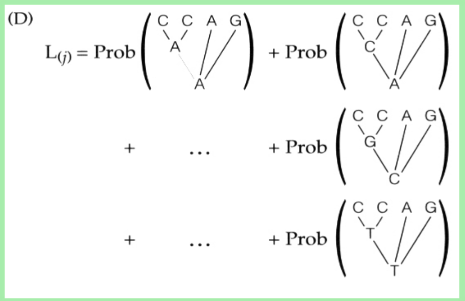


## Calculating L for a tree

- Root the tree at any internal node (models are time-reversible)
- Assumption of independence allows to calculate L for each site separately
- Then combine the likelihoods into a total value at the end
- To calculate L for some site j, we must consider all possible scenarios by which the tip sequences could have evolved; Specifically, the root (6) may have had A, C, T, or G.
- For each of these possibilities, the other internal node (5) also might have possessed any of the 4 nucleotides

## Calculating L for a tree

- Thus, there are 4x4=16 possibilities to consider



## Calculating L for a tree

- Calculate the probability of each and sum them to obtain the total probability for site j
- Assume that the changes along each branch are independent (Markov model)
- Thus, the Pr of any single scenario is equal to the product of the Pr of the changes required by that scenario

(E)

$$L = L_{(1)} \bullet L_{(2)} \bullet \ldots \bullet L_{(N)} = \prod_{j=1}^{N} L_{(j)}$$

# Calculating L for a tree

- Because the Probability of any single observation is an extremely small number, we evaluate the log of the likelihood instead
- Probabilities are accumulated as the sum of logs of the single-site likelihoods

$$\text{(F)}\quad \ln L = \ln L_{(1)} + \ln L_{(2)} + \ldots + \ln L_{(N)} = \sum_{j=1}^{N} \ln L_{(j)}$$

Typical assumptions of ML substitution models

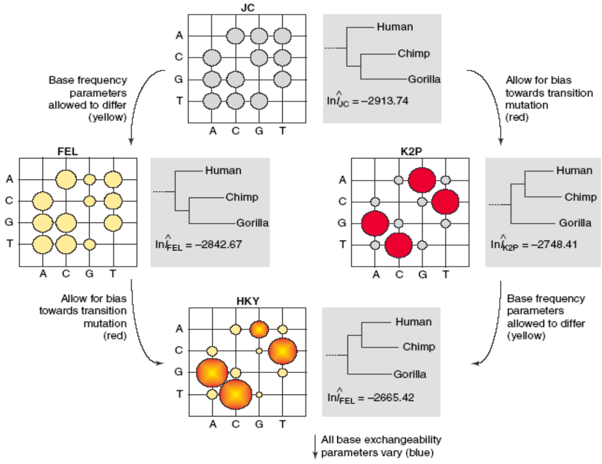
The probability of any change is independent of the prior history of the site (a Markov Model)

Substitution probabilities do not change with time or over the tree (a homogeneous Markov process)

Change is time reversible e.g. the rate of change of A to T is the same as T to A

## Typical assumptions of ML substitution models

- The probability of any change is independent of the prior history of the site (a Markov Model)
- Substitution probabilities do not change with time or over the tree (a homogeneous Markov process)
- Change is time reversible e.g. the rate of change of A to T is the same as T to A


JC
A
C
G
T
A C G T
Human
Chimp
Gorilla
$\ln\hat{L}_{JC} = -2913.74$
Base frequency parameters allowed to differ (yellow)
Allow for bias towards transition mutation (red)
FEL
$\ln\hat{L}_{FEL} = -2842.67$
K2P
$\ln\hat{L}_{K2P} = -2748.41$
Allow for bias towards transition mutation (red)
Base frequency parameters allowed to differ (yellow)
HKY
$\ln\hat{L}_{FEL} = -2665.42$
All base exchangeability parameters vary (blue)

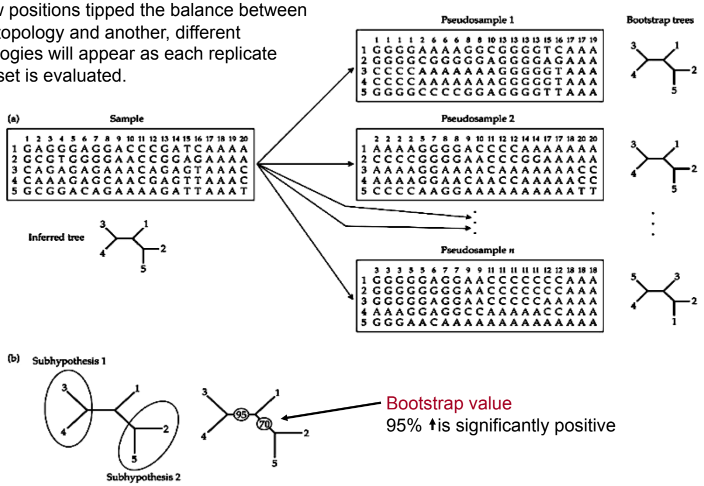
# Bootstrapping to get the best trees

Main outline of algorithm:

1. Select random columns from a multiple alignment – one column can then appear several times
2. Build a phylogenetic tree based on the random sample from (1)
3. Repeat (1), (2) many (say, 1000) times
4. Output the tree that is constructed most frequently

Jackknifing: Similar to bootstrapping; Generates a number of randomized data sets that are sampled without replacements -> each data set is smaller than the original

If few positions tipped the balance between one topology and another, different topologies will appear as each replicate dataset is evaluated.



```
Sample
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 G A G G G A G G A C C C G A T C A A A A
2 G C G T G G G G A A C C G G A G A A A A
3 C A G A G A G A A A C A G A G T A A A C
4 C A A A G A G C A A C G A G T T A A A C
5 G C G G A C A G A A A A G A T T A A A T
```

```
Pseudosample 1
  1 1 1 1 2 6 6 6 8 8 10 13 13 13 13 15 16 17 17 19
1 G G G G A A A A G G C G G G G T C A A A
2 G G G G C G G G G G A G G G G A G A A A
3 C C C C A A A A A A A G G G G T A A A A
4 C C C C A A A A A A A G G G G T A A A A
5 G G G G C C C C G G A G G G G T T A A A
```

```
Pseudosample 2
  2 2 2 2 5 7 8 8 9 10 11 12 12 14 14 17 17 18 20 20
1 A A A A G G G G A C C C C A A A A A A A
2 C C C C G G G G A A C C C C G G A A A A
3 A A A A G G A A A A C A A A A A A A C C
4 A A A A G G A A C A A C C A A A A A C C
5 C C C C A A G G A A A A A A A A A A T T
```

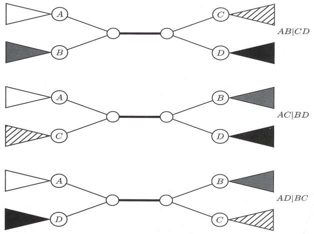
```
Pseudosample n
  3 3 3 5 5 6 7 7 9 9 11 11 11 11 11 12 12 18 18 18
1 G G G G G A G G A A C C C C C C C A A A
2 G G G G G G G G A A C C C C C C C A A A
3 G G G G G A G G A A C C C C C A A A A A
4 A A A G G A G G C C A A A A A C C A A A
5 G G G A A C A A A A A A A A A A A A A A
```

Bootstrap value

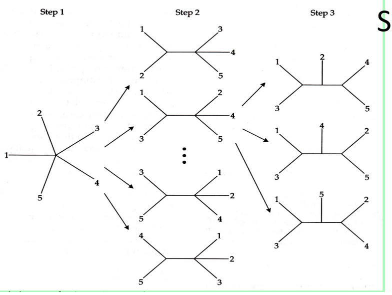
95% ↑is significantly positive

## Nearest Neighbor Interchange

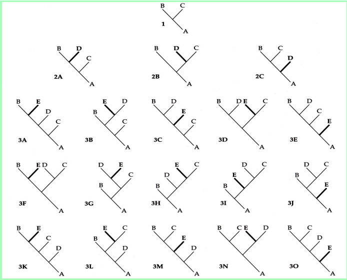
- A Branch Swapping algorithm
- Only evaluates a subset of all possible trees
- Defines a neighbor of a tree as one reachable by a nearest neighbor interchange
  - A rearrangement of the four subtrees defined by one internal edge
  - Only three different rearrangements per edge



- Start with an arbitrary tree and check its neighbors
- Move to a neighbor if it provides the best improvement in parsimony score
- No way of knowing if the result is the most parsimonious tree
- Could be stuck in local optimum



Star decomposition



Stepwise addition

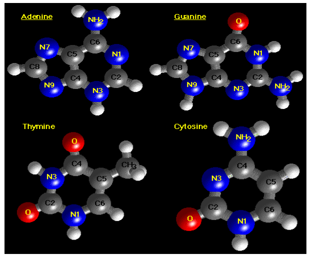
## BioInformatics 8: Information theory

- Given a text composed from an alphabet of 32 letters (each letter equally probable)
- Person A chooses a letter X (randomly)
- Person B wants to know this letter
- B may ask only binary questions
- Question: how many binary questions must B ask in order to learn which letter X was chosen by A
- Answer: entropy H(X), Here: H(X) = 5 bit

how many binary questions must person B ask in order to learn which DNA base was chosen by person A?

Purines

Pyrimidines



1 bit

1 bit

## Conditional entropy

- Given a text composed from an alphabet of 32 letters (each letter equally probable)
- Person A chooses a letter X (randomly)
- Person B wants to know this letter
- B may ask only binary questions
- A may tell B the letter Y preceding X
- Question: how many binary questions must B ask in order to learn which letter X was chosen by A
- Answer: conditional entropy H(X|Y) ; H(X|Y) <= H(X)
- In worst case – namely if B ignores all "information" in Y about X – B needs H(X) binary questions
- Under no circumstances should B need more than H(X) binary questions
- Knowledge of Y cannot increase the number of binary questions

## Mutual information

Compare two situations:

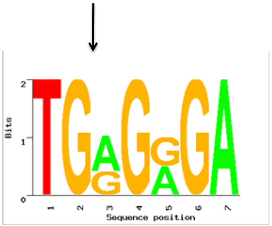
- I: learn X without knowing Y
- II: learn X with knowing Y
- How many binary questions in case of I? → H(X)
- How many binary questions in case of II? → H(X|Y)

- Question: How many binary questions could B save in case of II?
- Question: How many binary questions could B save by knowing Y?
- Answer: I(X;Y) = H(X) – H(X|Y) where I(X;Y) = information in Y about X
- H(X|Y) <= H(X) → I(X;Y) >= 0
- Example 1: random sequence composed of A, C, G, T (equally probable)
- H(X) = 2 bit; H(X|Y) = 2 bit; I(X;Y) = H(X) – H(X|Y) = 0 bit
- Example 2: deterministic sequence … ACGT ACGT ACGT ACGT …
- H(X) = 2 bit; H(X|Y) = 0 bit; I(X;Y) = H(X) – H(X|Y)= 2 bit

## Identifying Motifs and generating Motif Logo

- Genes are turned on or off by regulatory proteins;
- These proteins bind to a short DNA sequence called a motif (TFBS)
- So finding the same motif in multiple genes' regulatory regions suggests a regulatory relationship amongst those genes
- Motifs can mutate on non important bases
- The five motifs in five different genes have mutations in position 3 and 5
- Representations called *motif logos* illustrate the conserved and variable regions of a motif

TGGGGGA
TGAGAGA
TGGGGGA
TGAGAGA
TGAGGGA



## Information Content of a DNA Motif

Information at position j: $I_j = H_{before} - H_{after}$

Motif probabilities: $p_k$ (k = A, C, G, T)

Background probabilities: $q_k = \frac{1}{4}$ (k = A, C, G, T)

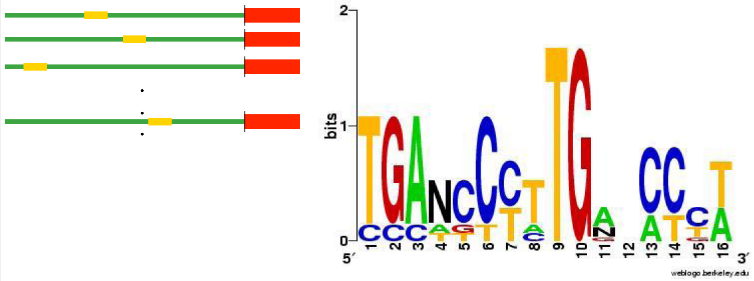
$$I_j = -\sum_{k=1}^{4} q_k \log_2 q_k - -\sum_{k=1}^{4} p_k \log_2 p_k = 2 - H_j$$

$$I_{motif} = \sum_{j=1}^{w} I_j = 2w - H_{motif} \text{ (motif of width w bases)}$$

Log base 2 gives entropy/information in 'bits'

## Sequence Logos

- http://weblogo.berkeley.edu/



- Height of letters given by $p_i(x) * I_i$

## Entropy estimation of alignment

- Define frequencies for the occurrence of each letter in each column of multiple alignment

  $p_A = 1$, $p_T=p_G=p_C=0$ (1st column)

  $p_A = 0.75$, $p_T = 0.25$, $p_G=p_C=0$ (2nd column)

  $p_A = 0.50$, $p_T = 0.25$, $p_C=0.25$ $p_G=0$ (3rd column)

  AAA
  AAA
  AAT
  ATC

- Compute entropy of each column

$$-\sum_{X=A,T,G,C} p_X \log p_X$$

## Multiple Alignment: Entropy Score

Best case

$$entropy\begin{pmatrix} A \\ A \\ A \\ A \end{pmatrix} = 0$$

Worst case

$$entropy\begin{pmatrix} A \\ T \\ G \\ C \end{pmatrix} = -\sum \frac{1}{4}\log\frac{1}{4} = -4(\frac{1}{4} * -2) = 2$$

Entropy for a multiple alignment is the sum of entropies of its columns:

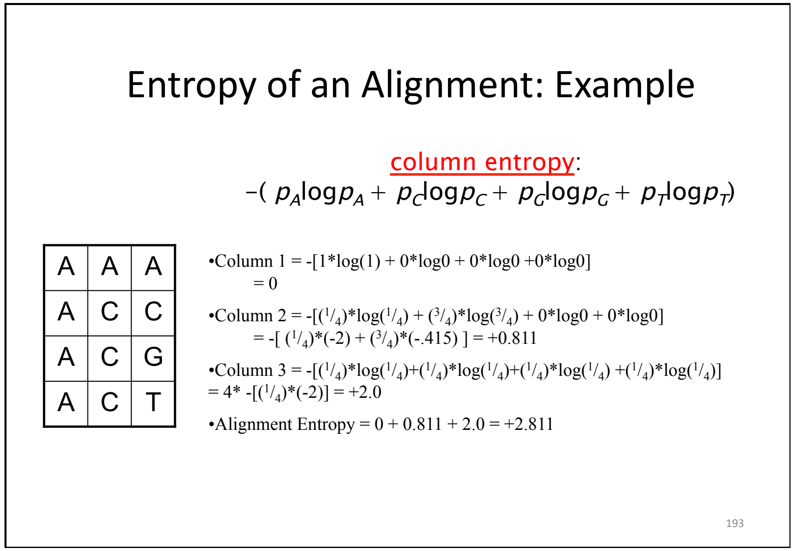
$\Sigma$ over all columns $\Sigma$ $_{X=A,T,G,C}$ $p_X \log p_X$

## Information Content

- In a positional weight matrix , PWM, convert frequencies to probabilities
- PWM W: $W_{\beta k}$ = frequency of base $\beta$ at position k
- $q_\beta$ = frequency of base $\beta$ by chance
- Information content of W:

$$\sum_k \sum_{\beta \in \{A,C,G,T\}} W_{\beta k} \log \frac{W_{\beta k}}{q_\beta}$$

- If $W_{\beta k}$ is always equal to $q_\beta$, i.e., if W is similar to random sequence, information content of W is 0.
- If W is different from q, information content is high.



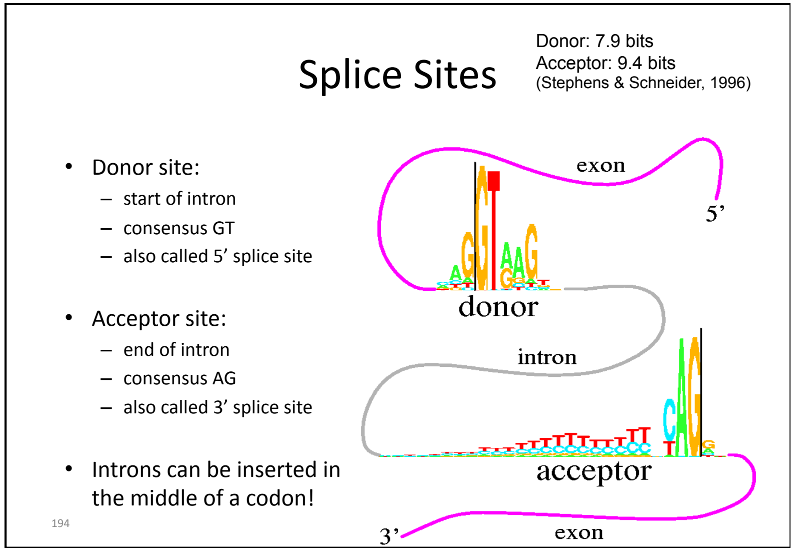
## Entropy of an Alignment: Example

column entropy:

$$-(\,p_A \log p_A + p_C \log p_C + p_G \log p_G + p_T \log p_T)$$

| | | |
|---|---|---|
| A | A | A |
| A | C | C |
| A | C | G |
| A | C | T |

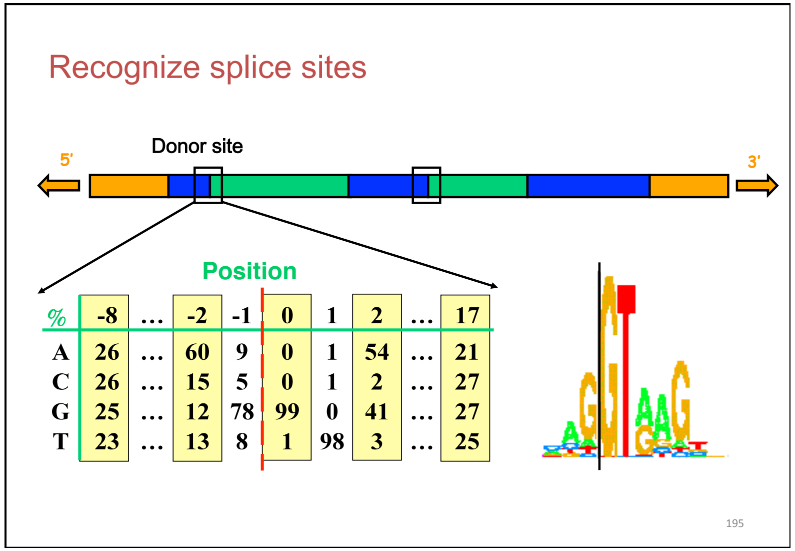
- •Column 1 = -[1*log(1) + 0*log0 + 0*log0 +0*log0]
  = 0
- •Column 2 = $-[(^1/_4)*\log(^1/_4) + (^3/_4)*\log(^3/_4) + 0*\log0 + 0*\log0]$
  $= -[\,(^1/_4)*(-2) + (^3/_4)*(-.415)\,] = +0.811$
- •Column 3 = $-[(^1/_4)*\log(^1/_4)+(^1/_4)*\log(^1/_4)+(^1/_4)*\log(^1/_4) +(^1/_4)*\log(^1/_4)]$
  $= 4* -[(^1/_4)*(-2)] = +2.0$
- •Alignment Entropy = 0 + 0.811 + 2.0 = +2.811



## Splice Sites

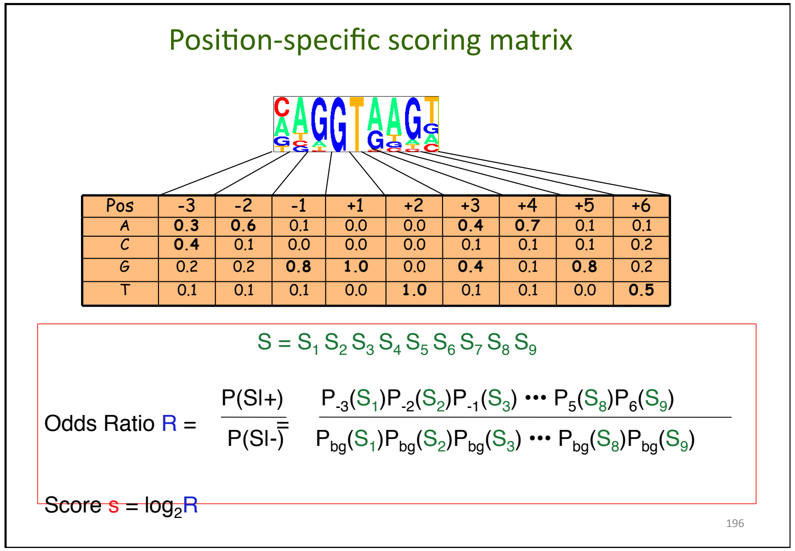
Donor: 7.9 bits
Acceptor: 9.4 bits
(Stephens & Schneider, 1996)

- Donor site:
  - start of intron
  - consensus GT
  - also called 5' splice site
- Acceptor site:
  - end of intron
  - consensus AG
  - also called 3' splice site
- Introns can be inserted in the middle of a codon!



## Recognize splice sites

| % | -8 | … | -2 | -1 | 0 | 1 | 2 | … | 17 |
|---|---|---|---|---|---|---|---|---|---|
| A | 26 | … | 60 | 9 | 0 | 1 | 54 | … | 21 |
| C | 26 | … | 15 | 5 | 0 | 1 | 2 | … | 27 |
| G | 25 | … | 12 | 78 | 99 | 0 | 41 | … | 27 |
| T | 23 | … | 13 | 8 | 1 | 98 | 3 | … | 25 |



## Position-specific scoring matrix

| Pos | -3 | -2 | -1 | +1 | +2 | +3 | +4 | +5 | +6 |
|---|---|---|---|---|---|---|---|---|---|
| A | **0.3** | **0.6** | 0.1 | 0.0 | 0.0 | **0.4** | **0.7** | 0.1 | 0.1 |
| C | **0.4** | 0.1 | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 | 0.1 | 0.2 |
| G | 0.2 | 0.2 | **0.8** | **1.0** | 0.0 | **0.4** | 0.1 | **0.8** | 0.2 |
| T | 0.1 | 0.1 | 0.1 | 0.0 | **1.0** | 0.1 | 0.1 | 0.0 | **0.5** |

$$S = S_1\,S_2\,S_3\,S_4\,S_5\,S_6\,S_7\,S_8\,S_9$$

$$\text{Odds Ratio } R = \frac{P(S|+)}{P(S|-)} = \frac{P_{-3}(S_1)P_{-2}(S_2)P_{-1}(S_3) \cdots P_5(S_8)P_6(S_9)}{P_{bg}(S_1)P_{bg}(S_2)P_{bg}(S_3) \cdots P_{bg}(S_8)P_{bg}(S_9)}$$

Score $s = \log_2 R$

# Motifs: Profiles and Consensus

```
Alignment     a G g t a c T t
              C c A t a c g t
              a c g t T A g t
              a c g t C c A t
              C c g t a c g G
_________________

Profile    A  3 0 1 0 3 1 1 0
           C  2 4 0 0 1 4 0 0
           G  0 1 4 0 0 0 3 1
           T  0 0 0 5 1 0 1 4
_________________

Consensus     A C G T A C G T
```

- Line up the patterns by their start indexes

  **s** = ($s_1$, $s_2$, …, $s_t$)
- Construct matrix profile with frequencies of each nucleotide in columns
- Consensus nucleotide in each position has the highest score in column
- Think of consensus as an "ancestor" motif, from which mutated motifs emerged
- The distance between a real motif and the consensus sequence is generally less than that for two real motifs

197

## Predicting the number of sites

Association between adjacent bases will lead to association between more distant bases, and an estimate of how far the relations extend may be found from Markov Chain theory.

Without invoking any biological mechanism, a Markov chain of order k supposes that the base present at a certain position in a sequence depends only on the bases present at the previous k positions.

$$p(GGATCC) = p(G)p(G)p(A)p(T)p(C)p(C)$$

For a zero order Markov chain we estimate the frequency of a word from base composition alone

$$p(GGATCC) = \frac{p(GG)p(GA)p(AT)p(TC)p(CC)}{p(G)p(A)p(T)p(C)}$$

first order Markov chain model can be used to estimate the same frequency

$$p(GGATCC) = \frac{p(GGA)p(GAT)p(ATC)p(TCC)}{p(GA)p(AT)p(TC)}$$

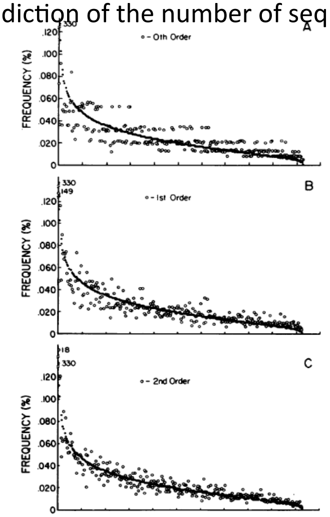
2nd order Markov chain that uses di and tri nucleotide frequencies

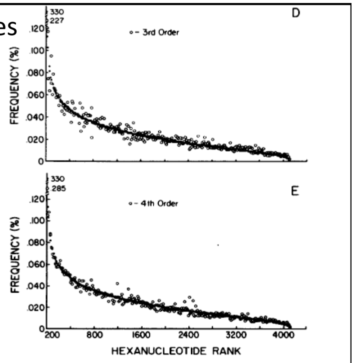
$$p(GGATCC) = \frac{p(GGAT)p(GATC)p(ATCC)}{p(GAT)p(ATC)}$$

3rd order Markov chain

198

## Prediction of the number of sequences



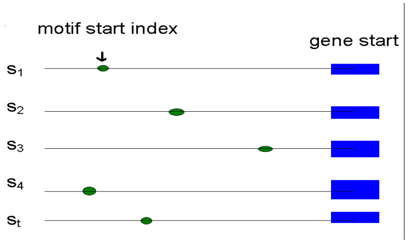


Yeast genome: Frequencies of each hexanucleotide were plotted from highest to lowest abundance along with values determined by each Markov chain

199

# Gibbs Sampling

- **Gibbs Sampling is** an iterative procedure that discards one *l*-mer after each iteration and replaces it with a new one.
- Gibbs Sampling proceeds slowly and chooses new *l*-mers at random increasing the odds that it will converge to the correct solution.



Genes regulated by same transcription factor

200

## How Gibbs Sampling Works

1) Randomly choose starting positions **s** = ($s_1$,...,$s_t$) and form the set of *l*-mers associated with these starting positions.
2) Randomly choose one of the *t* sequences.
3) Create a profile **p** from the other *t* -1 sequences.
4) For each position in the removed sequence, calculate the probability that the *l*-mer starting at that position was generated by **p**.
5) Choose a new starting position for the removed sequence based on the probabilities calculated in step 4.
6) Repeat steps 2-5 until there is no improvement

201

## Gibbs Sampling Algorithm

## Gibbs Sampling – Motif Positions

- For each position, $r$, in the omitted sequence, $s_t$, calculate a weight:

$$\frac{\prod_{k=r}^{r+W-1}\prod_{i=1}^{4} p_{i,k-r+1}^{I(s_{t,k}=i)}}{\prod_{k=r}^{r+W-1}\prod_{i=1}^{4} p_{i0}^{I(s_{t,k}=i)}}$$

  i.e. the probability of motif to background score
- New motif location in $s_t$ is choosen according to these weights. That is, instead of giving each position in the sequence equal weight so that each position has a $\frac{1}{L_t}$ chance of being selected, the chance of being selected is proportional to the weight. Large weight (meaning higher chance of the motif begin positioned there) gives large chance of selection.

## Gibbs Sampling

**Input**:

t = 5 sequences, motif length l = 8

1. GTAAACAATATTTATAGC
2. AAAATTTACCTCGCAAGG
3. CCGTACTGTCAAGCGTGG
4. TGAGTAAACGACGTCCCA
5. TACTTAACACCCTGTCAA

1) Randomly choose starting positions, ***s***=($s_1$,$s_2$,$s_3$,$s_4$,$s_5$) in the 5 sequences:

$s_1$=7GTAAACAATATTTATAGC

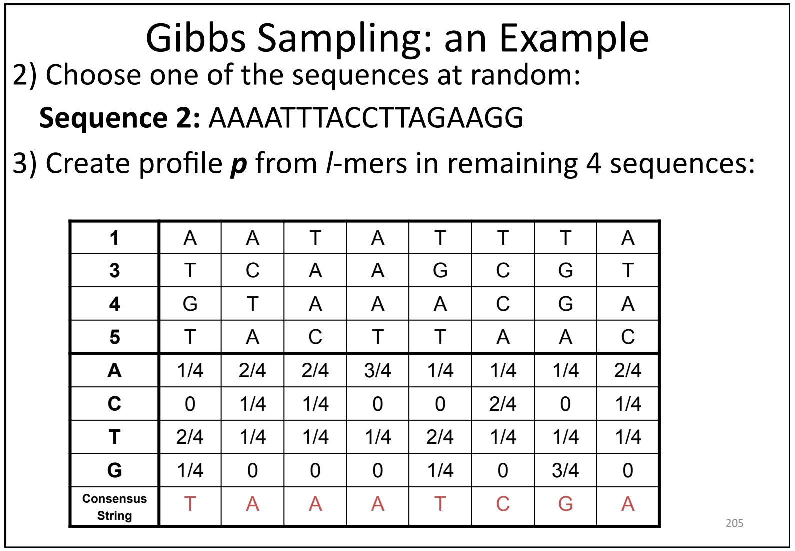
$s_2$=11 AAAATTTACCTTAGAAGG

$s_3$=9CCGTACTGTCAAGCGTGG

$s_4$=4 TGAGTAAACGACGTCCCA

$s_5$=1TACTTAACACCCTGTCAA

204



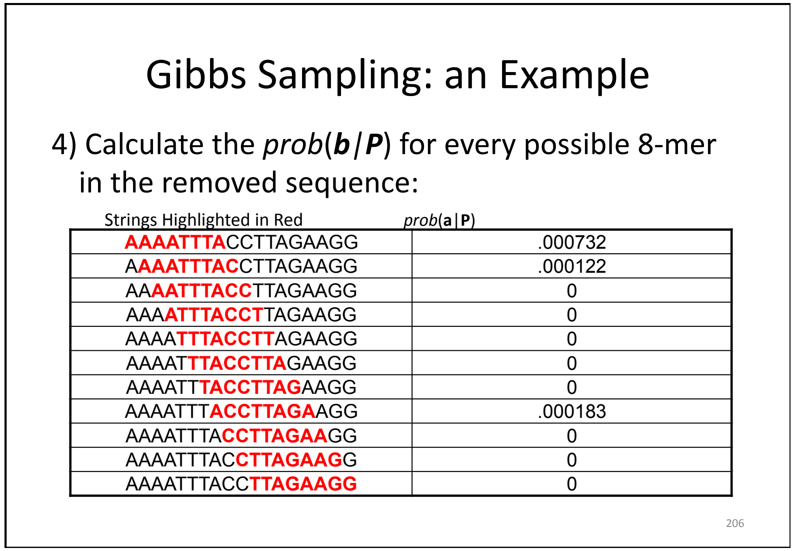
# Gibbs Sampling: an Example

2) Choose one of the sequences at random:

**Sequence 2:** AAAATTTACCTTAGAAGG

3) Create profile ***p*** from *l*-mers in remaining 4 sequences:

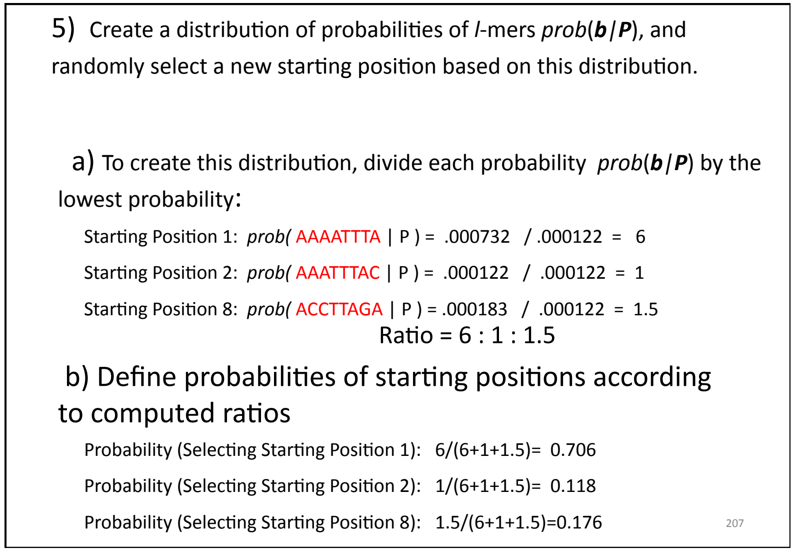
| **1** | A | A | T | A | T | T | T | A |
|---|---|---|---|---|---|---|---|---|
| **3** | T | C | A | A | G | C | G | T |
| **4** | G | T | A | A | A | C | G | A |
| **5** | T | A | C | T | T | A | A | C |
| **A** | 1/4 | 2/4 | 2/4 | 3/4 | 1/4 | 1/4 | 1/4 | 2/4 |
| **C** | 0 | 1/4 | 1/4 | 0 | 0 | 2/4 | 0 | 1/4 |
| **T** | 2/4 | 1/4 | 1/4 | 1/4 | 2/4 | 1/4 | 1/4 | 1/4 |
| **G** | 1/4 | 0 | 0 | 0 | 1/4 | 0 | 3/4 | 0 |
| **Consensus String** | T | A | A | A | T | C | G | A |



# Gibbs Sampling: an Example

4) Calculate the *prob*(***b***|***P***) for every possible 8-mer in the removed sequence:

| Strings Highlighted in Red | *prob*(**a**\|**P**) |
|---|---|
| **AAAATTTA**CCTTAGAAGG | .000732 |
| A**AAATTTAC**CTTAGAAGG | .000122 |
| AA**AATTTACC**TTAGAAGG | 0 |
| AAA**ATTTACCT**TAGAAGG | 0 |
| AAAA**TTTACCTT**AGAAGG | 0 |
| AAAAT**TTACCTTA**GAAGG | 0 |
| AAAATT**TACCTTAG**AAGG | 0 |
| AAAATTT**ACCTTAGA**AGG | .000183 |
| AAAATTTA**CCTTAGAA**GG | 0 |
| AAAATTTAC**CTTAGAAG**G | 0 |
| AAAATTTACC**TTAGAAGG** | 0 |



5) Create a distribution of probabilities of *l*-mers *prob*(***b***|***P***), and randomly select a new starting position based on this distribution.

a) To create this distribution, divide each probability *prob*(***b***|***P***) by the lowest probability:

Starting Position 1: *prob(* AAAATTTA | P ) = .000732 / .000122 = 6

Starting Position 2: *prob(* AAATTTAC | P ) = .000122 / .000122 = 1

Starting Position 8: *prob(* ACCTTAGA | P ) = .000183 / .000122 = 1.5

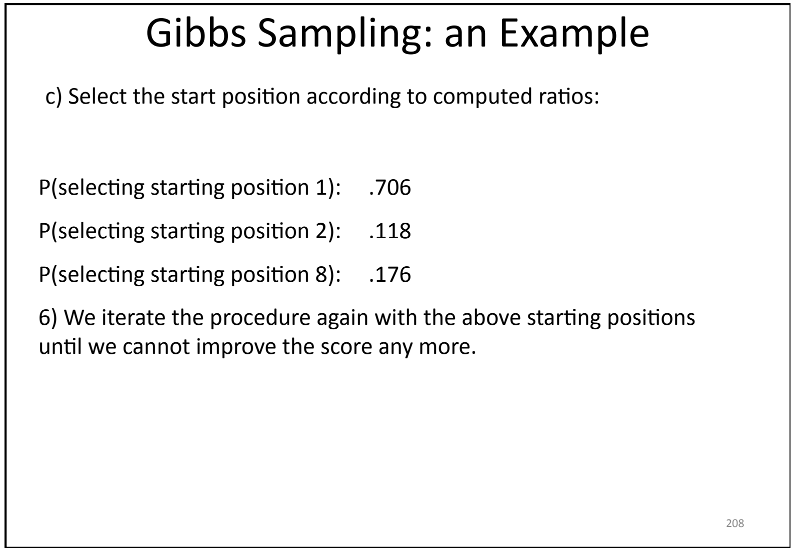
Ratio = 6 : 1 : 1.5

b) Define probabilities of starting positions according to computed ratios

Probability (Selecting Starting Position 1): 6/(6+1+1.5)= 0.706

Probability (Selecting Starting Position 2): 1/(6+1+1.5)= 0.118

Probability (Selecting Starting Position 8): 1.5/(6+1+1.5)=0.176



# Gibbs Sampling: an Example

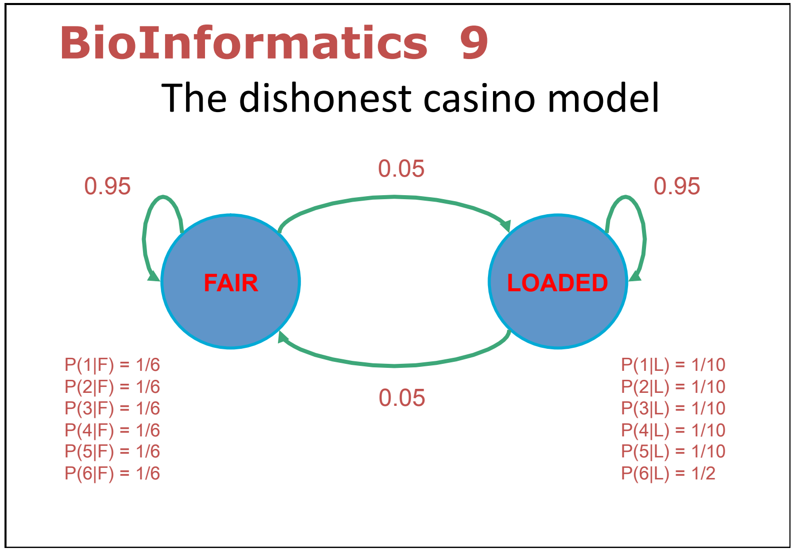
c) Select the start position according to computed ratios:

P(selecting starting position 1): .706

P(selecting starting position 2): .118

P(selecting starting position 8): .176

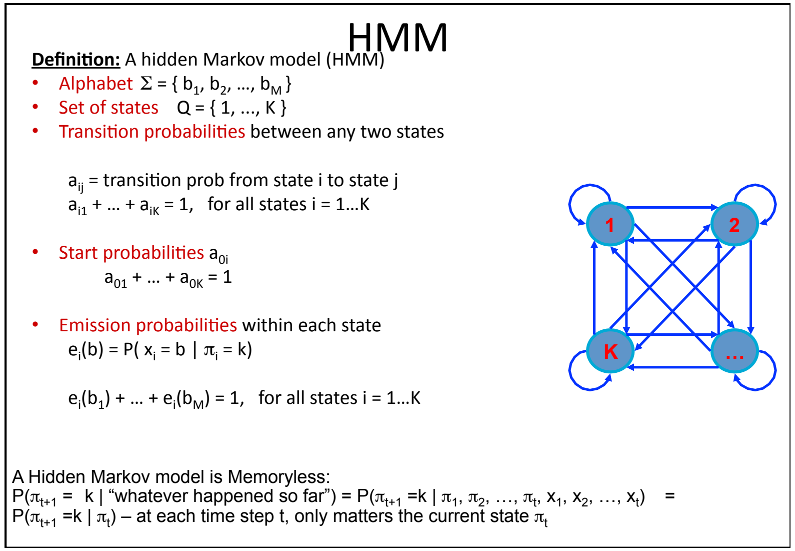
6) We iterate the procedure again with the above starting positions until we cannot improve the score any more.



# BioInformatics 9

## The dishonest casino model

| | |
|---|---|
| P(1\|F) = 1/6 | P(1\|L) = 1/10 |
| P(2\|F) = 1/6 | P(2\|L) = 1/10 |
| P(3\|F) = 1/6 | P(3\|L) = 1/10 |
| P(4\|F) = 1/6 | P(4\|L) = 1/10 |
| P(5\|F) = 1/6 | P(5\|L) = 1/10 |
| P(6\|F) = 1/6 | P(6\|L) = 1/2 |



## HMM

**Definition:** A hidden Markov model (HMM)

- Alphabet $\Sigma = \{ b_1, b_2, \ldots, b_M \}$
- Set of states $Q = \{ 1, \ldots, K \}$
- Transition probabilities between any two states

$a_{ij}$ = transition prob from state i to state j

$a_{i1} + \ldots + a_{iK} = 1$, for all states i = 1…K

- Start probabilities $a_{0i}$

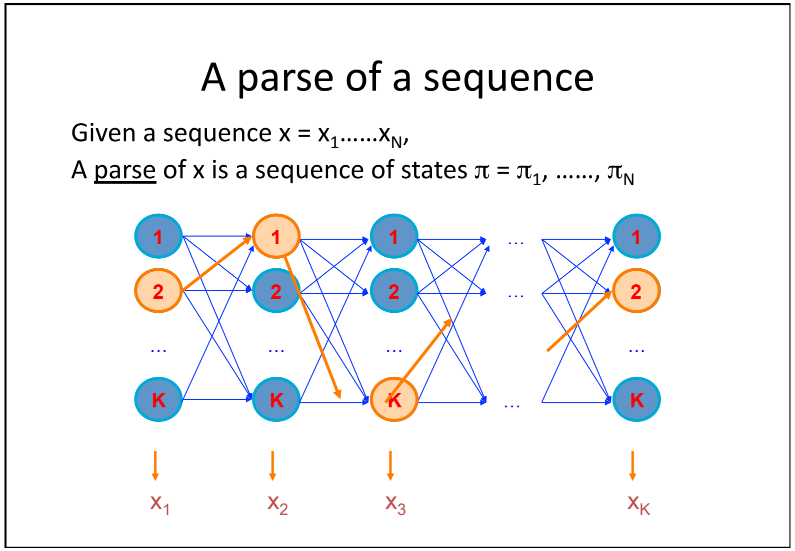
$a_{01} + \ldots + a_{0K} = 1$

- Emission probabilities within each state

$e_i(b) = P( x_i = b \mid \pi_i = k)$

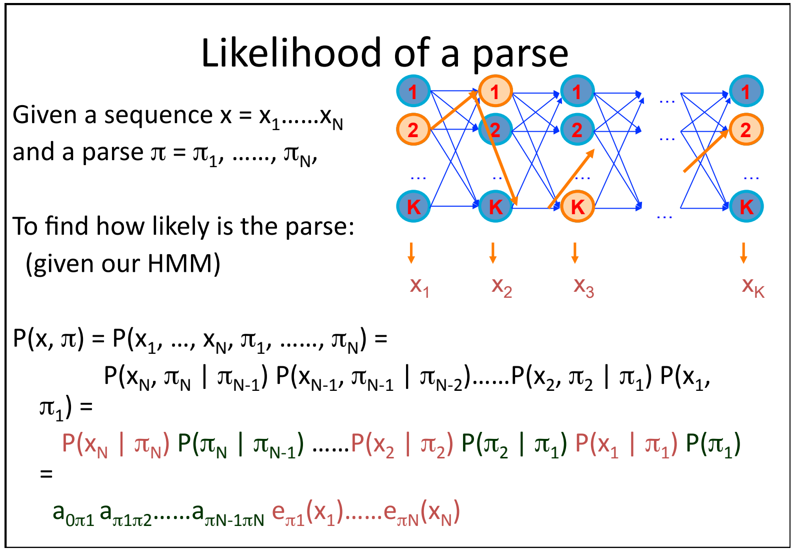
$e_i(b_1) + \ldots + e_i(b_M) = 1$, for all states i = 1…K

A Hidden Markov model is Memoryless:
$P(\pi_{t+1} = k \mid$ "whatever happened so far"$) = P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) =$
$P(\pi_{t+1} = k \mid \pi_t)$ – at each time step t, only matters the current state $\pi_t$



## A parse of a sequence

Given a sequence $x = x_1 \ldots\ldots x_N$,
A parse of x is a sequence of states $\pi = \pi_1, \ldots\ldots, \pi_N$



## Likelihood of a parse

Given a sequence $x = x_1 \ldots\ldots x_N$
and a parse $\pi = \pi_1, \ldots\ldots, \pi_N$,

To find how likely is the parse:
(given our HMM)

$$P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots\ldots, \pi_N) =$$
$$P(x_N, \pi_N \mid \pi_{N-1})\, P(x_{N-1}, \pi_{N-1} \mid \pi_{N-2}) \ldots\ldots P(x_2, \pi_2 \mid \pi_1)\, P(x_1, \pi_1) =$$
$$P(x_N \mid \pi_N)\, P(\pi_N \mid \pi_{N-1}) \ldots\ldots P(x_2 \mid \pi_2)\, P(\pi_2 \mid \pi_1)\, P(x_1 \mid \pi_1)\, P(\pi_1) =$$
$$a_{0\pi 1}\, a_{\pi 1\pi 2} \ldots\ldots a_{\pi N-1\pi N}\, e_{\pi 1}(x_1) \ldots\ldots e_{\pi N}(x_N)$$

## The three main questions on HMMs

1. Evaluation

GIVEN a HMM M, and a sequence x,

FIND Prob[ x | M ]

2. Decoding

GIVEN a HMM M, and a sequence x,

FIND the sequence $\pi$ of states that maximizes P[ x, $\pi$ | M ]

3. Learning

GIVEN a HMM M, with unspecified transition/emission probs., and a sequence x,

FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize P[ x | $\theta$ ]

## Let's not be confused by notation

P[ x | M ]: The probability that sequence x was generated by the model; The model is: architecture (#states, etc)

+ parameters $\theta = a_{ij}, e_i(.)$

So, P[ x | $\theta$ ], and P[ x ] are the same, when the architecture, and the entire model, respectively, are implied

Similarly, P[ x, $\pi$ | M ] and P[ x, $\pi$ ] are the same

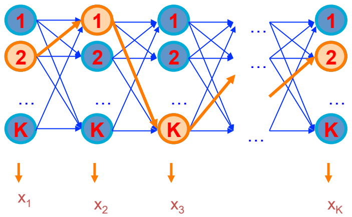
In the LEARNING problem we always write P[ x | $\theta$ ] to emphasize that we are seeking the $\theta$ that maximizes P[ x | $\theta$ ]

## Decoding

GIVEN $x = x_1x_2\ldots\ldots x_N$

We want to find $\pi = \pi_1, \ldots\ldots, \pi_N$, such that P[ x, $\pi$ ] is maximized

$\pi^* = \text{argmax}_\pi$ P[ x, $\pi$ ]



We can use dynamic programming!

Let $V_k(i) = \max_{\{\pi 1,\ldots,i-1\}} P[x_1\ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$

= Probability of most likely sequence of states ending at state $\pi_i = k$

## Decoding – main idea

Given that for all states k, and for a fixed position i,

$V_k(i) = \max_{\{\pi 1,\ldots,i-1\}} P[x_1\ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$

What is $V_k(i+1)$?

From definition,

$V_l(i+1) = \max_{\{\pi 1,\ldots,i\}} P[ x_1\ldots x_i, \pi_1, \ldots, \pi_i, x_{i+1}, \pi_{i+1} = l ]$

$= \max_{\{\pi 1,\ldots,i\}} P(x_{i+1}, \pi_{i+1} = l \mid x_1\ldots x_i, \pi_1, \ldots, \pi_i) P[x_1\ldots x_i, \pi_1, \ldots, \pi_i]$

$= \max_{\{\pi 1,\ldots,i\}} P(x_{i+1}, \pi_{i+1} = l \mid \pi_i ) P[x_1\ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i]$

$= \max_k P(x_{i+1}, \pi_{i+1} = l \mid \pi_i = k) \max_{\{\pi 1,\ldots,i-1\}} P[x_1\ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k] = e_l(x_{i+1}) \max_k a_{kl} V_k(i)$

# The Viterbi Algorithm

Input: $x = x_1 \ldots\ldots x_N$

**Initialization:**

$V_0(0) = 1$ (0 is the imaginary first position)

$V_k(0) = 0$, for all $k > 0$

**Iteration:**

$V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1)$

$\mathrm{Ptr}_j(i) = \mathrm{argmax}_k\, a_{kj} V_k(i-1)$

**Termination:**

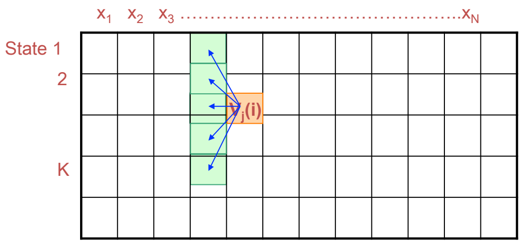
$P(x, \pi^*) = \max_k V_k(N)$

**Traceback:**

$\pi_N^* = \mathrm{argmax}_k\, V_k(N)$

$\pi_{i-1}^* = \mathrm{Ptr}_{\pi i}\,(i)$

# The Viterbi Algorithm



Similar to "aligning" a set of states to a sequence

**Time:** $O(K^2N)$

**Space:** $O(KN)$

Underflows are a significant problem

$P[\, x_1, \ldots, x_i, \pi_1, \ldots, \pi_i \,] = a_{0\pi1}\, a_{\pi1\pi2} \ldots\ldots a_{\pi i}\, e_{\pi1}(x_1) \ldots\ldots e_{\pi i}(x_i)$

These numbers become extremely small – underflow

**Solution:** Take the logs of all values

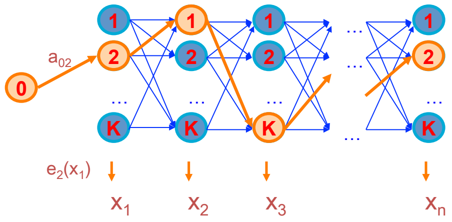
$V_l(i) = \log e_k(x_i) + \max_k [\, V_k(i-1) + \log a_{kl} \,]$

# Generating a sequence by the model

Given a HMM, we can generate a sequence of length n as follows:

Start at state $\pi_1$ according to prob $a_{0\pi1}$

1. Emit letter $x_1$ according to prob $e_{\pi1}(x_1)$
2. Go to state $\pi_2$ according to prob $a_{\pi1\pi2}$
3. … until emitting $x_n$



# A couple of questions

Given a sequence x,

- What is the probability that x was generated by the model?
- Given a position i, what is the most likely state that emitted $x_i$?

Example: the dishonest casino

Say x = 12341623162616364616234161221341

Most likely path: $\pi$ = FF……F

However: marked letters more likely to be L than unmarked letters

## Evaluation

We will develop algorithms that allow us to compute:

$P(x)$ Probability of x given the model

$P(x_i\ldots x_j)$ Probability of a substring of x given the model

$P(\pi_i = k \mid x)$ Probability that the ith state is k, given x

A more refined measure of which states x may be in

## The Forward Algorithm

We want to calculate

$P(x)$ = probability of x, given the HMM

Sum over all possible ways of generating x:

$$P(x) = \Sigma_\pi P(x, \pi) = \Sigma_\pi P(x \mid \pi) P(\pi)$$

To avoid summing over an exponential number of paths $\pi$, define

$f_k(i) = P(x_1\ldots x_i, \pi_i = k)$ (the forward probability)

## The Forward Algorithm – derivation

Define the forward probability:

$$f_l(i) = P(x_1\ldots x_i, \pi_i = l)$$

$$= \Sigma_{\pi 1\ldots\pi i-1} P(x_1\ldots x_{i-1}, \pi_1,\ldots, \pi_{i-1}, \pi_i = l)\, e_l(x_i)$$

$$= \Sigma_k \Sigma_{\pi 1\ldots\pi i-2} P(x_1\ldots x_{i-1}, \pi_1,\ldots, \pi_{i-2}, \pi_{i-1} = k)\, a_{kl}\, e_l(x_i)$$

$$= e_l(x_i)\, \Sigma_k f_k(i-1)\, a_{kl}$$

## The Forward Algorithm

We can compute $f_k(i)$ for all k, i, using dynamic programming!

**Initialization:**

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

**Iteration:**

$f_l(i) = e_l(x_i)\, \Sigma_k f_k(i-1)\, a_{kl}$

**Termination:**

$P(x) = \Sigma_k f_k(N)\, a_{k0}$

Where, $a_{k0}$ is the probability that the terminating state is k (usually $= a_{0k}$)

## Relation between Forward and Viterbi

**VITERBI**

**Initialization:**

$V_0(0) = 1$

$V_k(0) = 0$, for all $k > 0$

**Iteration:**

$V_j(i) = e_j(x_i)\ \mathbf{max_k}\ V_k(i-1)\, a_{kj}$

**Termination:**

$P(x, \pi^*) = \mathbf{max_k}\ V_k(N)$

**FORWARD**

**Initialization:**

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

**Iteration:**

$f_l(i) = e_l(x_i)\ \boldsymbol{\Sigma_k}\ f_k(i-1)\, a_{kl}$

**Termination:**

$P(x) = \boldsymbol{\Sigma_k}\ f_k(N)\, a_{k0}$

## Motivation for the Backward Algorithm

We want to compute

$P(\pi_i = k \mid x)$,

the probability distribution on the ith position, given x

We start by computing

$$P(\pi_i = k, x) = P(x_1 \ldots x_i, \pi_i = k, x_{i+1} \ldots x_N)$$

$$= P(x_1 \ldots x_i, \pi_i = k)\ P(x_{i+1} \ldots x_N \mid x_1 \ldots x_i, \pi_i = k)$$

$$= \underbrace{P(x_1 \ldots x_i, \pi_i = k)}_{\text{Forward, } f_k(i)}\ \underbrace{P(x_{i+1} \ldots x_N \mid \pi_i = k)}_{\text{Backward, } b_k(i)}$$

## The Backward Algorithm – derivation

Define the backward probability:

$$b_k(i) = P(x_{i+1} \ldots x_N \mid \pi_i = k)$$

$$= \Sigma_{\pi i+1 \ldots \pi N}\ P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N \mid \pi_i = k)$$

$$= \Sigma_l\ \Sigma_{\pi i+1 \ldots \pi N}\ P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1} = l, \pi_{i+2}, \ldots, \pi_N \mid \pi_i = k)$$

$$= \Sigma_l\ e_l(x_{i+1})\, a_{kl}\ \Sigma_{\pi i+1 \ldots \pi N}\ P(x_{i+2}, \ldots, x_N, \pi_{i+2}, \ldots, \pi_N \mid \pi_{i+1} = l)$$

$$= \Sigma_l\ e_l(x_{i+1})\, a_{kl}\ b_l(i+1)$$

## The Backward Algorithm

We can compute $b_k(i)$ for all k, i, using dynamic programming

**Initialization:**

$b_k(N) = a_{k0}$, for all k

**Iteration:**

$b_k(i) = \Sigma_l\ e_l(x_{i+1})\, a_{kl}\ b_l(i+1)$

**Termination:**

$P(x) = \Sigma_l\ a_{0l}\ e_l(x_1)\ b_l(1)$

What is the running time, and space required, for Forward, and Backward?

Time: $O(K^2N)$

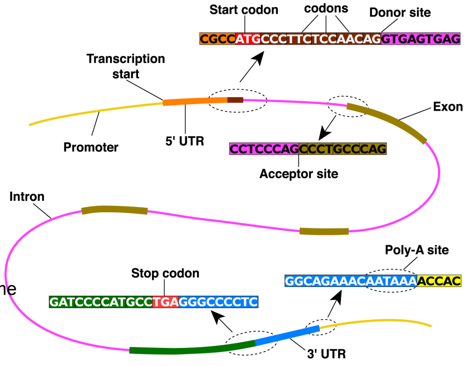
Space: $O(KN)$

Assume we are given a DNA sequence that begins in an exon, contains one splice site and ends in an intron. The problem is to identify where the switch from exon to intron occurred.
For us to guess intelligently, the sequences of exons, splice sites and introns must have different statistical properties.

Let's imagine some simple differences: say that exons have a uniform base composition on average (25% each base), introns are A+T rich (say, 40% each for A/T, 10% each for C/G), and the 5'SS consensus nucleotide is almost always a G (say, 95% G and 5% A).

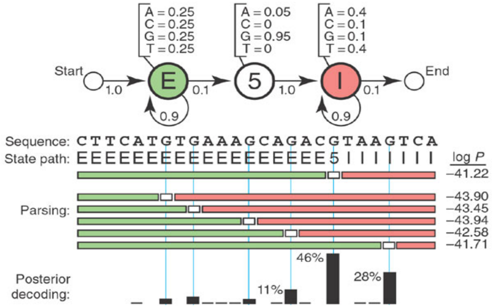
Starting from this information, we can draw an HMM that invokes three states, one for each of the three labels we might assign to a nucleotide: E (exon), 5 (5'SS) and I (intron).

Gene Structure



HUMAN GENES

Comprise about 3% of the genome
Average gene length: ~ 8,000 bp
Average of 5-6 exons/gene
Average exon length: ~200 bp
Average intron length: ~2,000 bp
~8% genes have a single exon
Some exons can be as small as 1 or 3 bp.
HUMFMR1S is not atypical:
17 exons 40-60 bp long, comprising 3% of a 67,000 bp gene



Each state has its own emission probabilities (shown above the states), which model the base composition of exons, introns and the consensus G at the 5'SS.

Each state also has transition probabilities (arrows), the probabilities of moving from this state to a new state.

The transition probabilities describe the linear order in which we expect the states to occur: one or more Es, one 5, one or more Is.

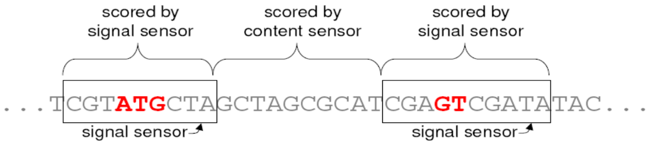
How confident are we that the fifth G is the right choice?

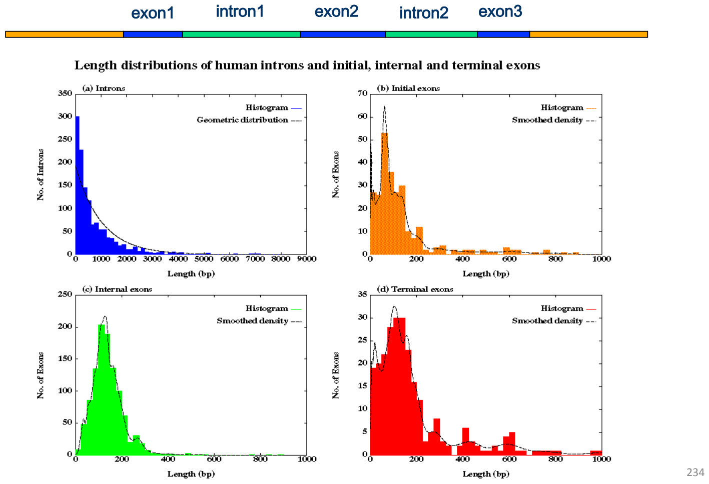
Our confidence will depend on *posterior decoding*.

Posterior decoding uses two dynamic programming algorithms called Forward and Backward, which have some similarity with Viterbi, but they sum over possible paths instead of choosing the best.

# Genescan model

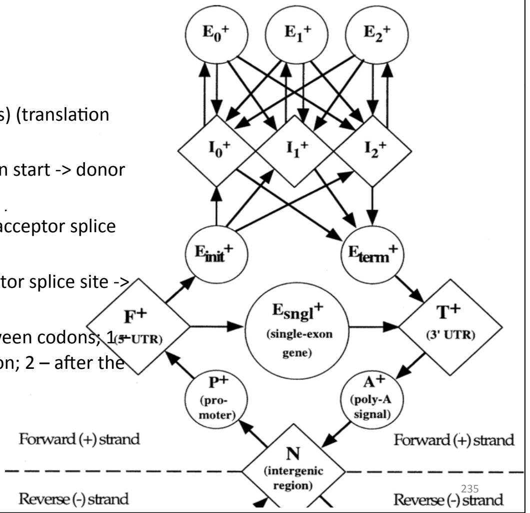
- Duration of states – length distributions of
  - Exons (coding)
  - Introns (non coding)
- Signals at state transitions
  - ATG Codon for gene start
  - Stop Codon TAG/TGA/TAA
  - Exon/Intron and Intron/Exon Splice Sites
- Emissions
  - Coding potential and frame at exons
  - Intron emissions



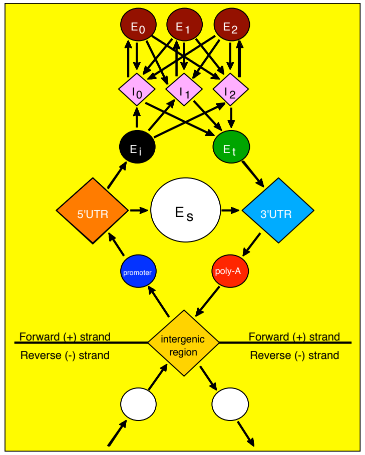


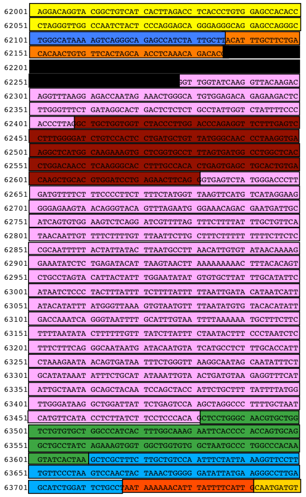
# GenScan

- N - intergenic region
- P - promoter
- F - 5' untranslated region
- $E_{sngl}$ – single exon (intronless) (translation start -> stop codon)
- $E_{init}$ – initial exon (translation start -> donor splice site)
- $E_k$ – phase k internal exon (acceptor splice site -> donor splice site)
- $E_{term}$ – terminal exon (acceptor splice site -> stop codon)
- $I_k$ – phase k intron: 0 – between codons; 1 – after the first base of a codon; 2 – after the second base of a codon



## GENSCAN (Burge & Karlin)





```
62001 AGGACAGGTA CGGCTGTCAT CACTTAGACC TCACCCTGTG GAGCCACACC
62051 CTAGGGTTGG CCAATCTACT CCCAGGAGCA GGGAGGGCAG GAGCCAGGGC
62101 TGGGCATAAA AGTCAGGGCA GAGCCATCTA TTGCTTACAT TTGCTTCTGA
62151 CACAACTGTG TTCACTAGCA ACCTCAAACA GACACC
62201
62251                                  GGT TGGTATCAAG GTTACAAGAC
62301 AGGTTTAAGG AGACCAATAG AAACTGGGCA TGTGGAGACA GAGAAGACTC
62351 TTGGGTTTCT GATAGGCACT GACTCTCTCT GCCTATTGGT CTATTTTCCC
62401 ACCCTTAGGC TGCTGGTGGT CTACCCTTGG ACCCAGAGGT TCTTTGAGTC
62451 CTTTGGGGAT CTGTCCACTC CTGATGCTGT TATGGGCAAC CCTAAGGTGA
62501 AGGCTCATGG CAAGAAAGTG CTCGGTGCCT TTAGTGATGG CCTGGCTCAC
62551 CTGGACAACC TCAAGGGCAC CTTTGCCACA CTGAGTGAGC TGCACTGTGA
62601 CAAGCTGCAC GTGGATCCTG AGAACTTCAG GGTGAGTCTA TGGGACCCTT
62651 GATGTTTTCT TTCCCCTTCT TTTCTATGGT TAAGTTCATG TCATAGGAAG
62701 GGGAGAAGTA ACAGGGTACA GTTTAGAATG GGAAACAGAC GAATGATTGC
62751 ATCAGTGTGG AAGTCTCAGG ATCGTTTTAG TTTCTTTTAT TTGCTGTTCA
62801 TAACAATTGT TTTCTTTTGT TTAATTCTTG CTTTCTTTTT TTTTCTTCTC
62851 CGCAATTTTT ACTATTATAC TTAATGCCTT AACATTGTGT ATAACAAAAG
62901 GAAATATCTC TGAGATACAT TAAGTAACTT AAAAAAAAAC TTTACACAGT
62951 CTGCCTAGTA CATTACTATT TGGAATATAT GTGTGCTTAT TTGCATATTC
63001 ATAATCTCCC TACTTTATTT TCTTTTATTT TTAATTGATA CATAATCATT
63051 ATACATATTT ATGGGTTAAA GTGTAATGTT TTAATATGTG TACACATATT
63101 GACCAAATCA GGGTAATTTT GCATTTGTAA TTTTAAAAAA TGCTTTCTTC
63151 TTTTAATATA CTTTTTTGTT TATCTTATTT CTAATACTTT CCCTAATCTC
63201 TTTCTTTCAG GGCAATAATG ATACAATGTA TCATGCCTCT TTGCACCATT
63251 CTAAAGAATA ACAGTGATAA TTTCTGGGTT AAGGCAATAG CAATATTTCT
63301 GCATATAAAT ATTTCTGCAT ATAAATTGTA ACTGATGTAA GAGGTTTCAT
63351 ATTGCTAATA GCAGCTACAA TCCAGCTACC ATTCTGCTTT TATTTTATGG
63401 TTGGGATAAG GCTGGATTAT TCTGAGTCCA AGCTAGGCCC TTTTGCTAAT
63451 CATGTTCATA CCTCTTATCT TCCTCCCACA GCTCCTGGGC AACGTGCTGG
63501 TCTGTGTGCT GGCCCATCAC TTTGGCAAAG AATTCACCCC ACCAGTGCAG
63551 GCTGCCTATC AGAAAGTGGT GGCTGGTGTG GCTAATGCCC TGGCCCACAA
63601 GTATCACTAA GCTCGCTTTC TTGCTGTCCA ATTTCTATTA AAGGTTCCTT
63651 TGTTCCCTAA GTCCAACTAC TAAACTGGGG GATATTATGA AGGGCCTTGA
63701 GCATCTGGAT TCTGCCTAAT AAAAAACATT TATTTTCATT GCAATGATGT
```

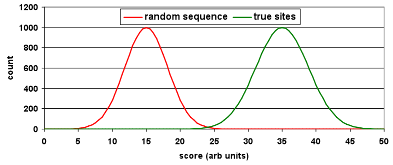
## Assessing performance: Sensitivity and Specificity

- Testing of predictions is performed on sequences where the gene structure is known
- ***Sensitivity*** is the fraction of known genes (or bases or exons) correctly predicted: **Sn=N$_{\text{True Positives}}$ /N$_{\text{All True}}$**
  - "Am I finding the things that I'm supposed to find?
- ***Specificity*** is the fraction of predicted genes (or bases or exons) that correspond to true genes: **Sp=N$_{\text{True Positives}}$ /N$_{\text{All Positives}}$**
  - "What fraction of my predictions are true?
- In general, increasing one decreases the other

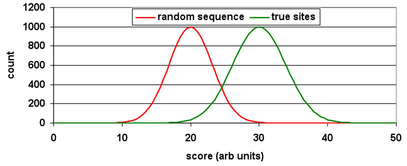
| | Accuracy per nucleotide | | | Accuracy per exon | | | | |
|---|---|---|---|---|---|---|---|---|
| Method | Sn | Sp | AC | Sn | Sp | (Sn+Sp)/2 | ME | WE |
| GENSCAN | 0.93 | 0.93 | 0.91 | 0.78 | 0.81 | 0.8 | 0.09 | 0.05 |
| FGENEH | 0.77 | 0.85 | 0.78 | 0.61 | 0.61 | 0.61 | 0.15 | 0.11 |
| GeneID | 0.63 | 0.81 | 0.67 | 0.44 | 0.45 | 0.45 | 0.28 | 0.24 |
| GeneParser2 | 0.66 | 0.79 | 0.66 | 0.35 | 0.39 | 0.37 | 0.29 | 0.17 |
| GenLang | 0.72 | 0.75 | 0.69 | 0.5 | 0.49 | 0.5 | 0.21 | 0.21 |
| GRAILII | 0.72 | 0.84 | 0.75 | 0.36 | 0.41 | 0.38 | 0.25 | 0.1 |
| SORFIND | 0.71 | 0.85 | 0.73 | 0.42 | 0.47 | 0.45 | 0.24 | 0.14 |
| Xpound | 0.61 | 0.82 | 0.68 | 0.15 | 0.17 | 0.16 | 0.32 | 0.13 |

Sn = Sensitivity
Sp = Specificity
Ac = Approximate Correlation
ME = Missing Exons
WE = Wrong Exons

## Specificity/Sensitivity Tradeoffs



Ideal Distribution of Scores



More Realistically…

$$Sn = \frac{TruePositive}{AllTrue} = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$Sp = \frac{TruePositive}{AllPositive} = \frac{TruePositive}{TruePositive + FalsePositive}$$

Correlation Coefficient

$$CC = \frac{[(TP)(TN) - (FP)(FN)]}{\sqrt{(AN)(PP)(AP)(PN)}}$$

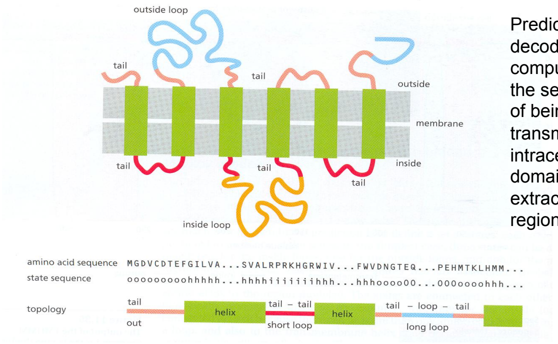
$$AN = TN + FP; AP = TP + FN;$$

$$PP = TP + FP; PN = TN + FN$$

TMHMM: Prediction of transmembrane topology of protein sequence
Model consists of submodels for:

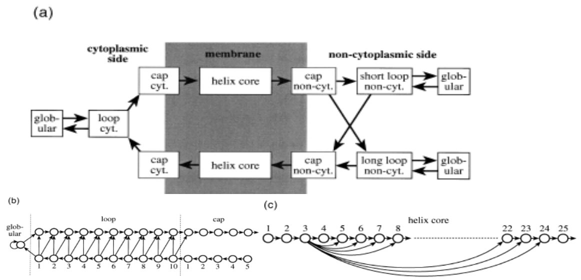
- helix core and cap regions (cytoplasmic and extracellular)
- cytoplasmic and extracellular loop regions
- globular domain regions

Trained form 160 proteins with experimentally determined transmembrane helices.



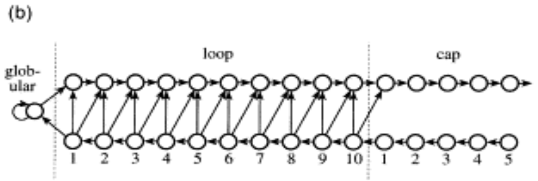
Prediction method: Posterior decoding, the program computes for each residue of the sequence the probability of being part if a transmembrane helix, an intracellular loop or globular domain region, or an extracellular loop or domain region.

## Model architecture of TMHMM

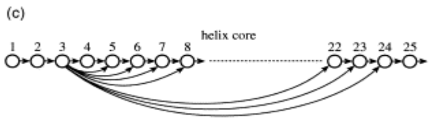


TMHMM: uses cyclic model with 7 states for
- TM helix core
- TM helix caps on the N- and C-terminal side
- non-membrane region on the cytoplasmic side
- 2 non-membrane regions on the non-cytoplasmic side (for short and long loops to account for different membrane insertion mechanism)
- a globular domain state in the middle of each non-membrane region

## TMHMM model architecture: submodels



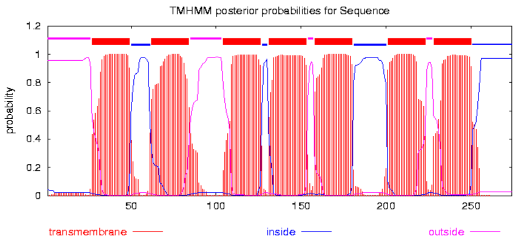
Detailed structure of the inside and outside loop models and helix cap models



The transitions from state 3 to non-adjacent states model the length distribution of trans-membrane helices.

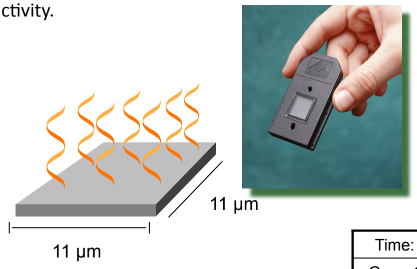
## TMHMM-Output

```
# Sequence Length: 274
# Sequence Number of predicted TMHs:  7
# Sequence Exp number of AAs in TMHs: 153.74681
# Sequence Exp number, first 60 AAs:  22.08833
# Sequence Total prob of N-in:        0.04171
# Sequence POSSIBLE N-term signal sequence
Sequence        TMHMM2.0        outside      1    26
Sequence        TMHMM2.0        TMhelix     27    49
Sequence        TMHMM2.0        inside      50    61
Sequence        TMHMM2.0        TMhelix     62    84
Sequence        TMHMM2.0        outside     85   103
Sequence        TMHMM2.0        TMhelix    104   126
Sequence        TMHMM2.0        inside     127   130
Sequence        TMHMM2.0        TMhelix    131   153
Sequence        TMHMM2.0        outside    154   157
Sequence        TMHMM2.0        TMhelix    158   180
Sequence        TMHMM2.0        inside     181   200
Sequence        TMHMM2.0        TMhelix    201   223
Sequence        TMHMM2.0        outside    224   227
Sequence        TMHMM2.0        TMhelix    228   250
Sequence        TMHMM2.0        inside     251   274
```



## BioInformatics 10: microarray

Microarrays measure the activity (expression level) of the genes under varying conditions/ time points; expression level is estimated by measuring the amount of mRNA for that particular gene; a gene is active if it is being transcribed; more mRNA usually indicates more gene activity.

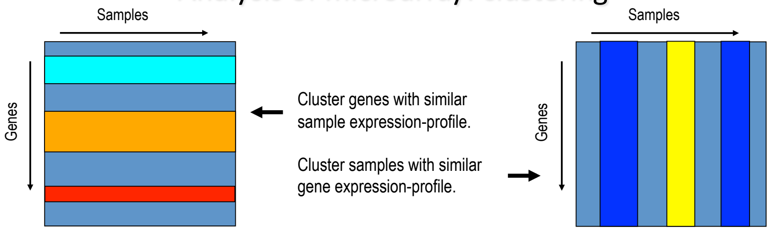


Intensity (expression level) of gene at measured time/condition (tumor and health) or different tissues, different patients

Millions of identical probes per feature (25 base-long single -strand DNA)

| Time: | Time X | Time Y | Time Z |
|---|---|---|---|
| Gene 1 | 10 | 8 | 10 |
| Gene 2 | 10 | 0 | 9 |
| Gene 3 | 4 | 8.6 | 3 |
| Gene 4 | 7 | 8 | 3 |
| Gene 5 | 1 | 2 | 3 |

## Analysis of microarray: clustering



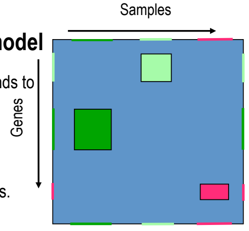
Cluster genes with similar sample expression-profile.

Cluster samples with similar gene expression-profile.

### Combination model

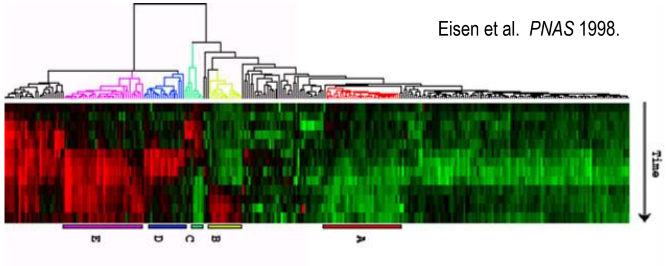
Each color corresponds to some "cause".

The cause affects a subset of genes in a subset of the samples.



Plot each measure as a point in N-dimensional space;
Make a distance matrix for the distance between every two gene points in the N-dimensional space;
Genes with a small distance share the same expression characteristics and might be functionally related or similar.
Clustering reveal groups of functionally related genes

## Clustering genes on expression profiles



Eisen et al. *PNAS* 1998.

Green = Expression level low with respect to reference sample.
Red = Expression level high with respect to reference sample.
Black = Expression level comparable to reference sample.

The columns are ordered such that similar expression profiles neighbor each other.

## K-Means Clustering Problem: Formulation

- **Input**: A set, ***V***, consisting of *n* points and a parameter *k*
- **Output**: A set ***X*** consisting of *k* points (*cluster centers*) that minimizes the squared error distortion *d(**V**,**X**)* over all possible choices of ***X***

## 1-Means Clustering Problem: an Easy Case

- **Input**: A set, ***V***, consisting of *n* points
- **Output**: A single points ***x*** (cluster center) that minimizes the squared error distortion *d(**V**,**x**)* over all possible choices of ***x***

1-Means Clustering problem is easy.
However, it becomes very difficult (NP-complete) for more than one center.
An efficient ***heuristic*** method for K-Means clustering is the Lloyd algorithm

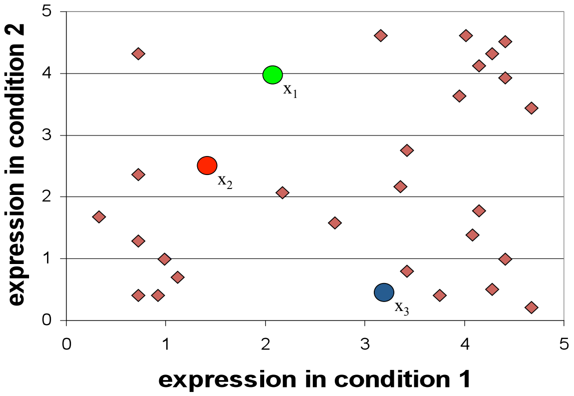
246

## K-Means Clustering: Lloyd Algorithm

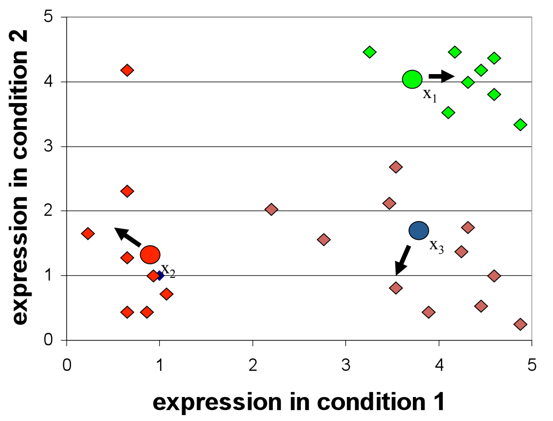
```
Lloyd Algorithm
    Arbitrarily assign the k cluster centers
    while the cluster centers keep changing
        Assign each data point to the cluster $C_i$
        corresponding to the closest cluster
        representative (center)  ($1 \le i \le k$)
        After the assignment of all data points,
        compute new cluster representatives
        according to the center of gravity of each
        cluster, that is, the new cluster
        representative is
              $\Sigma v \setminus |C|$  for all v in C   for every cluster C
```

*This may lead to merely a locally optimal clustering.

247



248



# Conservative K-Means Algorithm

- Lloyd algorithm is fast but in each iteration it moves many data points, not necessarily causing better convergence.
- A more conservative method would be to move one point at a time only if it improves the overall **clustering cost**
  - The smaller the clustering cost of a partition of data points is the better that clustering is
  - Different methods (e.g., the squared error distortion) can be used to measure this clustering cost

# K-Means "Greedy" Algorithm

```
ProgressiveGreedyK-Means(k)
Select an arbitrary partition P into k clusters
while forever
   bestChange ← 0
   for every cluster C
      for every element i not in C
         if moving i to cluster C reduces its clustering cost
            if (cost(P) – cost($P_{i \to C}$) > bestChange
             bestChange ← cost(P) – cost($P_{i \to C}$)
             $i^* \leftarrow I$
             $C^* \leftarrow C$
   if bestChange > 0
      Change partition P by moving $i^*$ to $C^*$
   else
      return P
```

# Squared Error Distortion

- Given a data point ***v*** and a set of points ***X***, define the **distance** from ***v*** to ***X***

$$d(v, \boldsymbol{X})$$

as the (Eucledian) distance from ***v*** to the ***closest*** point from ***X***.

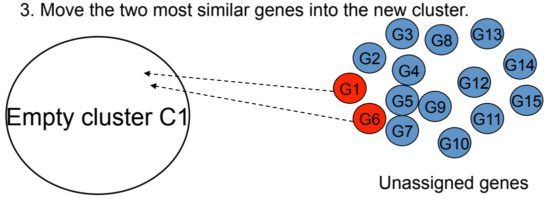
- Given a set of *n* data points $V=\{v_1 \ldots v_n\}$ and a set of *k* points ***X***, define the **Squared Error Distortion**

$$d(\boldsymbol{V},\boldsymbol{X}) = \sum d(v_i, \boldsymbol{X})^2 \;/\; n \qquad 1 \leq i \leq n$$

## Clustering Affinity Search Technique (CAST)-1

*Affinity = a measure of similarity between a gene, and all the genes in a cluster.*
*Threshold affinity = user-specified criterion for retaining a gene in a cluster, defined as %age of maximum affinity at that point*

1. Create a new empty cluster C1.
2. Set initial affinity of all genes to zero
3. Move the two most similar genes into the new cluster.



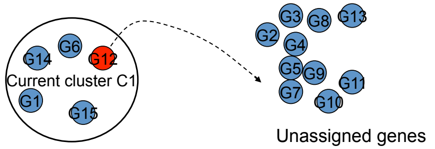
4. Update the affinities of all the genes (new affinity of a gene = its previous affinity + its similarity to the gene(s) newly added to the cluster C1)

**ADD GENES:**

5. While there exists an unassigned gene whose affinity to the cluster C1 exceeds the user-specified threshold affinity, pick the unassigned gene whose affinity is the highest, and add it to cluster C1. Update the affinities of all the genes accordingly.

**REMOVE GENES:** CAST – 2

6. When there are no more unassigned high-affinity genes, check to see if cluster C1 contains any elements whose affinity is lower than the current threshold. If so, remove the lowest-affinity gene from C1. Update the affinities of all genes by subtracting from each gene's affinity, its similarity to the removed gene.
7. Repeat step 6 while C1 contains a low-affinity gene



8. Repeat steps 5-7 as long as changes occur to the cluster C1.
9. Form a new cluster with the genes that were not assigned to cluster C1, repeating steps 1-8.
10. Keep forming new clusters following steps 1-9, until all genes have been assigned to a cluster

## Markov clustering algorithm

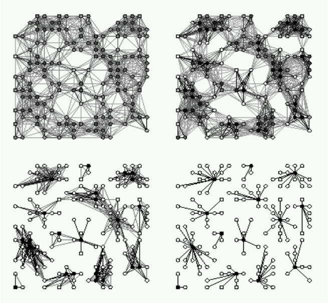
We take a random walk on the graph described by the similarity matrix, but after each step we weaken the links between distant nodes and strengthen the links between nearby nodes.

Unlike most clustering algorithms, the MCL does not require the number of expected clusters to be specified beforehand.

The basic idea underlying the algorithm is that dense clusters correspond to regions with a larger number of paths.

A random walk has a higher probability to stay inside the cluster than to leave it soon. The crucial point lies in boosting this effect by an iterative alternation of expansion and inflation steps.

The algorithm iterates three steps.
Given a network with $n$ vertexes, it takes the corresponding $n \times n$ adjacency matrix $A$ and normalises each column to obtain a stochastic matrix $M$. It takes the $k_{th}$ power $M^{k}$ of this matrix (expansion) and then the $r_{th}$ power $m^{r}(_{ij})$ of every element (inflation).
The expansion parameter $k$ is often taken equal to 2, while the granularity of the clustering is controlled by tuning the inflation parameter $r$.

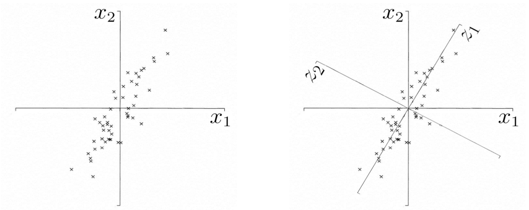


Graphic from van Dongen, 2000

Principle Components Analysis (PCA)

A sample of n observations in the 2-D space $X=(X_1,X_2)$

Goal: to account for the variation in a sample in as few variables as possible, to some accuracy



- the 1st PC $Z_1$ is a minimum distance fit to a line X in space
- the 2nd PC $Z_2$ is a minimum distance fit to a line in the plane perpendicular to the 1st PC

PCAs are a series of linear least squares fits to a sample, each orthogonal to all the previous.

# Principle Components Analysis (PCA)

- PCA seeks for a linear projection that best describes the data in a least mean squares sense
- Finds a set of principle components (PCs)
  - A PC defines a projection that encapsulates the maximum amount of variation in a dataset
  - Each PC is orthogonal to all other PCs
- Reduce dimensionality by picking the most informative PCs
  - Namely, for reducing from dimension d to dimension d', pick the d' most informative PCs

# PCA - Steps

Input: a dataset $S=\{s^1,\ldots,s^n\},\quad s^i=\left\langle s_1^i,\ldots,s_d^i\right\rangle$

- Subtract the mean from each dimension
- Compute the covariance matrix ∑ for the d dimensions
  - The covariance of two variables X and Y:

$$\operatorname{cov}(X,Y)=\sum_{i=1}^{n}\frac{(X_i-\overline{X})\cdot(Y_i-\overline{Y})}{(n-1)}$$

  - The covariance matrix: $\sum(X,Y)=\sum(Y,X)=\operatorname{cov}(X,Y)$

# PCA – Steps (cont.)

- Compute the eigenvectors and eigenvalues of the covariance matrix
- Choose the most informative PCs, construct a feature vector
  - Eigenvectors with highest eigenvalues carry the most information
  - Feature vector is simply the combination of all eigenvectors chosen

FeatureVector = ($\text{eig}_1$, $\text{eig}_2$, …, $\text{eig}_{d'}$)

- Transform dataset to the new axis system
  - For s∈S: $s'=FeatureVector^T\times s=\begin{bmatrix}eig_1\\eig_2\\\vdots\\eig_{d'}\end{bmatrix}\times\begin{bmatrix}s_1\\s_2\\\vdots\\s_d\end{bmatrix}$

## When Things Get Messy…

- PCA is fine when initial dimension is not too big
  - Space and time complexity are of O(d$^2$) - size of covariance matrix
- Otherwise – we have a problem…
  - E.g. when d=10$^4$ $\Rightarrow$ time/space complexity is O(10$^8$)…
- Luckily an alternative exists: SVD

## Eigengenes, Eigenarrays and SVD

- The idea:
  - Use the singular value decomposition (SVD) theorem for transforming the dataset from the gene/array space to the eigengene/eigenarray space
- Eigengenes, eigenarrays and eigenvalues:
  - Each dimension is represented by an eigengene/eigenarray/eigenvalue triplet
  - Eigenvalues are used for ranking dimensions

## Singular Value Decomposition (SVD)

- Theorem: if E is a real M by N matrix, then there exist orthogonal matrices

  $$U = [u^1,\ldots,u^M] \in \Re^{\mathrm{MxM}} \quad \text{and} \quad V = [v^1,\ldots,v^N] \in \Re^{\mathrm{NxN}}$$

  s.t.

  $$E = U \cdot W \cdot V^T$$

  Where

  $$W = diag(\sigma_1,\ldots,\sigma_p)$$

  and

  $$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0, \qquad p = \min(m,n)$$

## SVD

- $\sigma_i$ is the ith singular value of E.
  $u_i$ and $v_i$ are the ith left singular vector and right singular vector of E, respectively.
- It holds that

  $$\left.\begin{aligned} E \cdot v^i &= \sigma_i \cdot u^i \\ E^T \cdot u^i &= \sigma_i \cdot v^i \end{aligned}\right\} i = 1:\min(M,N)$$

- Efficient algorithms for calculating the SVD exist

# Orthogonality of Decomposition

$$E = U \cdot W \cdot V^T$$

$$V = [v^1, \ldots, v^N], \qquad v^i = \left\langle v_1^i, \ldots, v_N^i \right\rangle$$

$$U = [u^1, \ldots, u^M], \qquad u^i = \left\langle u_1^1, \ldots, u_M^1 \right\rangle$$

$$W = diag(\sigma_1, \ldots, \sigma_p)$$

$$\begin{bmatrix} e_{11} & e_{12} & \cdots & & e_{1N} \\ e_{21} & \ddots & & & \\ \vdots & & & & \vdots \\ e_{M1} & & & \cdots & e_{MN} \end{bmatrix} = \begin{bmatrix} u_1^1 & u_1^2 & \cdots & u_1^M \\ u_2^1 & \ddots & & \\ \vdots & & & \vdots \\ u_M^1 & & \cdots & u_M^M \end{bmatrix} \times \begin{bmatrix} \sigma_{11} & 0 & \cdots & & 0 \\ 0 & \sigma_{22} & & & \\ \vdots & & \ddots & & \vdots \\ 0 & & & \cdots & 0 \end{bmatrix} \times \begin{bmatrix} v_1^1 & v_2^1 & \cdots & & v_N^1 \\ v_1^2 & \ddots & & & \vdots \\ \vdots & & & & \\ & & & \ddots & \vdots \\ v_1^N & \cdots & & \cdots & v_N^N \end{bmatrix}$$

# Orthogonality of Decomposition

$$U \cdot W = \begin{bmatrix} \sigma_{11}u_1^1 & \sigma_{22}u_1^2 & \cdots & \sigma_{MM}u_1^M & \cdots & 0 \\ \sigma_{11}u_2^1 & \sigma_{22}u_2^2 & & & & \\ \vdots & & \ddots & & & \vdots \\ \sigma_{11}u_M^1 & & & \sigma_{MM}u_M^M & \cdots & 0 \end{bmatrix} \qquad V^T = \begin{bmatrix} v_1^1 & v_2^1 & \cdots & & v_N^1 \\ v_1^2 & \ddots & & & \vdots \\ \vdots & & & & \\ & & & \ddots & \vdots \\ v_1^N & \cdots & & \cdots & v_N^N \end{bmatrix}$$

$$U \cdot W \cdot V^T = \begin{bmatrix} e_{11} & e_{12} & \cdots & & e_{1N} \\ e_{21} & \ddots & & & \\ \vdots & & e_{ij} & & \vdots \\ e_{M1} & & & \cdots & e_{MN} \end{bmatrix} \qquad e_{ij} = \sum_{k=1}^{p} \sigma_k \cdot u_i^k \cdot v_j^k$$

$$\Rightarrow E = \sum_{k=1}^{p} \sigma_k \cdot u^k \cdot v^{k^T}$$

# SVD and Microarray analysis

- Reduction from the N genes x M arrays to p eigengenes x p eigenarrays space
  - W is the eigenexpression matrix
  - U represents the expression of genes over eigenarrays
  - V represents the expression of eigengenes over arrays
- The “fraction of eigenexpression”:

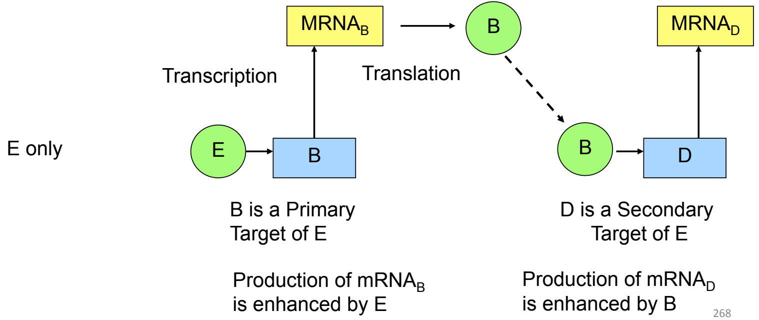
$$p_i = \sigma_i \Big/ \sum_{k=1}^{p} \sigma_k^2$$

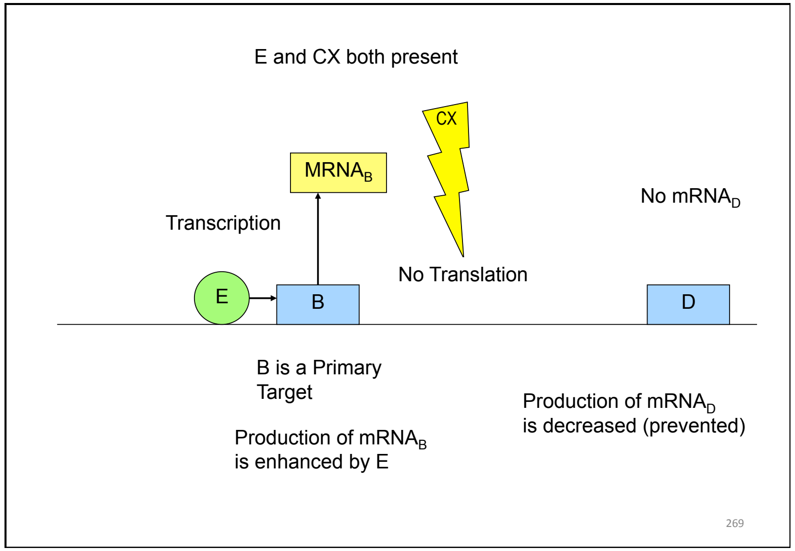
- “Shannon entropy” of the dataset:

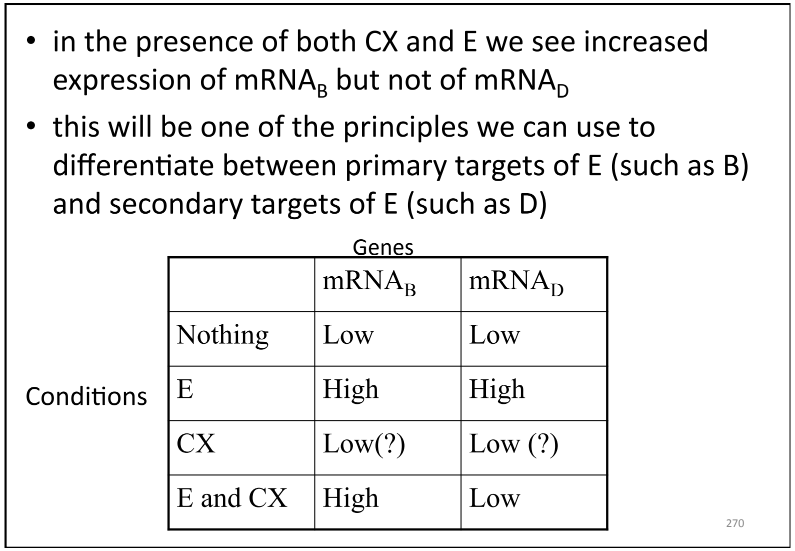
$$0 \le d = \frac{-1}{\log(p)} \sum_{k=1}^{p} p_k \log(p_k) \le 1$$

# BioInformatics 11: genetic networks

- assume that there are two related genes, B and D
- neither is expressed initially, but E causes B to be expressed and this in turn causes D to be expressed
- the addition of CX by itself may not affect expression of either B or D
- both CX and E will have elevated levels of mRNA$_B$ and low levels of mRNA$_D$

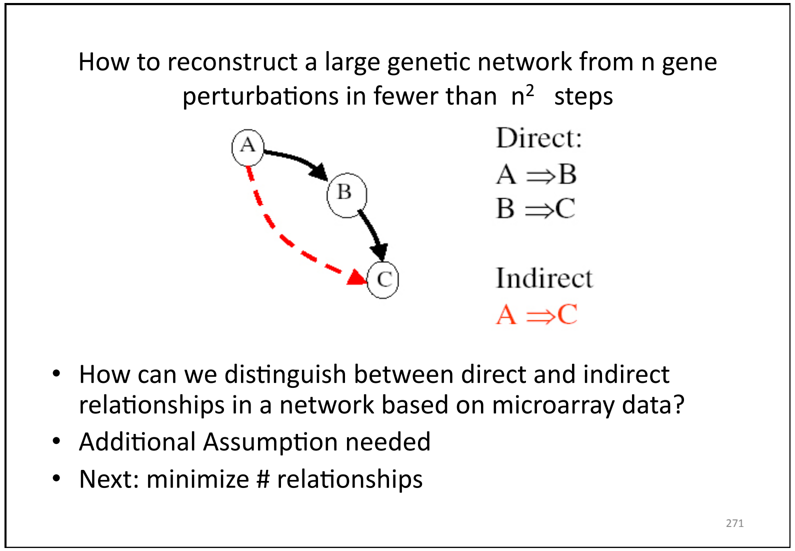

MRNA$_B$
B
MRNA$_D$
Transcription
Translation
E only
E
B
B
D
B is a Primary Target of E
D is a Secondary Target of E
Production of mRNA$_B$ is enhanced by E
Production of mRNA$_D$ is enhanced by B





- in the presence of both CX and E we see increased expression of mRNA$_B$ but not of mRNA$_D$
- this will be one of the principles we can use to differentiate between primary targets of E (such as B) and secondary targets of E (such as D)

| | | Genes | |
|---|---|---|---|
| | | mRNA$_B$ | mRNA$_D$ |
| Conditions | Nothing | Low | Low |
| | E | High | High |
| | CX | Low(?) | Low (?) |
| | E and CX | High | Low |

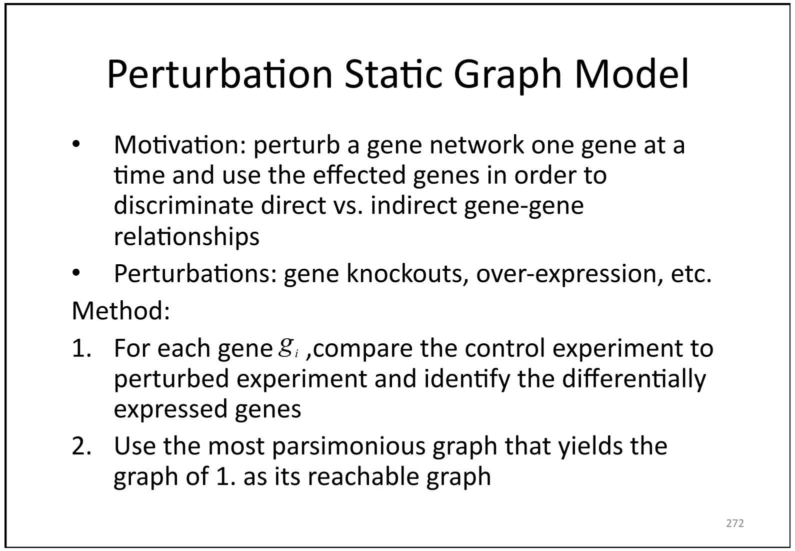


How to reconstruct a large genetic network from n gene perturbations in fewer than n$^2$ steps

Direct:
$A \Rightarrow B$
$B \Rightarrow C$

Indirect
$A \Rightarrow C$

- How can we distinguish between direct and indirect relationships in a network based on microarray data?
- Additional Assumption needed
- Next: minimize # relationships



## Perturbation Static Graph Model

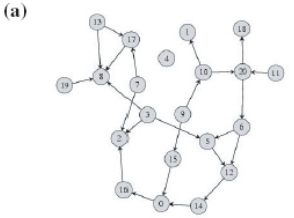
- Motivation: perturb a gene network one gene at a time and use the effected genes in order to discriminate direct vs. indirect gene-gene relationships
- Perturbations: gene knockouts, over-expression, etc.

Method:

1. For each gene $g_i$ ,compare the control experiment to perturbed experiment and identify the differentially expressed genes
2. Use the most parsimonious graph that yields the graph of 1. as its reachable graph

## An example

- (a) gene network
- (b) adjacency list
- (c) accessibility list
- Goal: (c) -> (a)

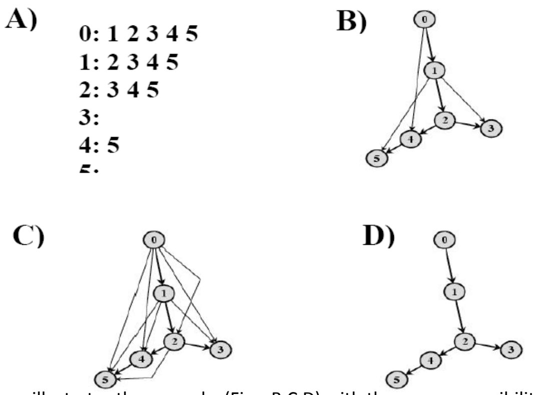


(b)

| | |
|---|---|
| 0: | 16 |
| 1: | |
| 2: | |
| 3: | 2 5 8 |
| 4: | |
| 5: | 12 |
| 6: | 5 12 |
| 7: | 2 17 |
| 8: | |
| 9: | 10 15 |
| 10: | 1 20 |
| 11: | 20 |
| 12: | 14 |
| 13: | 8 17 |
| 14: | 0 |
| 15: | 0 |
| 16: | 2 |
| 17: | 8 |
| 18: | |
| 19: | 8 |
| 20: | 6 18 |

(c)

| | |
|---|---|
| 0: | 2 16 |
| 1: | |
| 2: | |
| 3: | 0 2 5 8 12 14 16 |
| 4: | |
| 5: | 0 2 12 14 16 |
| 6: | 0 2 5 12 14 16 |
| 7: | 2 8 17 |
| 8: | |
| 9: | 0 1 2 5 6 10 12 14 15 16 18 20 |
| 10: | 0 1 2 5 6 12 14 16 18 20 |
| 11: | 0 2 5 6 12 14 16 18 20 |
| 12: | 0 2 14 16 |
| 13: | 8 17 |
| 14: | 0 2 16 |
| 15: | 0 2 16 |
| 16: | 2 |
| 17: | 8 |
| 18: | |
| 19: | 8 |
| 20: | 0 2 5 6 12 14 16 18 |



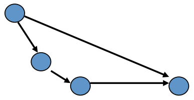
0: 1 2 3 4 5
1: 2 3 4 5
2: 3 4 5
3:
4: 5
5:

The figure illustrates three graphs (Figs. B,C,D) with the same accessibility list *Acc* (Fig. A). There is one graph (Fig. D) that has *Acc* as its accessibility list and is simpler than all other graphs, in the sense that it has fewer edges. Let's call *Gpars* the *most parsimonious network* compatible with *Acc*.

## Algorithm

- Step1: Graphs without cycles only (acyclic directed graph)
- Step2: Graphs with cycles

- Step 1: Shortcut:



- A shortcut-free graph compatible with an accessibility list is a unique graph with the fewest edges among all graphs compatible with the accessibility list, i.e, a shortcut-free graph is the most parsimonious graph.

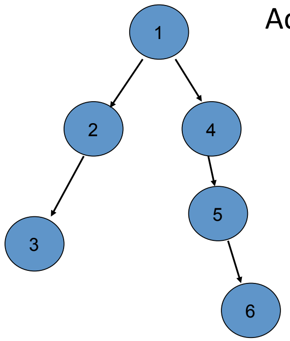
## Step1

- A theorem: Let Acc(G) be the accessibility list and Adj (G) be the adjacency list at an acyclic directed graph, its most $G_{pars}$ parsimonious graph, and V( ) the set $G_{pars}$ of all nodes of . Then the $G_{pars}$ following identity holds

$$\forall i \in V(G_{pars}) \quad Adj(i) = Acc(i) \setminus \bigcup_{j \in Acc(i)} Acc(j)$$

In words, for each node i the adjacency list Adj(i) of the most parsimonious genetic network is equal to the accessibility list Acc(i) after removal of all nodes that are accessible from any node in Acc(i).

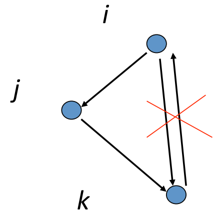
## An example



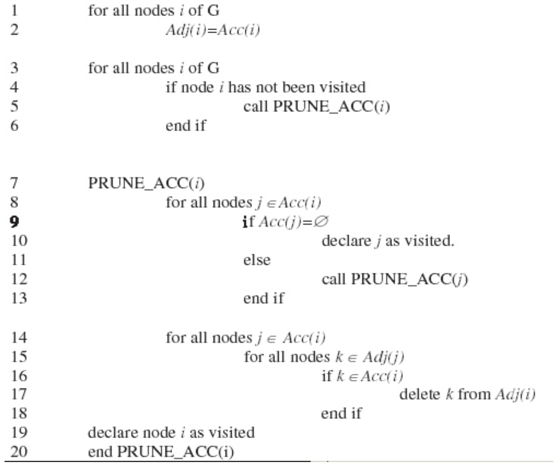
Adj(1) = Acc(1) -
(Acc(2) + Acc(3)
+ Acc(4) + Acc(5)
+Acc(6))
= (2,3,4,5,6) –
(3∪(5,6)∪ 6)
= (2,4)

## Step 1

- A Corollary: Let *i, j*, and *k* be any three pairwise different nodes of an acyclic directed shortcut-free graph G. If *j* is accessible from *i*, then no node *k* accessible from *j* is adjacent to *i*.



## The algorithm of step 1



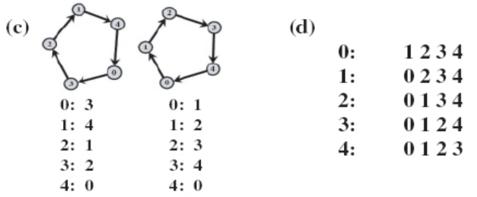
```
for all nodes $i$ of G
        $Adj(i)=Acc(i)$

for all nodes $i$ of G
        if node $i$ has not been visited
                call PRUNE_ACC($i$)
        end if


PRUNE_ACC($i$)
        for all nodes $j \in Acc(i)$
                if $Acc(j)=\varnothing$
                        declare $j$ as visited.
                else
                        call PRUNE_ACC($j$)
                end if

        for all nodes $j \in Acc(i)$
                for all nodes $k \in Adj(j)$
                        if $k \in Acc(i)$
                                delete $k$ from $Adj(i)$
                        end if
declare node $i$ as visited
end PRUNE_ACC(i)
```

## Step 2: How about graphs with cycles?



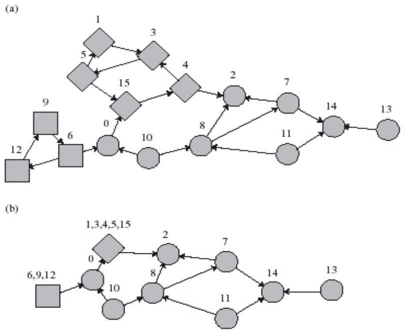
```
0: 3        0: 1
1: 4        1: 2
2: 1        2: 3
3: 2        3: 4
4: 0        4: 0
```

```
0:      1 2 3 4
1:      0 2 3 4
2:      0 1 3 4
3:      0 1 2 4
4:      0 1 2 3
```

- Two different cycles have the same accessibility list
- Perturbations of any gene in the cycle influences the activity of all other genes in the same cycle.
- Can’t decide a unique graph if cycle happens
- Not an algorithmic but an experimental limitation

# The algorithm of step 2

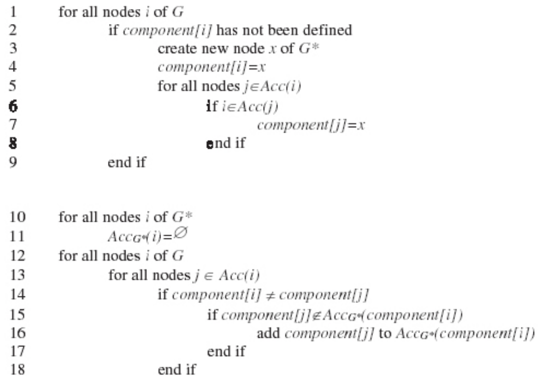
- Basic idea: Shrink each cycles (strongly connected components) into one node and apply the algorithm of step 1.



# The algorithm of step 2

- A corollary: Let *i* and *j* (*i* ≠ *j*) be two nodes of a directed graph *G*. *i* and *j* are in the same component iff *i*∈Acc(*j*) and *j*∈Acc(*i*).
- A graph after shrinking all the cycles into nodes is called a condensation graph.

# The algorithm of step 2



```
for all nodes $i$ of $G$
    if $component[i]$ has not been defined
        create new node $x$ of $G*$
        $component[i]=x$
        for all nodes $j \in Acc(i)$
            if $i \in Acc(j)$
                $component[j]=x$
            end if
    end if

for all nodes $i$ of $G*$
    $Acc_{G*}(i)=\varnothing$
for all nodes $i$ of $G$
    for all nodes $j \in Acc(i)$
        if $component[i] \neq component[j]$
            if $component[j] \notin Acc_{G*}(component[i])$
                add $component[j]$ to $Acc_{G*}(component[i])$
            end if
        end if
```

# Missing genes and messy data

- Some genes are difficult to perturb
- Problem: some information is missing for certain genes. How well does the algorithm perform in such cases?
- Simulation: Randomly generate graphs with pre-specified nodes and edges. Then eliminate pre-specified fraction of nodes from the accessibility list. Apply the algorithm to both graphs without elimination and with elimination.

## Limitation of the algorithm

- Unable to resolve cycled graphs
- Require more data than conventional methods using gene expression correlations.
- There are many networks consistent with the given accessibility list. The algorithm construct the most parsimonious one.
- The same problem was proposed around 1980 which is called “transitive reduction”.
- The transitive reduction of a directed graph G is the directed graph G' with the smallest number of edges such for every path between vertices in G, G' has a path between those vertices.
- An O(V) algorithm for computing transitive reduction of a planar acyclic digraph was proposed by Sukhamay Kundu. (V is the number of nodes in G)

## BioInformatics lecture 12

System Biology

1. Large scale integration of information on molecules, genes, cell, tissue, organ, organism, health
2. Markup language

   -- development of SBML (Systems Biology Markup Language) for representing biochemical networks and CellML for electrophysiology, mechanics, energetics and general pathway. SBML is an XML-based markup language for describing the biochemical network models that arise in Systems Biology.
3. Computational models

   -- development of models that are “anatomically based” and “biophysically based” to link gene, protein, cell, tissue ,organ and whole body systems physiology.

   Methodologies: differential equations and stochastic algorithms

### The Gillespie algorithm

A **reaction rate** ***w**$_i$* is associated to each reaction step. These probabilites are related to the kinetics constants.

**Initial number** of molecules of each species are specified.

The **time interval** is computed stochastically according the reation rates.

At each time interval, the **reaction** that occurs is chosen randomly according to the probabilities *w$_i$* and both the number of molecules and the reaction rates are updated.

$$A \xrightarrow{w_1} B$$

$$B + C \xrightarrow{w_2} D$$

$$D \xrightarrow{w_3} E + F$$

...

## Gillespie algorithm

Probability that reaction *r* occurs

$$P_r = \frac{w_r}{\sum_{i=1}^{R} w_i}$$

Reaction *r* occurs if

$$P_{r-1} < z_1 \leq P_{r-1} + P_r$$

Time step to the next reaction

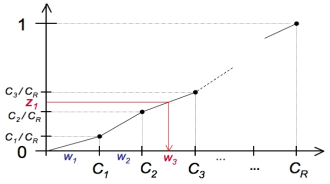
$$\Delta t = \frac{1}{\sum_{i=1}^{R} w_i} ln \frac{1}{z_2}$$

$$A \xrightarrow{w_1} B$$

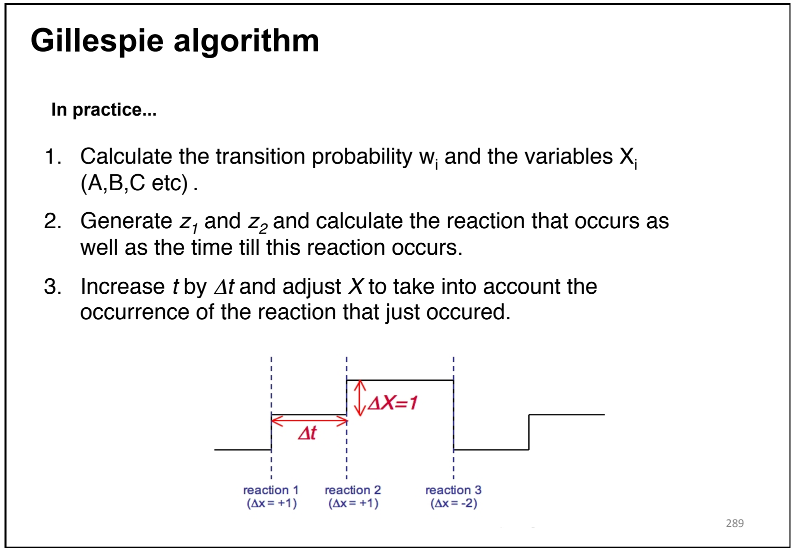
$$B + C \xrightarrow{w_2} D$$

$$D \xrightarrow{w_3} E + F$$

...



Gillespie D.T. (1977) Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.* 81: 2340-2361.
**Gillespie D.T.**, (1976) A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *J. Comp. Phys.*, 22: 403-434.



## Gillespie algorithm

**In practice...**

1. Calculate the transition probability $w_i$ and the variables $X_i$ (A,B,C etc) .
2. Generate $z_1$ and $z_2$ and calculate the reaction that occurs as well as the time till this reaction occurs.
3. Increase $t$ by $\Delta t$ and adjust $X$ to take into account the occurrence of the reaction that just occured.

$\Delta X=1$
$\Delta t$
reaction 1
($\Delta x = +1$)
reaction 2
($\Delta x = +1$)
reaction 3
($\Delta x = -2$)