

# ***Bioinformatics***



**Computer Laboratory**

---

**Computer Science Tripos, Part II  
Michaelmas Term 2013**

**Dr Pietro Liò**

**Copies of slides**

**William Gates Building  
15 JJ Thomson Avenue  
Cambridge  
CB3 0FD**

<http://www.cl.cam.ac.uk/>

## Objectives

- ▶ The course focuses on algorithms used in bioinformatics
- ▶ The algorithms presented in this course could be also applied in other data-rich fields.
- ▶ At the end of the course the student should be able to describe the main aspects of the algorithms.
- ▶ The student should understand how bioinformatics combines biology and computing.
- ▶ The exam papers will not contain biological questions.
- ▶ References and links to additional material at the end of the lecture notes may help the students to understand better the applications of the algorithms (this is not essential to answer exam questions).

## Overview of the course

First we learn how to compare two and then one or more DNA sequences. Searching a database for nearly exact matches (using Blast algorithm) is the most important routine work in a Bioinformatics lab. The big efforts in sequencing human genomes and also single cell genomes will require new algorithms. We will show some examples of the first wave. We learn how to build trees to study sequences relationship and how to cluster biological data using K-means and the Markov clustering algorithms. We use hidden Markov models to predict exon/intron arrangements in a gene or the structure of a membrane protein. Then we look at an algorithm to reconstruct genetic network (Wagner algorithm). A set of biochemical reactions could be simulated using the Gillespie algorithm. Material and figure acknowledgments at the end of the Notes and during the lectures.

## Topics and List of algorithms

- ▶ Basic concepts in genetics.
- ▶ Dynamic programming (Longest Common Subsequence, Needleman-Wunsch, Smith-Waterman, Hirschberg, Nussinov).
- ▶ Progressive alignment (Clustal).
- ▶ Homology database search (Blast, Patternhunter).
- ▶ Next Generation sequencing (De Bruijn graph, Burrows-Wheeler transform)
- ▶ Phylogeny - parsimony-based - (Fitch, Sankoff).
- ▶ Phylogeny - distance based - (UPGMA, Neighbor Joining).
- ▶ Clustering (K-means, Markov Clustering)
- ▶ Hidden Markov Models applications in Bioinformatics (Genescan, TMHMM).
- ▶ Pattern search in sequences (Gibbs sampling).
- ▶ Biological Networks reconstruction (Wagner) and simulation (Gillespie).



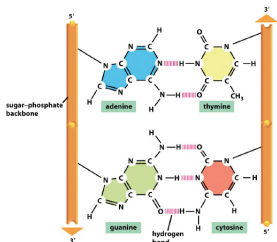
## Basic concepts in genetics

DNA, RNA, protein, the genetic code, the gene, exons, introns, the genome, the cell

## Basic concepts in genetics

DNA could be thought as a string of symbols from a 4-letter (bases) alphabet, A (adenine), T (thymine), C (cytosine) and G (guanine). In the double helix A pairs with T, C with G (so only the sequence of one filament is vital to keep). A gene is a string of DNA that contains information for a specific cell function. The Genome is the entire DNA in a cell nucleus.

RNA is same as DNA but T is replaced by U (uracil); proteins are strings of amino acids from an alphabet of 20. The proteins have a 3D shape that could be described by a 3 dimensional graph. The genetic code is a map between 61 triplets of DNA bases and 20 amino acid.



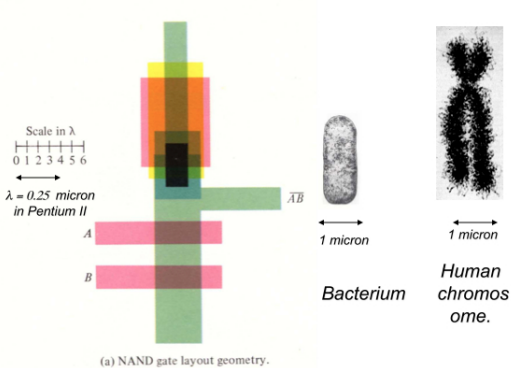
**A to T**  
**G to C**



In RNA (different five carbon sugar in the backbone) uracil is used instead of thymine

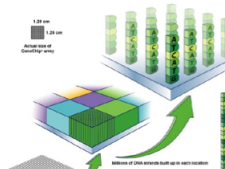
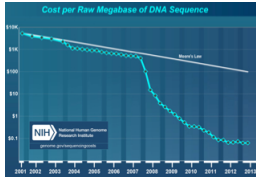
# DNA and chromosomes

In eukaryotes the genetic information is distributed over different DNA molecules. A human cell contains 24 different such chromosomes. If all DNA of a human cell would be laid out end-to-end it would reach approximately 2 meters. The nucleus however measures only  $6\mu\text{m}$ . Equivalent of packing 40 km of fine thread into a tennis ball with a compression ratio of 10000.

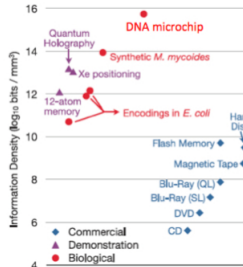


## Reading and writing DNA is cheap and effective

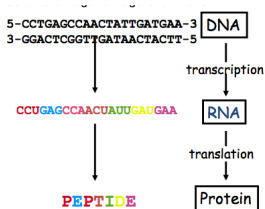
Sequencing costs have plummeted from 2.7 billion dollars for the first human genome in 2003 down to 1000 dollars today; Base errors is now 1 in 10 million. An average size book could be converted into thousands DNA sequences each encoding data block, an address specifying the location of the data block in the bit stream, and flanking common sequences to facilitate wet lab procedures to enable the reading. This library could be synthesized by ink-jet printed in high-fidelity DNA microchips.

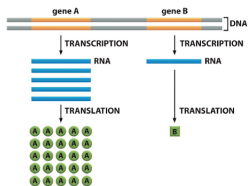
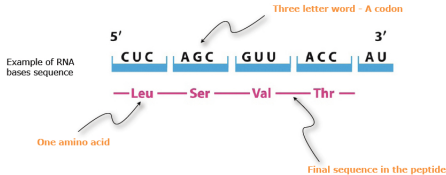


Left: DNA reading; bottom left: use of DNA sequences placed in DNA chip to write DNA or other information. Bottom right: advantage of using DNA chips.



DNA makes RNA makes proteins (the 3D graph below); given the pairing rule in a DNA double strands molecule, all the information is in each single strand. The RNA is termed mRNA and is translated by triplet of bases into a chain of amino acids (the protein).





**Figure:** The central dogma of molecular biology is that DNA is transcribed to RNA which is translated to protein. The amount of RNA depends on gene activity which is influenced by other proteins binding before the start of the gene; different tissues contain cells with different amount of RNA for each gene

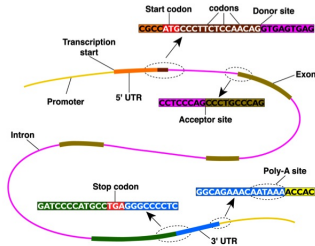
	AGA									UUA
	AGG									UUG
GCA	CGA						GGA		AUA	CUA
GCC	CGC						GGC	CAC	AUC	CUC
GCG	CGG	GAC	AAC	UGC	GAA	CAA	GGG	CAU	AUC	CUG
GCU	CGU	GAU	AAU	UGU	GAG	CAG	GGU	CAU	AUU	CUU
<b>Ala</b>	<b>Arg</b>	<b>Asp</b>	<b>Asn</b>	<b>Cys</b>	<b>Glu</b>	<b>Gln</b>	<b>Gly</b>	<b>His</b>	<b>Ile</b>	<b>Leu</b>
<b>A</b>	<b>R</b>	<b>D</b>	<b>N</b>	<b>C</b>	<b>E</b>	<b>Q</b>	<b>G</b>	<b>H</b>	<b>I</b>	<b>L</b>

				AGC						
				AGU						
			CCA	UCA	ACA			GUA		UAA
AAA			CCC	UCC	ACC			GUC		UAG
AAG	AUG	UUC	CCG	UCG	ACG		UAC	GUG		UGA
		UUU	CCU	UCU	ACU	UGG	UAU	GUU		
<b>Lys</b>	<b>Met</b>	<b>Phe</b>	<b>Pro</b>	<b>Ser</b>	<b>Thr</b>	<b>Trp</b>	<b>Tyr</b>	<b>Val</b>	<b>stop</b>	
<b>K</b>	<b>M</b>	<b>F</b>	<b>P</b>	<b>S</b>	<b>T</b>	<b>W</b>	<b>Y</b>	<b>V</b>		

**Figure:** The genetic code provides the information for the translation of codons (triplets of bases, in black) into amino acids (single and triple letter code in red) that are chained together to form a protein; 61 codons code for 20 amino acids; 3 special codons say “stop”.

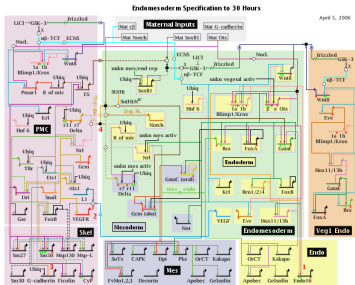
## The structure of a human gene

A gene starts with the promoter region, which is followed by a transcribed but non-coding region called 5' untranslated region (5' UTR). Then follows the initial exon which contains the start codon which is usually ATG. There is an alternating series of introns and internal exons, followed by the terminating exon, which contains the stop codon. It is followed by another non-coding region called the 3' UTR; at the end there is a polyadenylation (polyA) signal, i.e. a repetition of Adenine (example AAAAA). The intron/exon and exon/intron boundaries are conserved short sequences and called the acceptor and donor sites.





# Genes and proteins make networks that look like digital circuits



April 5, 2006

Ubiquitous; Mat = maternal; act = activator; rep = repressor; wks = unknown; Nucl. = nuclear;  $\beta$  =  $\beta$ -catenin source;  $\beta$ -TCF = mechanical  $\beta$ -catenin-Tcf; ES = early signal; ECNS = early cytoplasmic recombination system; Zyg. N. = zygotic notch

Copyright © 2001–2006 Harriet Borker and Eric Davidson

[discussions](#) [view source](#) [history](#)

## Registry of Standard Biological Parts

[Log in](#)

---

**Catalog**

[← Back to Registry](#)

- Browse parts by type - [click on by type](#)
- Browse parts and devices by function - [by chassis](#) - [by standard](#) - [or by contributor](#)
- Browse chassis
- Browse user-supplied catalog pages - these pages have not undergone curation by the Registry but have been made by the Registry user community. Please feel free to add new catalog pages to this section.

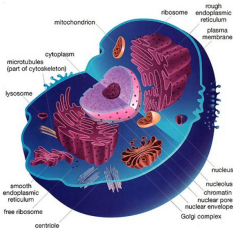
**Browse parts by type**

**Coding**

- ▶ **Promoters (7)**: A promoter is a DNA sequence that tends to recruit transcriptional machinery and lead to transcription of the downstream DNA sequence.
- ▶ **Ribosome Binding Sites (7)**: A ribosome binding site (RBS) is an RNA sequence found in mRNA to which ribosomes can bind and initiate translation.
- ▶ **Protein domains (7)**: Protein domains are portions of proteins cloned in frame with other protein domains to make up a protein coding sequence. Some protein domains might change the protein's location, alter its degradation rate, target the protein for cleavage, or enable it to be readily purified.
- ▶ **Protein coding sequences (7)**: Protein coding sequences encode the amino acid sequence of a particular protein. Note that some protein coding sequences only encode a protein domain or half a protein. Others encode a full-length protein from start codon to stop codon. Coding sequences for gene expression reporters such as LacZ and GFP are also included here.
- ▶ **Translational units (7)**: Translational units are composed of a ribosome binding site and a protein coding sequence. They begin at the site of translational initiation, the RBS, and end at the site of translational termination, the stop codon.
- ▶ **Terminators (7)**: A terminator is an RNA sequence that usually occurs at the end of a gene or open mRNA and causes transcription to stop.
- ▶ **DNA (7)**: DNA parts provide functionality to the DNA itself. DNA parts include cloning sites, scars, primer binding sites, spacers, recombination sites, conjugative transfer elements, transposons, origins, and aptamers.
- ▶ **Plasmid backbones (7)**: A plasmid is a circular, double-stranded DNA molecule typically containing a few thousand base pairs that replicate within the cell independently of the chromosomal DNA. A plasmid backbone is defined as the plasmid sequence beginning with the *NotI* restriction site, including the replication origin and antibiotic resistance cassette, and ending with the *BlnI* restriction site.

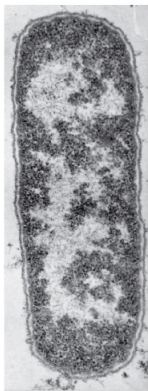
# Gene networks make cells make tissue

Top: a human cell (it measures  $10\mu m$  across); bottom: detail of a stomach tissue



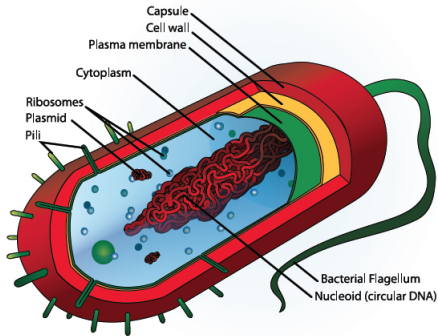
Lumen of stomach	Cell Types	Substance Secreted
	Mucous neck cell	Mucus (protects lining) Bicarbonate
	Parietal cells	Gastric acid (HCl) Intrinsic factor (Ca <sup>++</sup> absorption)
	Enterochromaffin-like cell	Histamine (stimulates acid)
	Chief cells	Pepsin(ogen) Gastric lipase
	D cells	Somatostatin (inhibits acid)
	G cells	Gastrin (stimulates acid)

A bacterium (for example *Escherichia coli*) measures about  $2\mu\text{m}$  in length, yet it contains about  $1,600\mu\text{m}$  ( $1.6\text{ mm}$ ) of circular double strands DNA ( $5 \times 10^6$  DNA bases in *E. coli*).

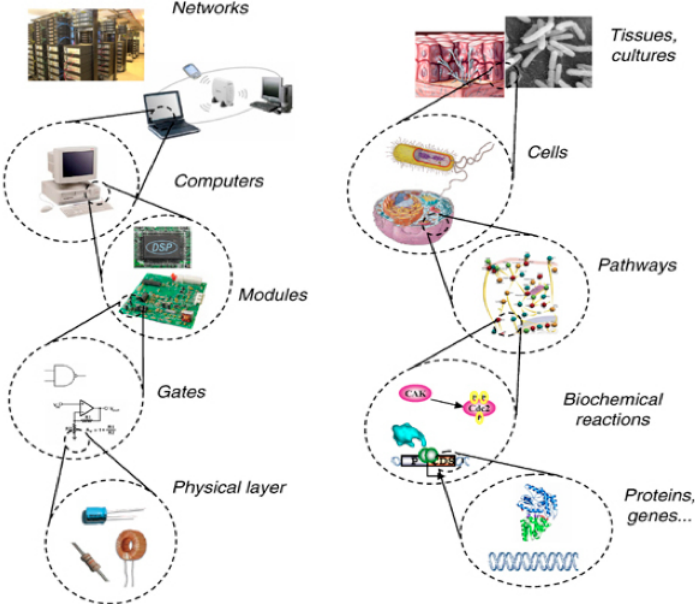


$1\mu\text{m}$

*Electron micrograph of E.coli  
(DNA in light stained region)*

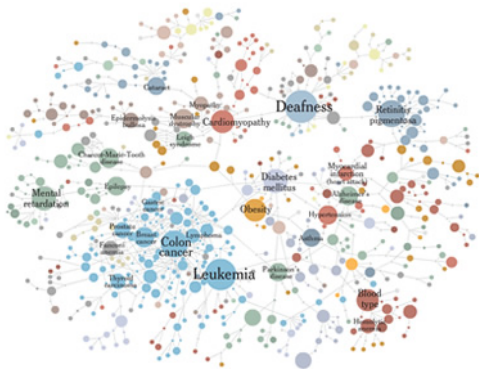


# Comparison between biological systems and system networks



## (previous figure)

from bottom: Molecules (for example genes and proteins), reactions (interactions between proteins and other molecules), pathways (composition of reactions, large scale interactions), cells (composition of pathways), tissues (composition of cells), organs (composition of tissues), organisms (compositions of organs?). Also diseases are linked because they share common genes (see below).



## Dynamic programming algorithms for sequence alignment

Longest common subsequence, Needleman-Wunsch, Smith Waterman, Affine gap, Hirschberg algorithm, RNA folding.

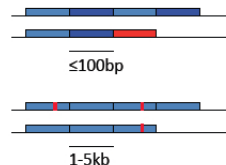
## Sequence Alignment: The Biological problem

- Single nucleotide polymorphisms (SNPs)
  - 1 every few hundred bp, mutation rate\*  $\approx 10^{-9}$
- Short indels (=insertion/deletion)
  - 1 every few kb, mutation rate v. variable
- Microsatellite (STR) repeat number
  - 1 every few kb, mutation rate  $\leq 10^{-3}$
- Minisatellites
  - 1 every few kb, mutation rate  $\leq 10^{-1}$
- Repeated genes
  - rRNA, histones
- Large deletions, duplications, inversions
  - Rare, e.g. Y chromosome

TGCATTGCGTAGGC  
TGCATTCCGTAGGC

TGCATT---TAGGC  
TGCATTCCGTAGGC

TGCTCATCATCATCAGC  
TGCTCATCA-----GC



**Figure:** Type and frequency of mutations in the human genome per generation; mutations changes DNA bases or rearrange DNA strings at different scales. In sequence alignment we compare sequences that are different because of mutations.

## Sequence Alignment

Alignment is a way of arranging two DNA or protein sequences to identify regions of similarity that are conserved among species. Each aligned sequence appears as a row within a matrix. Gaps are inserted between the residues (=amino acids) of each sequence so that identical or similar bases in different sequences are aligned in successive positions. Each gap spans one or more columns within the alignment matrix. Given two strings  $x = x_1, x_2, \dots, x_M$ ,  $y = y_1, y_2, \dots, y_N$ , an alignment is an assignment of gaps to positions  $0, \dots, M$  in  $x$ , and  $0, \dots, N$  in  $y$ , so as to line up each letter in one sequence with either a letter, or a gap in the other sequence.

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC  
TAGCTATCACGACCGCGGTTCGATTTGCCCGAC
```

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



Hamming distance  
always compares  
 $i^{\text{th}}$  letter of  $\mathbf{v}$  with  
 $i^{\text{th}}$  letter of  $\mathbf{w}$

$\mathbf{V} = \text{ATATATAT}$

$\mathbf{W} = \text{TATATATA}$

**Hamming distance:**

$$d(\mathbf{v}, \mathbf{w}) = 8$$

Computing Hamming distance  
is a **trivial** task

Just one shift  
Make it all line up  $\rightarrow$

Edit distance  
may compare  
 $i^{\text{th}}$  letter of  $\mathbf{v}$  with  
 $j^{\text{th}}$  letter of  $\mathbf{w}$

$\mathbf{V} = - \text{ATATATAT}$

$\mathbf{W} = \text{TATATATA}$

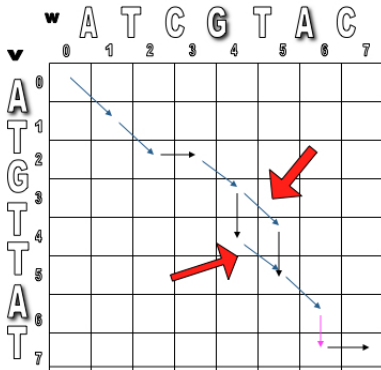
**Edit distance:**

$$d(\mathbf{v}, \mathbf{w}) = 2$$

Computing edit distance  
is a **non-trivial** task

**Figure:** The Hamming distance is a column by column number of mismatches; the Edit distance between two strings is the minimum number of operations (insertions, deletions, and substitutions) to transform one string into the other.

# Alignment as a Path in the Edit Graph



first alignment

0122345677

v= AT\_GTTAT\_

w= ATCGT\_A\_C

0123455667

second alignment

0122345677

v= AT\_GTTAT\_

w= ATCG\_TA\_C

0123445667

The score of the alignment paths are 5.

**Figure:** Create a matrix  $M$  with one sequence as row header and the other sequence as column header. Assign a 1 where the column and row site matches (diagonal segments), zero otherwise (horizontal or vertical segments); sequence alignment can be viewed as a Path in the Edit Graph. The edit graph is useful to introduce the dynamic programming technique.

## Dynamic programming, DP

DP is a method for reducing a complex problem to a set of identical sub-problems. The best solution to one sub-problem is independent from the best solution to the other sub-problems. **Recursion is a top-down mechanism**, we take a problem, split it up, and solve the smaller problems that are created; **DP is a bottom-up mechanism**: we solve all possible small problems and then combine them to obtain solutions for bigger problems. The reason that this may be better is that, using recursion, it is possible that we may solve the same small subproblem many times. Using DP, we solve it once. Consider the Fibonacci Series:  $F(n) = F(n - 1) + F(n - 2)$  where  $F(0) = 0$  and  $F(1) = 1$ . A recursive algorithm will take exponential time to find  $F(n)$  while a DP solution takes only  $n$  steps. A recursive algorithm is likely to be polynomial if the sum of the sizes of the subproblems is bounded by  $kn$ . If, however, the obvious division of a problem of size  $n$  results in  $n$  problems of size  $n-1$  then the recursive algorithm is likely to have exponential growth.

## The Longest Common Subsequence (LCS)

A subsequence of a string  $v$ , is a set of characters that appear in left-to-right order, but not necessarily consecutively. A common subsequence of two strings is a subsequence that appears in both strings. Substrings are consecutive parts of a string, while subsequences need not be.

A longest common subsequence is a common subsequence of maximal length. Example:

$v_1 = \langle A, C, B, D, E, G, C, E, D, B, G \rangle$  and

$v_2 = \langle B, E, G, C, F, E, U, B, K \rangle$

the LCS is  $\langle B, E, G, C, E, B \rangle$ .

With respect to DNA sequences:

$v_1 = \text{AAACCGTGAGTTATTCGTTCTAGAA}$

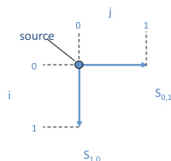
$v_2 = \text{CACCCCTAAGGTACCTTTGGTTC}$

LCS is  $\text{ACCTAGTACTTTG}$

## The Longest Common Subsequence (LCS)

- ▶ The LCS problem is the simplest form of sequence alignment; it allows only insertions and deletions (no mismatches).
- ▶ Given two sequences  $v = v_1 v_2 \dots v_m$  and  $w = w_1 w_2 \dots w_n$ . The LCS of  $v$  and  $w$  is a sequence of positions in  $v$ :  
 $1 < i_1 < i_2 < \dots < i_t < m$  and a sequence of positions in  $w$ :  
 $1 < j_1 < j_2 < \dots < j_t < n$  such that  $i_t$  letter of  $v$  equals to  $j_t$ -letter of  $w$  and  $t$  is maximal
- ▶ In the LCS problem, we score 1 for matches and 0 for indels; in DNA sequence alignment we will use different scores for match, mismatch and gap.

# The Longest Common Subsequence

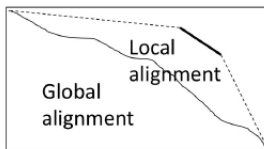


```

LCS(v,w)
for i ← 1 to n
  s_{i,0} ← 0
for j ← 1 to m
  s_{0,j} ← 0
for i ← 1 to n
  for j ← 1 to m
    s_{i,j} ← max {
      s_{i-1,j}
      s_{i,j-1}
      s_{i-1,j-1} + 1, if v_i = w_j
    }
    b_{i,j} ← {
      ↑ if s_{i,j} = s_{i-1,j}
      ← if s_{i,j} = s_{i,j-1}
      ↖ if s_{i,j} = s_{i-1,j-1} + 1
    }
  return (s_{n,m}, b)
  
```

**Figure:** It takes  $O(nm)$  time to fill in the  $n$  by  $m$  dynamic programming matrix. The pseudocode consists of two nested for loops to build up a  $n$  by  $m$  matrix.

- 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
          |||
aattgccgccgctcgttttcagCAGTTATGTCAGatc

```

**Figure:** The same sequences could be used in both alignments; we need to set the match score, the mismatch and gap penalties.

## Needleman-Wunsch algorithm (Global alignment)

1. Initialization (two sequences of length M and N).

- a.  $F(0, 0) = 0$
- b.  $F(0, j) = -j \times d$
- c.  $F(i, 0) = -i \times d$

2. Main Iteration. Filling-in partial alignments

For each  $i = 1 \dots M$

For each  $j = 1 \dots N$

$$F(i, j) = \max \begin{cases} F(i-1, j) - d & \text{[case 1]} \\ F(i, j-1) - d & \text{[case 2]} \\ F(i-1, j-1) + s(x_i, y_j) & \text{[case 3]} \end{cases}$$

$$\text{Ptr}(i, j) = \begin{cases} \text{UP,} & \text{if [case 1]} \\ \text{LEFT} & \text{if [case 2]} \\ \text{DIAG} & \text{if [case 3]} \end{cases}$$

3. Termination.  $F(M, N)$  is the optimal score, and from  $\text{Ptr}(M, N)$  can trace back optimal alignment



Match: 2; Gap: -1; Mismatch=-1

Gap penalty=-1; match=+2; mismatch=-1			A	C	G	C	T	G
		0	1	2	3	4	5	6
0	0	← -1	← -2	← -3	← -4	← -5	← -6	
C 1	-1	↑	-1	1				
A 2	-2	↑						
T 3	-3	↑						
G 4	-4	↑						
T 5	-5	↑						

AC  
-C

Match: 2; Gap: -1; Mismatch=-1

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

Match: 2; Gap: -1; Mismatch=-1

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

ACGCTG-  
-C-ATGT

Match: 2; Gap: -1; Mismatch=-1

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

ACGCTG-  
-CA-TGT

Match: 2; Gap: -1; Mismatch=-1

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

**-ACGCTG**  
**CATG-T-**

The choice of scores (match, gap and mismatch) depends on the data

		x			
			A	G	C
y		0	1	2	3
	Match 1				
Mismatch -1					
Gap -2					
	0	0	-2	-4	-6
	A 1	-2	1	-1	-3
AAAC	A 2	-4	-1	0	-2
A-GC	A 3	-6	-3	-2	-1
AAAC	C 4	-8	-5	-4	-1
-AGC					
AAAC					
AG-C					

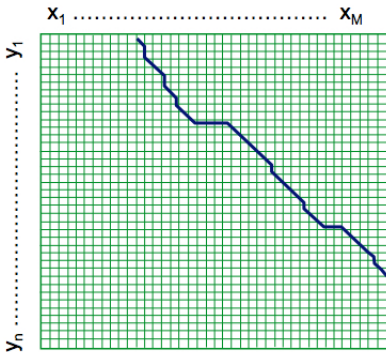
**Figure:** Given a  $m \times n$  matrix, the overall complexity of computing all sub-values is  $O(nm)$ . The final optimal score is the value at position  $n,m$ . In this case we align the sequences AGC and AAAC.

## How good is an alignment?

The score of an alignment is calculated by summing the rewarding scores for match columns that contain the same bases and the penalty scores for gaps and mismatch columns that contain different bases. A scoring scheme specifies the scores for matches and mismatches, which form the scoring matrix, and the scores for gaps, called the gap cost. **There are two types of alignments for sequence comparison.** Given a scoring scheme, calculating a **global alignment** is a kind of global optimization that forces the alignment to span the entire length of two query sequences, whereas **local alignments** identify regions of high similarity within two sequences.

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

-----CTATCACCTGACCTCCAGGCCGATGCCCCCTCCGGC  
 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: the Smith-Waterman algorithm

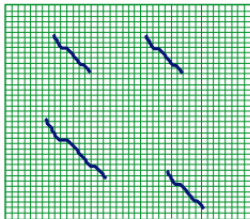
**Idea:** Ignore badly aligning regions: Modifications to Needleman-Wunsch

e.g.  $x = \text{aaaacccccgggg}$

$y = \text{cccgggaaccaacc}$

**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...

$$F_{\text{OPT}} = \max_{i,j} F(i, j)$$

2. If we want **all** local alignments **scoring > t**

For all  $i, j$  find  $F(i, j) > t$ , and trace back

# Example, Local alignment TAATA vs TACTAA

y = TAATA  
 x = TACTAA

y \ x	0	T	A	C	T	A	A
0	0	0	0	0	0	0	0
T 1	0	1	0	0	1	0	0
A 2	0	0	2	0	0	2	1
A 3	0	0	1	1	0	1	3
T 4	0	0	0	0	2	0	1
A 5	0	0	1	0	0	3	-1

y = TAATA  
 x = TACTAA

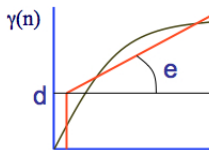
y \ x	0	T	A	C	T	A	A
0	0	0	0	0	0	0	0
T 1	0	1	0	0	1	0	0
A 2	0	0	2	0	0	2	1
A 3	0	0	1	1	0	1	3
T 4	0	0	0	0	2	0	1
A 5	0	0	1	0	0	3	-1

## Affine gap: two penalties for gap insertion

Insertions and deletions often occur in blocks longer than a single nucleotide. If there are many gaps we do not want to penalise too much; so we think at due penalties: one for the first gap (opening) and one, smaller, for the following required gaps.

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

|                      |  
gap                      gap  
open                      extend



To compute optimal alignment,

At position  $i, j$ , need to “remember” best score if gap is open  
best score if gap is not open

$F(i, j)$ : score of alignment  $x_1 \dots x_i$  to  $y_1 \dots y_j$   
if  $x_i$  aligns to  $y_j$

$G(i, j)$ : score if  $x_i$  aligns to a gap after  $y_j$

$H(i, j)$ : score if  $y_j$  aligns to a gap after  $x_i$

$V(i, j) =$  best score of alignment  $x_1 \dots x_i$  to  $y_1 \dots y_j$

## Affine gap: two penalties for gap insertion

Time complexity - As before  $O(nm)$ , as we only compute four matrices instead of one.

Space complexity: there's a need to save four matrices (for  $F$ ,  $G$ ,  $H$  and  $V$  respectively) during the computation. Hence,  $O(nm)$  space is needed, for the trivial implementation.

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

**Iteration:**

$$V(i, j) = \max\{ F(i, j), G(i, j), H(i, j) \}$$

$$F(i, j) = V(i - 1, j - 1) + s(x_i, y_j)$$

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

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

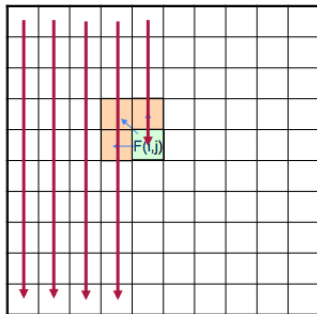
**Termination:** similar

## Space-Efficient Sequence Alignment, Hirschberg algorithm

In comparison of long DNA fragments, the limited resource in sequence alignment is not time but space. Hirschberg in 1975 proposed a divide-and-conquer approach that performs alignment in linear space for the expense of just doubling the computational time. The time complexity of the dynamic programming algorithm for sequence alignment is roughly the number of edges in the edit graph, i.e.,  $O(nm)$ . The space complexity is roughly the number of vertices in the edit graph, i.e.,  $O(nm)$ . However, if we only want to compute the score of the alignment (rather than the alignment itself), then the space can be reduced to just twice the number of vertices in a single column of the edit graph, i.e.,  $O(n)$ .

## Space-Efficient Sequence Alignment, Hirschberg algorithm

It is easy to compute  $F(M, N)$  in linear space



```
Allocate ( column[1] )
```

```
Allocate ( column[2] )
```

```
For i = 1...M
```

```
  If i > 1, then:
```

```
    Free( column[i - 2] )
```

```
    Allocate( column[ i ] )
```

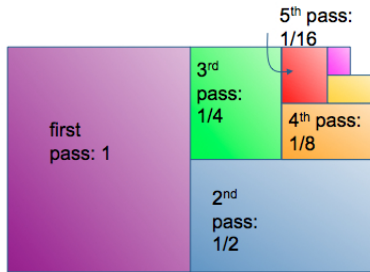
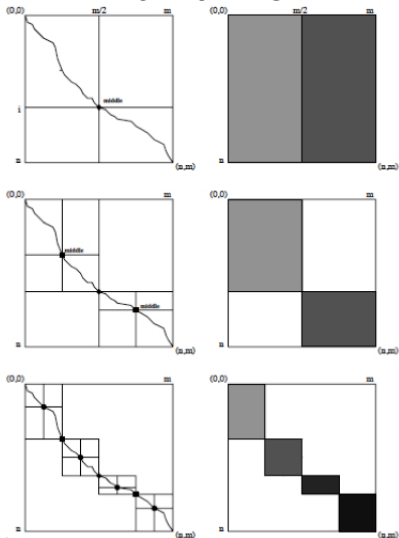
```
  For j = 1...N
```

```
    F(i, j) = ...
```

**Figure:** 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 have done using it

# Space-Efficient Sequence Alignment, Hirschberg algorithm

## Linear-Space Sequence Alignment



## Space-Efficient Sequence Alignment Hirschberg algorithm

The reduction comes from observation that the only values needed to compute the alignment scores  $s_{*j}$  (column  $j$ ) are the alignment scores  $s_{*,j-1}$  (column  $j-1$ ). Therefore, the alignment scores in the columns before  $j-1$  can be discarded while computing alignment scores for columns  $j, j+1, \dots$ . The longest path in the edit graph connects the start vertex  $(0,0)$  with the sink vertex  $(n, m)$  and passes through an (unknown) middle vertex  $(i, m/2)$  (assume for simplicity that  $m$  is even). Let's try to find its middle vertex instead of trying to find the entire longest path. This can be done in linear space by computing the scores  $s_{*,m/2}$ . (lengths of the longest paths from  $(0,0)$  to  $(i, m/2)$  for  $0 < i < n$ ) and the scores of the paths from  $(i, m/2)$  to  $(n,m)$ . The latter scores can be computed as the scores of the paths  $s_*^{reverse}$  from  $(n,m)$  to  $(i, m/2)$  in the reverse edit graph (i.e., the graph with the directions of all edges reversed). The value  $S_{i,m/2} + S_{i,m/2}^{reverse}$  is the length of the longest path from  $(0,0)$  to  $(n, m)$  passing through the vertex  $(i, m/2)$ . Therefore,  $\max_i [S_{i,m/2} + S_{i,m/2}^{reverse}]$  computes the length of the longest path and identifies a middle vertex.



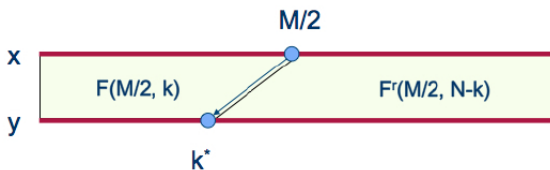
## Space-Efficient Sequence Alignment, Hirschberg algorithm

Computing these values requires the time equal to the area of the left rectangle (from column 1 to  $m/2$ ) plus the area of the right rectangle (from column  $m/2 + 1$  to  $m$ ) and the space  $O(n)$ . After the middle vertex  $(i, m/2)$  is found the problem of finding the longest path from  $(0,0)$  to  $(n, m)$  can be partitioned into two subproblems: finding the longest path from  $(0,0)$  to the middle vertex  $(i, m/2)$  and finding the longest path from the middle vertex  $(i, m/2)$  to  $(n, m)$ . Instead of trying to find these paths, we first try to find the middle vertices in the corresponding rectangles. This can be done in the time equal to the area of these rectangles, which is two times smaller than the area of the original rectangle. Computing in this way, we will find the middle vertices of all rectangles in  $time = area + area/2 + area/4 + \dots < 2 * area$  and therefore compute the longest path in time  $O(nm)$  and space  $O(n)$ .

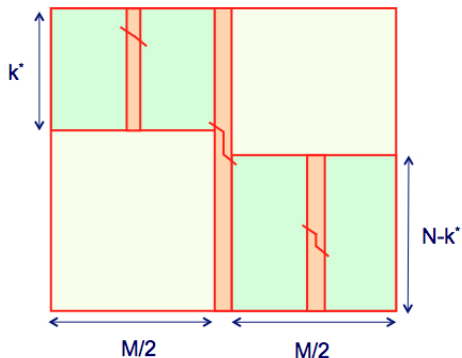
## Space-Efficient Sequence Alignment, Hirschberg algorithm

1. **Path** (source, sink)
2. **if** source and sink are in consecutive columns
3. output the longest path from the source to the sink
4. **else**
5. middle  $<$  middle vertex between source and sink
6. **Path** (source, middle)
7. **Path** (middle, sink)

## Space-Efficient Sequence Alignment, Hirschberg algorithm: details



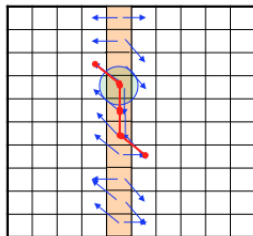
- Iterate this procedure to the left and right!



## Space-Efficient Sequence Alignment, Hirschberg algorithm: details

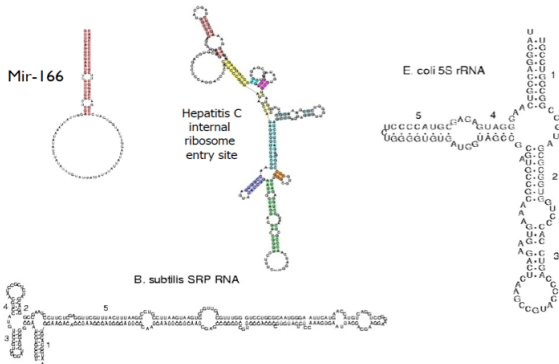
Now, we can find  $k^*$  maximizing  $F(M/2, k) + F^r(M/2, k)$

Also, we can trace the path exiting column  $M/2$  from  $k^*$



Conclusion: In  $O(NM)$  time,  $O(N)$  space,  
we found optimal alignment path at column  $M/2$

# Nussinov Algorithm: The Biological problem

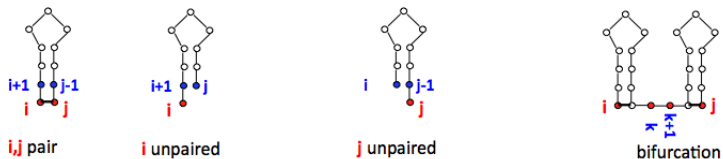


**Figure:** Examples of RNA molecules in nature; many molecules of RNA do not translate into proteins; using the pairing rule A-T, C-G, the molecule could find regions of perfect pairing so to have intrachain interactions. Therefore, the molecule folds into 2 Dimensional shape (termed secondary structure) and then into 3 Dimensional shape (tertiary structure) and regulates cell processes by interacting with proteins.

## Folding i.e. intra chain alignment of a RNA molecule

The intrachain folding of RNA reveals the RNA Secondary Structure

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.



**Figure:** Set of paired positions on interval  $[i, j]$ . 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:  $i, j$  paired;  $i$  unpaired;  $j$  unpaired; combining two substructures. Note that each of these consists of extending or joining substructures of length less than  $j - i + 1$ .

## Nussinov dynamic programming algorithm for RNA folding

1. Let  $\gamma(i, j)$  be the maximum number of base pairs in a folding of subsequence  $S[i \dots j]$ .
2. for  $1 \leq i \leq n$  and  $i < j \leq n$ :  
for  $i = 1, \dots, n : \gamma(i, i) = 0$ ;  
for  $2 \leq i \leq n : \gamma(i, i - 1) = 0$

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

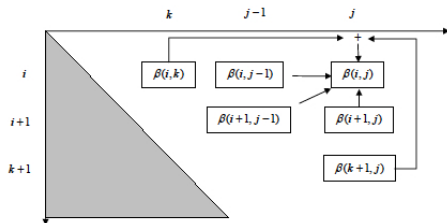
3. Where  $\delta(i, j) = 1$  if  $x_i$  and  $x_j$  are a complementary base pair i.e. (A, U) or (C, G), and  $\delta(i, j) = 0$ , otherwise.

There are  $O(n^2)$  terms to be computed, each requiring calling of  $O(n)$  already computed terms for the case of bifurcation. Thus overall complexity is  $O(n^3)$  time and  $O(n^2)$  space.

## Nussinov algorithm for RNA folding

Note that only the upper (or lower) half of the matrix needs to be filled. Therefore, after initialization the recursion runs from smaller to longer subsequences as follows:

1. for  $l = 1$  to  $n$  do
2. for  $i = 1$  to  $(n + 1 - l)$  do
3.  $j = i + l$
4. compute  $\gamma(i, j)$
5. end for
6. end for

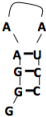




# Nussinov algorithm: example

	G	G	G	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
C								0

Example:  
GGGAAAUCC



	G	G	G	A	A	U	C	C
G	0	0	0	0	0	1	2	3
G	0	0	0	0	0	1	2	3
G		0	0	0	0	1	2	2
A			0	0	0	1	1	1
A				0	0	0	1	1
A					0	0	1	1
U						0	0	0
C							0	0
C								0

Fill up the table (DP matrix) -- diagonal by diagonal



	G	G	G	A	A	U	C	C
G	0	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	1
U						0	0	0
C							0	0
C								0

**Figure:** order: top left, bottom left, right: a matrix will be filled along the diagonals and the solution can be recovered through a traceback step.

## Approximate Search algorithms

Blast, Patternhunter, spaced seeds

## Homology search algorithms: The Biological problem

It is common to observe strong sequence similarity between a protein and its counterpart in another species that diverged hundreds of millions of years ago. Accordingly, the best method to identify the function of a new gene or protein is to find its sequence- related genes or proteins whose functions are already known. The Basic Local Alignment Search Tool (BLAST) is a computer program for finding regions of local similarity between two DNA or protein sequences. It is designed for comparing a query sequence against a target database. It is a heuristic that finds short matches between query and database sequences and then attempts to start alignments from these seed hits. BLAST is arguably the most widely used program in bioinformatics. By sacrificing sensitivity for speed, it makes sequence comparison practical on huge sequence databases currently available.

## BLAST (Basic Local Alignment Search Tools)

While Dynamic Programming (DP) is a nice way to construct alignments, it will often be too slow. Since the DP is  $O(n^2)$ , matching two  $9 \times 10^9$  length sequences would take about  $9 \times 10^{18}$  operations. BLAST is an alignment algorithm which runs in  $O(n)$  time. The key to BLAST is that we only actually care about alignments that are very close to perfect. A match of 70% is worthless; we want something that matches 95% or 99% or more. What this means is that correct (near perfect) alignments will have long substrings of nucleotides that match perfectly. Most popular Blast-wise algorithms use a seed-and-extend approach that operates in two steps: 1. Find a set of small exact matches (called seeds) 2. Try to extend each seed match to obtain a long inexact match.

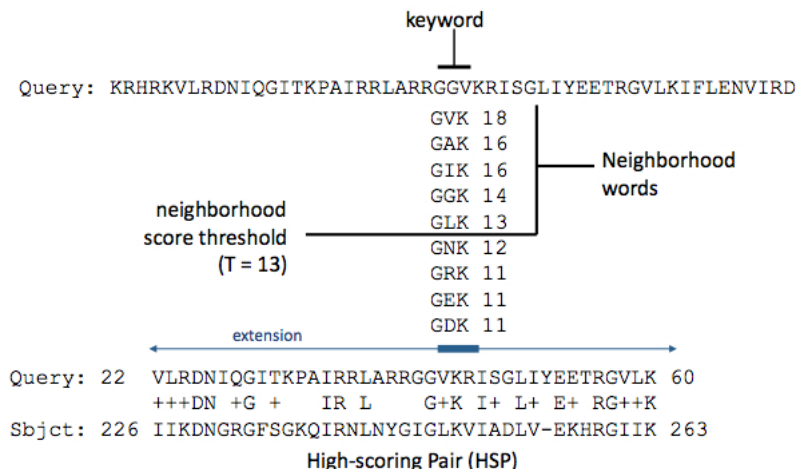
## BLAST (Basic Local Alignment Search Tools)

The steps are as follows:

1. Split query into overlapping words of length  $W$  (the  $W$ -mers)
2. Find a neighborhood of similar words for each word (see the figure next slide).
3. Lookup each word in the neighborhood in a hash table to find where in the database each word occurs. Call these the seeds.
4. Extend all seeds collections until the score of the alignment drops off below a threshold.
5. Report matches with overall highest scores.

## BLAST (Basic Local Alignment Search Tools)

BLAST provides a trade off between speed and sensitivity, with the setting of a "threshold" parameter  $T$ . A higher value of  $T$  yields greater speed, but also an increased probability of missing weak similarities.



## BLAST (Basic Local Alignment Search Tools)

To speed up the homology search process, BLAST employs a filtration strategy: It first scans the database for length- $w$  word matches of alignment score at least  $T$  between the query and target sequences and then extends each match in both ends to generate local alignment (in the sequences) whose alignment score is larger than a threshold  $x$ . The matches are called high-scoring segment pairs (HSPs). BLAST outputs a list of HSPs together with E-values that measure how frequent such HSPs would occur by chance. A HSP has the property that it cannot be extended further to the left or right without the score dropping significantly below the best score achieved on part of the HSP. The original BLAST algorithm performs the extension without gaps. Variants are gapped Blast, psi-blast and others.

## Statistical significance in Blast

- ▶ Assume that the length  $m$  and  $n$  of the query and database respectively are sufficiently large; a segment-pair  $(s, t)$  consists of two segments, one in  $m$  (say the amino acid string: VALLAR) and one in  $n$  (say PAMMAR), of the same length. We think of  $s$  and  $t$  as being aligned without gaps and score this alignment using a substitution score; the alignment score for  $(s, t)$  is denoted by  $\sigma(s, t)$ .
- ▶ Given a cutoff score  $x$ , a segment pair  $(s, t)$  is called a high-scoring segment pair (HSP), if it is locally maximal and  $\sigma(s, t) \geq x$  and the goal of BLAST is to compute all HSPs.
- ▶ The BLAST algorithm has three parameters: the word size  $W$ , the word similarity threshold  $T$  and the minimum match score  $x$ .



## For protein sequences, BLAST operates as follows

The list of all words of length  $W$  that have similarity  $\geq T$  to some word in the query sequence  $m$  is generated. The database sequence  $n$  is scanned for all hits  $t$  of words  $s$  in the list. Each such seed  $(s, t)$  is extended until its score  $\sigma(s, t)$  falls a certain distance below the best score found for shorter extensions and then all best extensions are reported that have score  $\geq x$ . In practice,  $W$  is usually 4 for proteins.

The list of all words of length  $W$  that have similarity  $\geq T$  to some word in the query sequence  $m$  can be produced in time proportional to the number of words in the list. These are placed in a keyword tree and then, for each word in the tree, all exact locations of the word in the database  $n$  are detected in time linear to the length of  $n$ . The original version of BLAST did not allow indels, making hit extension very fast.

Note that the use of seeds of length  $W$  and the termination of extensions with fading scores are both steps that speed up the algorithm, but also imply that BLAST is not guaranteed to find all HSPs.

## For DNA sequences, BLAST operates as follows

- ▶ The list of all words of length  $W$  in the query sequence  $m$  is generated. The database  $n$  is scanned for all hits of words in this list. Blast uses a two-bit encoding for DNA. This saves space and also search time, as four bases are encoded per byte. In practice,  $W$  is usually 12 for DNA.
- ▶ HSP scores are characterized by two parameters,  $W$  and  $\lambda$ . The expected number of HSPs with score at least  $Z$  is given by the E-value, which is:  $E(Z) = Wmne^{-\lambda Z}$ .
- ▶ Essentially,  $W$  and  $\lambda$  are scaling-factors for the search space and for the scoring scheme, respectively.
- ▶ As the E-value depends on the choice of the parameters  $W$  and  $\lambda$ , one cannot compare E-values from different BLAST searches.

- ▶ For a given HSP  $(s, t)$  we transform the raw score  $Z = \sigma(s, t)$  into a bit-score thus:  $Z' = \frac{\lambda Z - \ln W}{\ln 2}$ . Such bit-scores can be compared between different BLAST searches. To see this, solve for  $Z$  in the previous equation and then plug the result into the original E-value.
- ▶ E-values and bit scores are related by  $E = mn2^{-Z'}$
- ▶ The number of random HSPs  $(s, t)$  with  $\sigma(s, t) \geq x$  can be described by a Poisson distribution. Hence the probability of finding exactly  $k$  HSPs with a score  $\geq S$  is given by
 
$$P(k) = \frac{E^k}{k!} e^{-E}$$
- ▶ The probability of finding at least one HSP by chance is  $P = 1 - P(X = 0) = 1 - e^{-E}$ , called the P-value, where  $E$  is the E-value for  $Z$ .
- ▶ BLAST reports E-values rather than P-values as it is easier, for example, to interpret the difference between an E-value of 5 and 10, than to interpret the difference between a P-value of 0.993 and 0.99995. For small E-values  $< 0.01$ , the two values are nearly identical.

## Example of Blast output

### Blast of human beta globin DNA against human DNA

	Score	E	(bits)	Value
Sequences producing significant alignments:				
gi 19849266 gb AF487523.1  Homo sapiens gamma A hemoglobin (HBG1...	289	1e-75		
gi 183868 gb M11427.1 HUMHBG3E Human gamma-globin mRNA, 3' end	289	1e-75		
gi 44887617 gb AY534688.1  Homo sapiens A-gamma globin (HBG1) ge...	280	1e-72		
gi 31726 emb V00512.1 HSGGL1 Human messenger RNA for gamma-globin	260	1e-66		
gi 38683401 ref NR_001589.1  Homo sapiens hemoglobin, beta pseud...	151	7e-34		
gi 18462073 gb AF339400.1  Homo sapiens haplotype PB26 beta-glob...	149	3e-33		

#### ALIGNMENTS

>gi|28380636|ref|NG\_000007.3| Homo sapiens beta globin region (HBB@) on chromosome 11

Length = 81706

Score = 149 bits (75), Expect = 3e-33

Identities = 183/219 (83%)

Strand = Plus / Plus

```
Query: 267   ttgggagatgccacaaagcacctggatgatctcaagggcacctttgccagctgagtga 326
           || ||| | ||   | || | ||||| ||||| ||||| ||||| |||||
Sbjct: 54409 ttcggaaaagctgttatgtctcacggatgacctcaaagggcacctttgctacactgagtgc 54468
```

```
Query: 327   ctgcactgtgacaagctgcatgtggatcctgagaacttc 365
           ||||| ||||| ||||| ||||| ||||| |||||
Sbjct: 54469   ctgcactgtaacaagctgcacgtggaccctgagaacttc 54507
```

## It is possible to search a protein sequence against a DNA database

ttgacctagatgagatgtcgttcacttactgagctacagaaaa

ttg|acc|tag|atg|aga|tgt|cgt|tca|ctt|tta|ctg|agc|tac|aga|aaa  
L T x M R C R S L L L S Y R K

t|tga|cct|aga|tga|gat|gtc|gtt|cac|ttt|tac|tga|gct|aca|gaa|aa  
x P R x D V V H F Y x S T E

tt|gac|cta|gat|gag|atg|tcg|ttc|act|ttt|act|gag|cta|cag|aaa|a  
D L D E M S F T F T E L Q K

**Figure:** Blast DNA query (top) against a database of proteins will process all the potential triplets forming codons

## Example of Blast Pitfalls

### BLAST may also miss a hit

GAGTACTCAACA CCAACAT TAGTGGGCAATGGAAAAT  
|| ||||| ||||| | ||||| |||||  
GAATACTCAACAGCAACATCAATGGGCA GCAGAAAAT



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

The biggest problem for BLAST is low sensitivity (and low speed). Massive parallel machines are built to do Smith Waterman exhaustive dynamic programming. A spaced seed is formed by two words, one from each input sequence, that match at positions specified by a fixed pattern and one don't care symbol respectively. For example, the pattern 1101 specifies that the first, second and fourth positions must match and the third one contain a mismatch. PatternHunter (PH) 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.

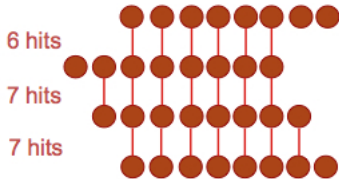
## Blast vs PH vs PH II

If you want to speed up, have to use a longer seed. However, we now face a dilemma: increasing seed size speeds up, but loses sensitivity; decreasing seed size gains sensitivity, but loses speed. How do we increase sensitivity and speed simultaneously? Spaced Seed: nonconsecutive matches and optimized match positions. Represent BLAST seed by 11111111111; Spaced seed: 111010010100110111 where 1 means a required match and 0 means don't care position. This simple change makes a huge difference: significantly increases hit to homologous region while reducing bad hits. 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 uses multiple optimal seeds; it approaches Smith-Waterman sensitivity while is 3000 times faster. Example: Smith-Waterman (SSearch): 20 CPU-days, PatternHunter II with 4 seeds: 475 CPU-seconds: 3638 times faster than Smith-Waterman dynamic programming at the same sensitivity



Sensitivity: The probability to find a local alignment. Specificity:  
 In all local alignments, how many alignments are homologous

Consecutive Positions

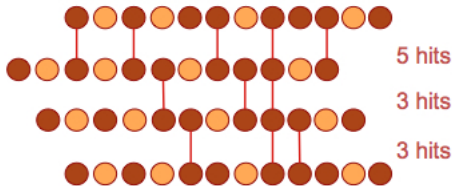


On a 70% conserved region:

Consecutive

Expected # hits: 1.07  
 Prob[at least one hit]: 0.30

Non-Consecutive Positions



Non-consecutive

0.97  
 0.47

- 111010010100110111 (called a model)
  - Eleven required matches (weight=11)
  - Seven "don't care" positions

```

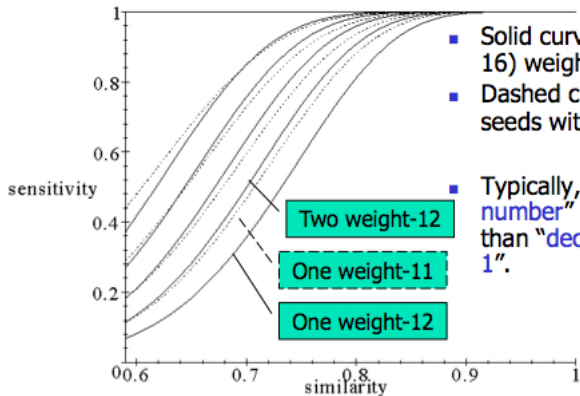
GAGTACTCAACACCAACATTAGTGGCAATGGAAAAT...
|| ||||| |||| | | |||| | |||||
GAATACTCAACAGCAACACTAATGGCAGCAGAAAAT...
      111010010100110111
  
```

- Hit = all the required matches are satisfied.
- BLAST seed model = 1111111111

```

111010010100110111
 111010010100110111
   111010010100110111
    111010010100110111
     111010010100110111
      111010010100110111
       111010010100110111
        111010010100110111
         . . . . .
  
```

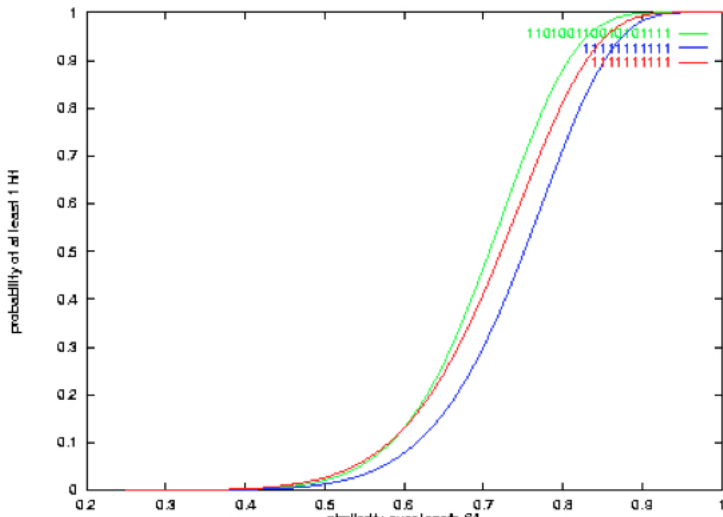
In PatternHunter, the spaced model has often weight 11 and length 18.



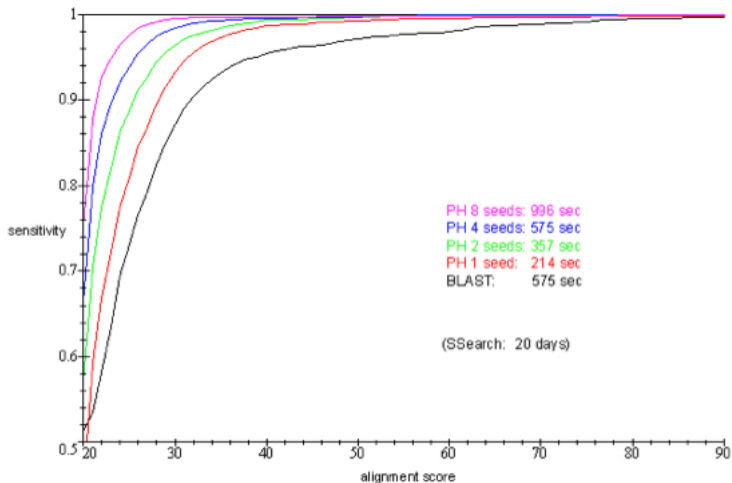
- Solid curves: Multiple (1, 2, 4, 8, 16) weight-12 spaced seeds.
- Dashed curves: Optimal spaced seeds with weight = 11, 10, 9, 8.
- Typically, "Doubling the seed number" gains better sensitivity than "decreasing the weight by 1".

The non-consecutive seed is the primary difference and strength of Patternhunter

## Sensitivity: PH weight 11 seed vs BLAST 11 & 10



## Homology search algorithms growing usage



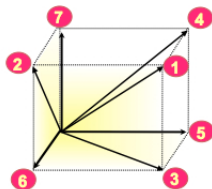
**Figure:** sensitivity versus alignment score

## Progressive alignment

Clustal

## Challenges of extending dynamic programming to n sequences

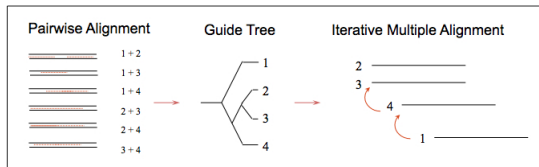
- ▶ For two sequences, there are three ways to extend an alignment
- ▶ for n sequences, a n-dimensional dynamic programming hypercube has to be computed and for each entry we have to evaluate  $(2^n - 1)$  predecessors.
- ▶ Given 3 sequences, the figure below shows a three-dimensional alignment path matrix: there are  $= (2^3 - 1) = 7$  ways to extend an alignment.



1	2	3	4	5	6	7
ABC	AB-	ABC	ABC	ABC	AB-	AB-
ABC	ABC	AB-	ABC	AB-	AB-	ABC
ABC	ABC	ABC	AB-	AB-	ABC	AB-

## Progressive alignment

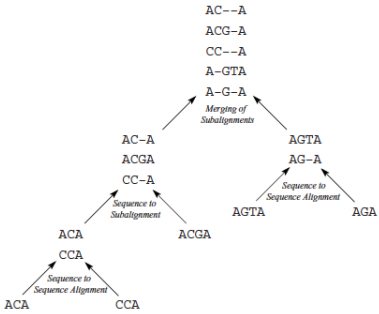
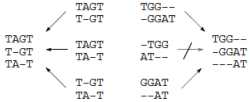
- ▶ Progressive alignment methods are heuristic in nature. They produce multiple alignments from a number of pairwise alignments.
- ▶ Perhaps the most widely used algorithm of this type is CLUSTALW.
- ▶ Given N sequences, align each sequence against each other and obtain a similarity matrix; similarity = exact matches / sequence length (percent identity)
- ▶ Create a guide tree using the similarity matrix; the tree is reconstructed using clustering methods such as UPGMA or neighbor-joining (explained later).
- ▶ Progressive Alignment guided by the tree.



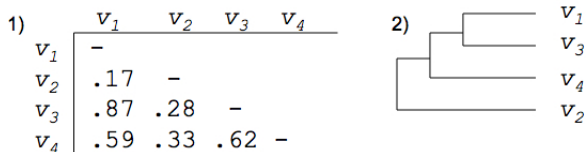


# Progressive alignment

Not all the pairwise alignments build well into multiple sequence alignment (MSA); the progressive alignment builds a final alignment by merging sub-alignments with a guide tree.



## Progressive alignment



Calculate:

- 3)
- $v_{1,3}$  = alignment  $(v_1, v_3)$
  - $v_{1,3,4}$  = alignment  $((v_{1,3}), v_4)$
  - $v_{1,2,3,4}$  = alignment  $((v_{1,3,4}), v_2)$

**Figure:** Progressive alignment of 4 sequences: 1) distance matrix from pairwise alignment; 2) pairwise alignment score analysis; tree showing the best order of progressive alignment, 3) building up the alignment.

## A matrix to measure amino acid changes

Blosum is a symmetric amino acid replacement matrix used as scoring matrix in Blast search and in phylogeny. Using only the conserved regions of protein sequences in a MSA, we compute  $p_{ij}$  i.e. for each column of the MSA, the probability of two amino acids  $i$  and  $j$  replacing each other, and  $p_i$  and  $p_j$  are the background probabilities of finding the amino acids  $i$  and  $j$  in any protein sequence. Finally we compute:

$Score_{ij} = (k^{-1})\log(p_{ij}/p_i p_j)$  where the  $k$  is a scaling factor.

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

## Entropy measure of a multiple alignment

AAA  
AAA  
AAT  
ATC

Let's start from an alignment of four sequences (above the first three columns); Compute the 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);

Compute entropy of each column:  $E = - \sum_{X=A,C,G,T} p_X \log(p_X)$

The entropy for a multiple alignment is the sum of entropies of each column of the alignment.

## Example of a multiple sequence alignment

```
HBA_HUMAN      -----VLSPADKTNVKAANGKVGGAHAGEYGA--EALERMFLSFPPTTKTYFFPHF-DL 48
HBA_HORSE      -----VLSAADKTNVKAAMSKVGGHAGEYGA--EALERMFLGFPPTTKTYFFPHF-DL 48
HBB_HUMAN      -----VHLTPEEKSAVTALWGKVN--VEVGG--EALGRLLVVPVPTQRFDFSPGDL 48
HBB_HORSE      -----VQLSGEEKAAVALALWDKVN--EEVGG--EALGRLLVVPVPTQRFDFSPGDL 48
GLB5_PETMA     PIVDTGSAVPLSAAEKTKIRSAMAPVYSTYETSQV--DILVRFKFTSTPAAQEFFFKPKGL 58
MYG_PHYCA     -----VLSGEWQLVLHVWAKVEADVAGHGQ--DILIRLFKSHPETLEKFRDKL 49
GLB1_GLYDI     -----GLSAAQRQVIAATWEDIAGADNGAVGMDCLIKLRFSAHPQMAAVFGFS--- 48
GLB3_CHITH     -----LSADQISTVQASFDKVK-----GDPVGIYAVFKADPSIMAKFTQFAGK 44
LGB2_LUPLU     -----GALTESQAALVKSSWEEFNANIPKHTH--RFFILVLEIAPAAKDLFSFLKGT 50
              *  :  :  :  :  .  :  :  *  *

HBA_HUMAN      S----HGSAQVKGHGKQVADALTNVAHVYDD----MPNALSALSDLHA--HKLRYDVP 96
HBA_HORSE      S----HGSAQVKAHGKQVGDALTLAVGHLDL----LPGALSNSDLHA--HKLRYDVP 96
HBB_HUMAN      STPDAVHGNPKVKAHGKQVLAFAFSDGLAHLDN----LKGTFATLSELHC--DKLHVDPE 101
HBB_HORSE      SNPGAVHGNPKVKAHGKQVLAHSPGEGVHLDN----LKGTFATLSELHC--DKLHVDPE 101
GLB5_PETMA     TTADQLKKSADVRGHAERINAVNDVASRDDT--EKMSMKLRDLGKHA--KSFQVDPQ 114
MYG_PHYCA     KTEAENKASEDLKQHGVTVLTALGAILKQKGH----HEAEKFLQAQSHA--TGHKPIPK 102
GLB1_GLYDI     -----GASDPGVAALGAKVLAQIGVAVSHLGDV--GRMVAQMKAVGRHKGYGKHKQAQ 101
GLB3_CHITH     DLES-IKGTAPFEIHANRIVGFFSKIIIGELPN-----IEADVNTFVASH(---PRGVTID 95
LGB2_LUPLU     SEVP--QNNPELQAHAGKQVFKLVYEAALQLQVTVVVDATLKNLHLSGVHV---SKGVADA 105
              .  :  :  .  .  .  .  *  :

HBA_HUMAN      NFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR----- 141
HBA_HORSE      NFKLLSHCLLSTLAVHLFNDFTPAVHASLDKFLSSVSTVLTSKYR----- 141
HBB_HUMAN      NFKLLGNVLYCVLAHHPGKEFTPPVQAAYQKVVAVGANALAHKYH----- 146
HBB_HORSE      NFKLLGNVLYVVLARHPGDFPTPELQAASYQKVVAVGANALAHKYH----- 146
GLB5_PETMA     YFKVLAAYIADTYAAG-----DAGFEKLMHSICILLRSAY----- 149
MYG_PHYCA     YLEFISEAIIHVLHRSRHPGDFGADAGGAMKALELFRKDLAAKYKELGYGG 153
GLB1_GLYDI     YFEPLGASLLSMEHRIGGKRNAAAKDANAAYADISGALISGLQS----- 147
GLB3_CHITH     QLNFRAGFPVSYMKAHTD---FAGAEAAVATLDFPGHIFSKH----- 136
LGB2_LUPLU     HFPVVIKALKTIKVYVGAQWSEEKNSAVTIAYDELAIVIKERHNDAA--- 153
              :  .  :  :  :  .  .  .  :  :
```

**Figure:** Chemical properties of amino acids are in color code. The globin proteins from different species could be easily aligned because they have many similar substrings in common.

## Insight into protein structure (3D graph) from MSA analysis



**Figure:** Human globin 3D structure. The small amount of changes in the globin alignment suggests that globin are likely to have very similar structure (3 D graph). Columns rich of gaps often correspond to unstructured regions (loops); conserved regions often correspond to binding sites or regions where one protein interacts with a DNA sequence or with another protein.

## Genome alignments

Burrows- Wheeler transform, de Bruijn graph.

## Burrows- Wheeler transform: saving memory in alignment problems

- ▶ The current sequencing procedures are characterized by highly parallel operations, much lower cost per base, but (unfortunately) they produce several millions of "reads", short stretches of DNA bases (usually 35-400 bp).
- ▶ In many experiments, e.g., in ChIP-Seq, the task is now to align these reads to a reference genome.
- ▶ The main effort is to reduce the memory requirement for sequence alignment (such as Bowtie, BWA and SOAP2); the Burrows- Wheeler transform, BWT (a text compression method) is used.

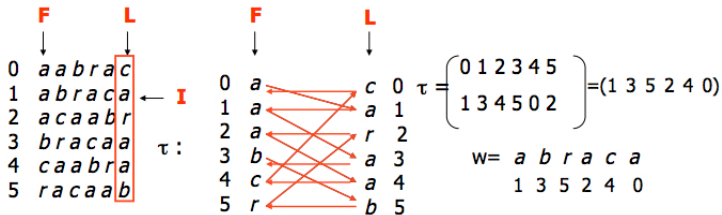


## Burrows-Wheeler Transform

INPUT (example):  $T = \text{"abraca"}$ ; then we sort lexicographically all the cyclic shifts of  $T$

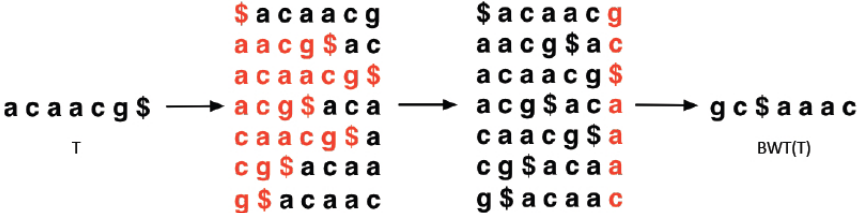
For all  $i \neq l$ , the character  $L[i]$  is followed in  $T$  by  $F[i]$ ; for any character  $ch$ , the  $i$ -th occurrence of  $ch$  in  $F$  corresponds to the  $i$ -th occurrence of  $ch$  in  $L$ .

OUTPUT:  $BWT(T) = \text{caraab}$  and the index  $l=1$ , that denotes the position of the original word  $T$  after the lexicographical sorting. The Burrows-Wheeler Transform is reversible, in the sense that, given  $BWT(T)$  and an index  $l$ , it is possible to recover the original word  $T$ .



# Burrows-Wheeler Transform in alignment: example

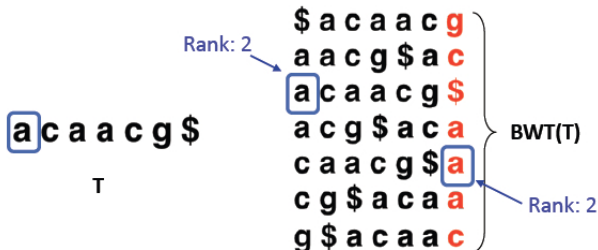
Reversible permutation used originally in compression  
Once BWT(T) is built, all else shown here is discarded



**Figure:** in red the analogy with the suffix array (from Wall lab in Harvard); Note that the cycle and the sort procedures of the Burrows-Wheeler induces a partial clustering of similar characters providing the means for compression.

## Burrows-Wheeler Transform in alignment: example

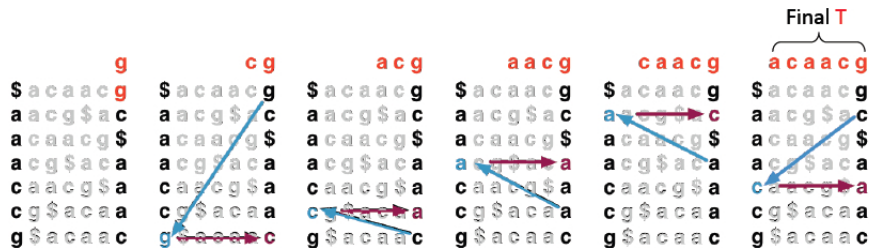
Property that makes  $BWT(T)$  reversible is LF Mapping  
ith occurrence of a character in **L**ast column is same text  
occurrence as the ith occurrence in the **F**irst column



## Burrows-Wheeler Transform in alignment: example

To recreate T from BWT(T), repeatedly apply rule:  $T = \text{BWT}[\text{LF}(i)] + T$ ;  $i = \text{LF}(i)$ .

Where  $\text{LF}(i)$  maps row  $i$  to row whose first character corresponds to  $i$ 's last per LF Mapping



## Burrows-Wheeler Transform in alignment: example

t	a	t	a	t	a	t	a	t	a	\$
a	t	a	t	a	t	a	t	a	\$	t
t	a	t	a	t	a	t	a	\$	t	a
a	t	a	t	a	t	a	\$	t	a	t
t	a	t	a	t	a	\$	t	a	t	a
a	t	a	t	a	\$	t	a	t	a	t
t	a	t	a	\$	t	a	t	a	t	a
a	t	a	\$	t	a	t	a	t	a	t
t	a	\$	t	a	t	a	t	a	t	a
a	\$	t	a	t	a	t	a	t	a	t
\$	t	a	t	a	t	a	t	a	t	a

\$	t	a	t	a	t	a	t	a	t	a
a	\$	t	a	t	a	t	a	t	a	t
a	t	a	\$	t	a	t	a	t	a	t
a	t	a	t	a	\$	t	a	t	a	t
a	t	a	t	a	t	a	\$	t	a	t
a	t	a	t	a	t	a	t	a	\$	t
t	a	\$	t	a	t	a	t	a	t	a
t	a	t	a	\$	t	a	t	a	t	a
t	a	t	a	t	a	\$	t	a	t	a
t	a	t	a	t	a	t	a	\$	t	a
t	a	t	a	t	a	t	a	t	a	\$

**Figure:** in the left, the word "tatatatata\$" undergoes cyclic shift and it is sorted in the right. Note that the BWT(tatatatata\$) is a word (atTTTTaaa\$) with good clustering of T's and A's and so it can be written in a more compact way. The DNA is from an alphabet of 4 symbols so the clustering happens very often

# Next Generation sequencing: The Biological problem

Instead of considering a DNA sequence, for sake of clarity, let's consider an english sentence and we trim all spaces.

Copies of the sentence are divided into fragments called reads which could be converted into kmers. We would like to assemble the original sentence using the reads or the kmers.

Itwasbestoftimesitwastheworstoftimesitwastheageofwisdomitwastheageoffoolishness...

Generate random 'reads'



How do we assemble?

fincredul geoffools itwashebe itwasidom itwasheep apochofinc linesitwas sthepocho nessitwas wastheago theepoch sthepacho hafincrodu astofimes eoffoolish ishnessit hebelleit pochofincr itwashevo twastheage loftimesit domitwasit ochorbelle apochofhe repochofhe astheworst chufonred theageofwi sithwashe sathpoch ofwashe widomitwas ageoffooli twastheor ochorbelle adomitwas silwashe apochofhe ffoolshne eofeliderni hebostofi stheageoff twasthego eworstofi loftimes theepoch of silwashe theepochf stdomitwas astheworst rstofimes worstofim sthepacho geoffools ffoolshne timeshas ishnessit stheageoff eworstofi orstofime fwidomitw wastheago heageofwis incrudul ishnessitw twasthego wastheworst wasthepoch heworstof obelleit wastheago thepochof pochofncr heageofwis stheageofw fincredul astheageof widomitwas wastheago sthepoch dithnessi asthepoch itwasheep twastheage widomitwas hebelleit bestofime apochofhe theepochf sthebestof ishnessit hebelleit itwashebe ishnessitw silwashe agefelidom twastheage silwashe twastheage shnessitw fincredul hebelleit thepochf messitwas domitwas ochorbelle heageofwis oltimesitw sthepacho bestofime twastheage foolshnes finnessitw thebestof itwasheep thepochf obelleit bestofime itwashebe inositwas timesitwas orstofime astofimes twasthebes stofimes adomitwas widomitwas theworst astheworst silwashe theageoff apochofhe etc etc

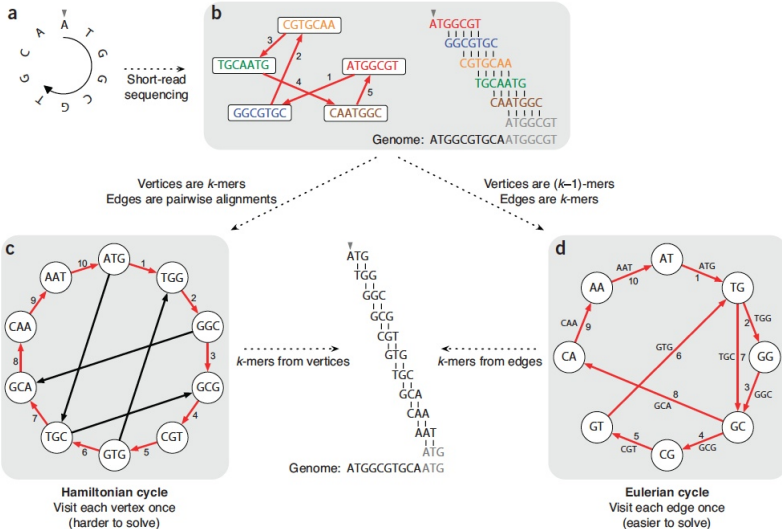
## Convert reads into "Kmers"

Kmer: a substring of defined length

Reads: theageofwi sthebestof astheageof worstofim imesitwas

Kmers : the sth ast wor ime  
(k=3) hea the sth ors mes  
eag heb the rst esi  
age ebe hea sto sit  
geo bes eag tof itw  
eof est age oft twa  
ofw sto geo fti was  
fwi tof eof tim ast

# Three methods to reconstruct the original sequence (a) from the reads (one method (b) uses the reads, two use k-mers derived from the reads (c,d); from Pevzner's paper (see citation))



## Graph approaches in alignment

**(a)** An example small circular genome. In **(b)** reads were represented as nodes in a graph, and edges represented alignments between reads. Following the edges in numerical order allows one to reconstruct the circular genome by combining alignments between successive reads. In **(c)** reads are divided into all possible  $k$ -mers ( $k = 3$ ), ATGGCGT comprises ATG, TGG, GGC, GCG and CGT. Following a Hamiltonian cycle (indicated by red edges) allows one to reconstruct the genome by forming an alignment in which each successive  $k$ -mer (from successive nodes) is shifted by one position. **(d)** Modern short-read assembly algorithms construct a de Bruijn graph by representing all  $k$ -mer prefixes and suffixes as nodes and then drawing edges that represent  $k$ -mers having a particular prefix and suffix. For example, the  $k$ -mer edge ATG has prefix AT and suffix TG. Finding an Eulerian cycle allows one to reconstruct the genome by forming an alignment in which each successive  $k$ -mer (from successive edges) is shifted by one position.



## Hamiltonian graph using reads

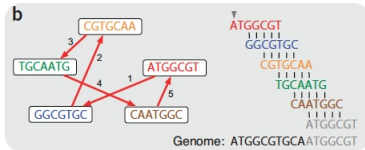


Figure b: The Hamiltonian graph is a graph in which each read is represented by a node and overlap between reads is represented by an arrow (called a directed edge) joining two reads. For instance, two nodes representing reads may be connected with a directed edge if the reads overlap by at least five nucleotides.

**The Hamiltonian cycle, is a path that travels to every node exactly once and ends at the starting node, meaning that each read will be included once in the assembly.**

## Hamiltonian graph using k-mers

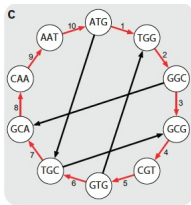


Figure c: The Hamiltonian cycle approach can be generalized to make use of k-mers by constructing a graph as follows. **First**, from a set of reads, make a node for every k-mer appearing as a consecutive substring of one of these reads. **Second**, given a k-mer, define its suffix as the string formed by all its nucleotides except the first one and its prefix as the string formed by all of its nucleotides except the last one. **k-mer to another using a directed edge if the suffix of the former equals the prefix of the latter**, that is, if the two k-mers completely overlap except for one nucleotide at each end. **Third**, look for a Hamiltonian cycle, which represents a candidate genome because it visits each detected k-mer.

## Hamiltonian graph

There is no known efficient algorithm for finding a Hamiltonian cycle in a large graph with millions (let alone billions) of nodes. The Hamiltonian cycle approach was feasible for sequencing the first microbial genome in 1995 and the human genome in 2001. **The computational problem of finding a Hamiltonian cycle belongs to the class of problems that are collectively called NP-Complete.**

## Eulerian graph

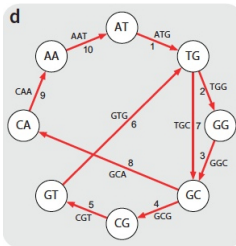


Figure d: Instead of assigning each k-mer contained in some read to a node, we will now assign each such k-mer to an edge. This allows the construction of a **de Bruijn** graph. First, form a node for every distinct prefix or suffix of a k-mer, meaning that a given sequence of length  $k - 1$ . Then, connect node  $x$  to node  $y$  with a directed edge if some k-mer has prefix  $x$  and suffix  $y$ , and label the edge with this k-mer.

## Eulerian graph

We visit all edges of the de Bruijn graph, which represents all possible k-mers; traveling will result in spelling out a candidate genome; for each edge that is traversed, one records the first nucleotide of the k-mer assigned to that edge. Euler considered graphs for which there exists a path between every two nodes (called connected graphs). **He proved that a connected graph with undirected edges contains an Eulerian cycle exactly when every node in the graph has an even number of edges touching it.** The case of directed graphs (that is, graphs with directed edges) is similar. For any node in a directed graph, define its indegree as the number of edges leading into it and its outdegree as the number of edges leaving it. **A graph in which indegrees are equal to outdegrees for all nodes is called balanced.**

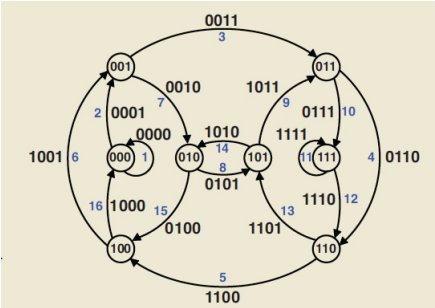
## Eulerian graph

**Eulers theorem states that a connected directed graph has an Eulerian cycle if and only if it is balanced.** In particular, Eulers theorem implies that our de Bruijn graph contains an Eulerian cycle as long as we have located all  $k$ -mers present in the genome. Indeed, in this case, for any node, both its indegree and outdegree represent the number of times the  $k - 1$ -mer assigned to that node occurs in the genome. To see why Eulers theorem must be true, first note that a graph that contains an Eulerian cycle is balanced because every time we traverse an Eulerian cycle and we need to pass through a particular vertex, we enter on one edge of the cycle and exits on the next edge. This pairs up all the edges touching each vertex, showing that half the edges touching the vertex lead into it and half lead out from it. It is a bit harder to see the converse: that every connected balanced graph contains an Eulerian cycle.

# De Bruijn graph: represent the data as a graph

The De Bruijn graph for  $k = 4$  and a two character alphabet composed of the digits 0 and 1. This graph has an Eulerian cycle because each node has indegree and outdegree equal to 2.

Following the blue numbered edges in order from 1 to 16 traces an Eulerian cycle 0000, 0001, 0011, 0110, 1100, 1001, 0010, 0101, 1011, 0111, 1111, 1110, 1101, 1010, 0100, 1000. Recording the first character of each edge label spells the cyclic superstring 0000110010111101.



## Hamiltonian and Eulerian graph complexity

The time required to run a computer implementation of Euler's algorithm is roughly proportional to the number of edges in the de Bruijn graph. In the Hamiltonian approach, the time is potentially a lot larger, because of the large number of pairwise alignments needed to construct the graph and the NP-Completeness of finding a Hamiltonian cycle.



## Summary of the De Bruijn graph method

Sequencing is easy, we generate sub-strings (reads) at random from throughout the genome. In next generation sequencing we have 10s of millions of reads. The difficult part is how we put them back together again in the right order. An intuitive way to do this may be in all versus all comparisons to search for overlaps. This is how traditional assemblers work. The solution offered by the De Bruijn approach is to represent the data as a graph.

The first step of the De Bruijn assembler is to deconstruct the sequencing reads into its constitutive kmers. As specified before a K-mer is a substring of defined length. **If we split reads in kmers we control the size and the overlapping.** To Kmerize the dataset, we move through our read in one letter increments from the beginning to the end until we have recorded all possible 3 letter words. We then do this for all reads in the dataset. From this point on the algorithm operates on kmers rather than on the reads.

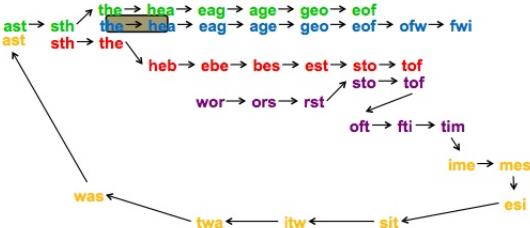
## Details of the De Bruijn graph method

The next stage is to represent the stored kmers in the De Bruijn graph. This is done by searching for overlaps of  $k - 1$ . The graph has all consecutive kmers by  $k - 1$  bases.

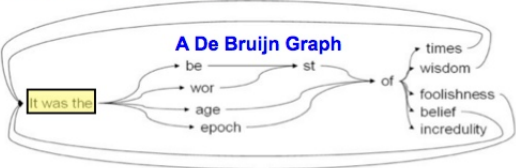
-Adding kmers from a second read of an overlapping region of the genome shows how the graph can be extended. It also reveals the redundancy in the data which need not be stored by the computer. This is how memory efficiency is achieved. -Adding a kmers from a third read that comes from a similar but non-overlapping part of the genome illustrates the effect of repeats, i.e. we get a branch in the graph (see figure in the next slide). **Long unbranched stretches represent unique sequence in the genome, branches and loops are the result of repeats.**

# Details of the De Bruijn graph method

Build a De-Bruijn graph from the kmers



Simplify the graph as much as possible:



## Details of the De Bruijn graph method

The final step is to remove redundancy, result in the final De Bruijn Graph representation of our genome. From this graph, we can see examples of both the strengths and weaknesses of this approach. Strength is that the information from millions of reads is stored in computer memory in a graph that is proportional to the genome size. Another strength is that the overlaps between reads are implicit in the graph, so all the millions versus millions of comparisons are not required. **On the downside, information is lost as repetitive sequences are collapsed into a single representation** (see figure next slide). Another problem is that while this may be a satisfying solution to a computational person, it is not practically useful to a biologist who wants to annotate genes etc.

# Details of the De Bruijn graph method

## De Bruijn assemblies 'broken' by repeats longer than kmer

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity,.... "

### The final assembly (k=3)

wor times itwasthe foolishness st wisdom  
incredulity age epoch be of belief

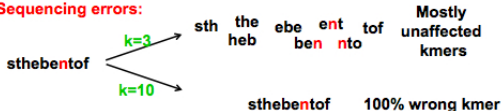


Repeat with a longer "kmer" length

### A better assembly (k=20)

itwasthebestoftimesitwasthehorstoftimesitwastheageofwisdomitwastheageoffools...

## Sequencing errors:

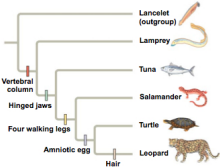
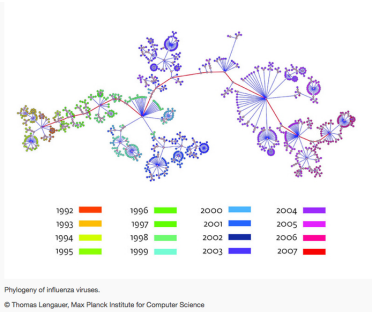


## Phylogeny (parsimony)

Fitch, Sankoff algorithms

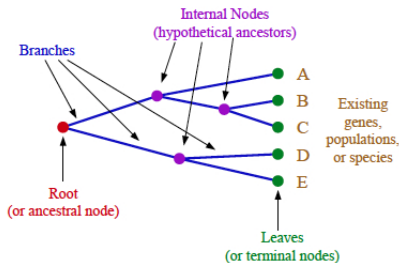
# Phylogeny: The Biological problem

The computational comparison of DNA sequences from different species provides a methodology to understand evolution. Top: a tree of influenza strains; bottom: phylogeny.



## Phylogenetic analysis

The reconstruction of the evolutionary history of species formation could be done by comparing DNA and amino acid sequences. A phylogeny is a tree where the leaves are existing species; an internal node is node with degree greater than one. Internal nodes represent common ancestors. We typically do not have DNA data for internal nodes. Here we use the terms species and taxa in a synonymous way. We compute the tree for each column of an alignment.

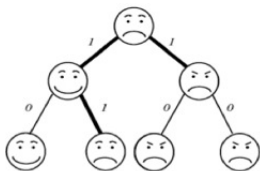


**Figure:** tree representation:  $((a, (b, c)), (d, e))$ ; trees could also be

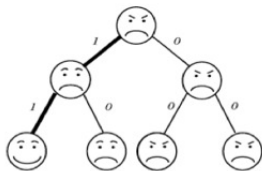


## Phylogeny using parsimony (= economy of mutations)

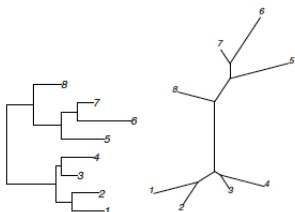
Biological aims: from sequence alignment to phylogeny (a tree) by minimising the number of changes (mutations). Parsimony means economy; there are two main algorithms (Fitch, Sankoff); the output trees are rooted (below the difference between rooted, left, and unrooted, right)



(a) Parsimony Score=3



(b) Parsimony Score=2



## Fitch parsimony model for DNA sequences

### Fitch downpass algorithm

Bottom-up phase: Determine set of possible states for each internal node; top-down phase: Pick states for each internal node. If the descendant state sets  $S_q$  and  $S_r$  overlap, then the state set of node p will include the states present in the intersection of  $S_q$  and  $S_r$ . If the descendant state sets do not overlap, then the state set of p will include all states that are the union of  $S_q$  and  $S_r$ . States that are absent from both descendants will never be present in the state set of p.

1.  $S_p \leftarrow S_q \cap S_r$
2. if  $S_p = 0$  then
3.  $S_p \leftarrow S_q \cup S_r$
4.  $l \leftarrow l + 1$
5. end if

Initialization:  $R_i = [ s_i ]$ ; Do a post-order (from leaves to root) traversal of tree Determine  $R_i$  of internal node i with children j, k:

$$R_i = \begin{cases} R_j \cap R_k & \text{if } R_j \cap R_k \neq 0 \\ R_j \cup R_k & \text{otherwise} \end{cases}$$

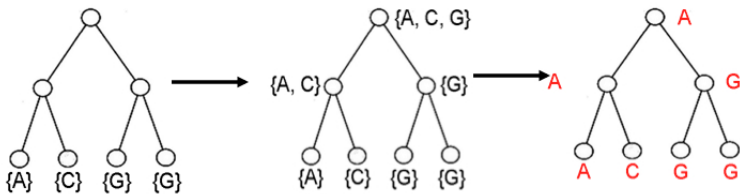
Assume that we have the final state set  $F_a$  of node a, which is the immediate ancestor of node p ( $S_p$ ) that has two children q ( $S_q$ ) and r ( $S_r$ ).

1.  $F_p \leftarrow S_p \cap F_a$
2. if  $F_p \neq F_a$  then
3. if  $S_q \cap S_r \neq 0$  then
4.  $F_p \leftarrow ((S_q \cup S_r) \cap F_a) \cup S_p$
5. else
6.  $F_p \leftarrow S_p \cup F_a$
7. end if
8. end if

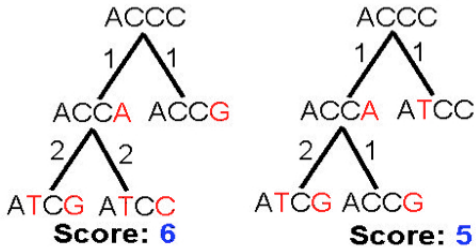
$$R_i(s) = \begin{cases} 0 & \text{if } s_i = s \\ \infty & \text{otherwise} \end{cases}$$

$$R_i(s) = \min_{s'} \{R_j(s') + S(s', s)\} + \min_{s'} \{R_k(s') + S(s', s)\}$$

If the downpass state set of p includes all of the states in the final set of a, then each optimal assignment of final state to a can be combined with the same state at p to give zero changes on the branch between a and p and the minimal number of changes in the subtree rooted at p. If the final set of a includes states that are not present in the downpass set of p, then there is a change on the branch between a and p.



**Figure:** Fitch



**Figure:** Parsimony-score = number of union operations

## Sankoff general parsimony or Sankoff optimisation

### Sankoff downpass algorithm

1. for all i do
2.  $h_i^{(q)} \leftarrow \min_j (c_{ij} + g_j^{(q)})$
3.  $h_i^{(r)} \leftarrow \min_j (c_{ij} + g_j^{(r)})$
4. end for
5. for all i do
6.  $g_i^{(p)} \leftarrow h_i^{(q)} + h_i^{(r)}$
7. end for

Sankoff parsimony is based on a cost matrix  $C = c_{ij}$ , the elements of which define the cost  $c_{ij}$  of moving from a state i to a state j along any branch in the tree.

The cost matrix is used to find the minimum cost of a tree and the set of optimal states at the interior nodes of the tree.

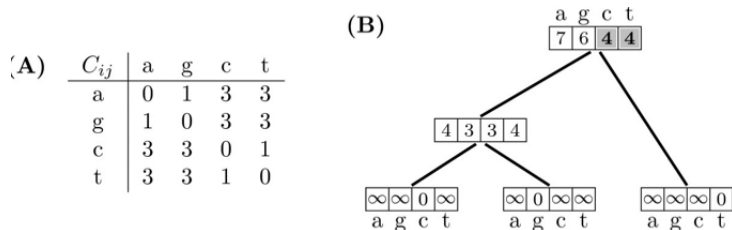
## Sankoff: finding optimal state sets (left) and uppass algorithm (right)

1.  $F_p \leftarrow 0$
2. for all  $i$  in  $F_a$  do
3.  $m \leftarrow c_{i1} + g_1^{(p)}$
4. for all  $j \neq 1$  do
5.  $m \leftarrow \min(c_{ij} + g_j^{(p)}, m)$
6. end for
7. for all  $j$  do
8. if  $c_{ij} + g_j^{(p)} = m$  then
9.  $F_p \leftarrow F_p \cup j$
10. end if
11. end for
12. end for

1. for all  $j$  do
2.  $f_j^{(p)} \leftarrow \min_i (f_i^{(a)} - h_i^{(p)} + c_{ij})$
3. end for

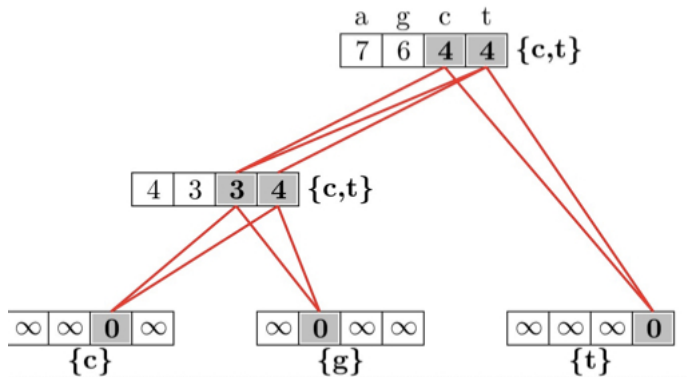
Complexity: if we want to calculate the overall length (cost) of a tree with  $m$  taxa,  $n$  characters, and  $k$  states, it is relatively easy to see that the Fitch algorithms has complexity  $O(mnk)$  and the Sankoff algorithm is of complexity  $O(mnk^2)$ .

## Sankoff: example of downpass



**Figure:** If the leaf has the character in question, the score is 0; else, score is  $\infty$ . Each mutation  $a \rightarrow b$  costs the same in Fitch and differently in Sankoff parsimony algorithm (weighted matrix in A). An example of a weighted matrix for Sankoff (for proteins) is the Blosum, presented before in this course.

## example of uppass



**Figure:** Example of Sankoff algorithm. Note that in the parsimony approaches (Fitch and Sankoff) the tree (i.e. the topology and leaves order) is the hypothesis you are testing. So you try different trees and select the one that is most parsimonious for each column of the alignment, then you select the tree that is the most representative.



## Phylogeny (distance based algorithms)

Additivity, UPGMA, Neighbor Joining

## Phylogeny based on a matrix of distances

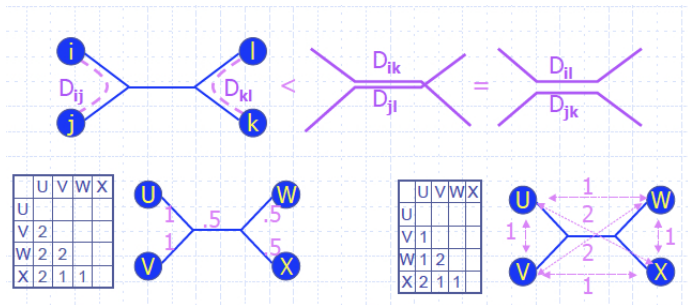
Distance methods convert the changes counted in each column of the alignment, top figure, into a single distance matrix, bottom figure (dissimilarity matrix =  $1 - \text{similarity}$ ) to construct a tree and are kin to clustering methods. We can use the same matrix we use for Blast search, for example the Blosum matrix or others. The UPGMA outputs a rooted tree while the neighbour joining outputs an unrooted tree.

Species	Characters
A	ACTGTTGTTCTGA
B	ACCGTTCCTTCTAG
C	CCTGTTGCTTCTGA
D	ACTGTCCCTTCTAG

	A	B	C	D
A	-	0.75	0.35	0.27
B	0.75	-	0.85	0.33
C	0.35	0.85	-	0.31
D	0.27	0.33	0.31	-

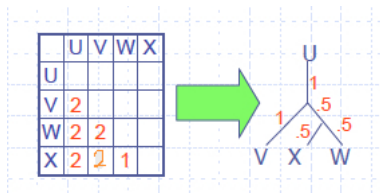
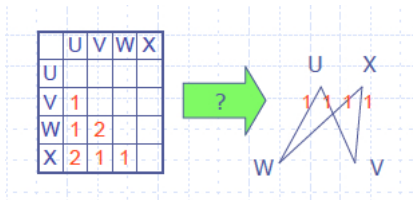
## Additivity: when a distance matrix could be converted into a tree

A matrix  $D$  is additive if for every four indices  $i, j, k, l$  we can write the following:  $D_{ij} + D_{kl} \leq D_{ik} + D_{jl} = D_{il} + D_{jk}$ . If the distance matrix is not additive we could find the tree which best fits the distance matrix.



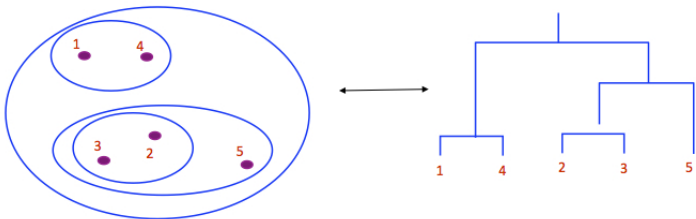
## The additivity property

Top: distance matrix does not turn into a tree; Bottom: the distance matrix turns into a tree.



## UPGMA: Unweighted Pair Group Method with Arithmetic Mean

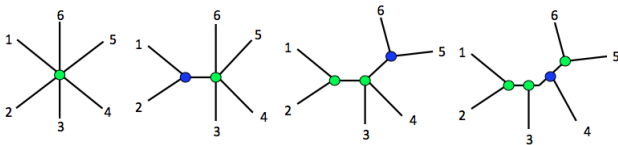
UPGMA is a sequential clustering algorithm that computes the distance between clusters using average pairwise distance and assigns a height to every vertex in the tree, effectively assuming the presence of a molecular clock and dating every vertex. The algorithm produces an ultrametric tree : the distance from the root to any leaf is the same (this corresponds to a constant molecular clock: the same proportion of mutations in any pathway root to leaf). Input is a distance matrix of distances between species; the iteration combines the two closest species until we reach a single cluster.



## UPGMA is also hierarchical clustering

1. Initialization: Assign each species to its own cluster  $C_i$
2. Each such cluster is a tree leaf
3. Iteration:
4. Determine  $i$  and  $j$  so that  $d(C_i, C_j)$  is minimal
5. Define a new cluster  $C_k = C_i \cup C_j$  with a corresponding node at height  $d(C_i, C_j)/2$
6. Update distances to  $C_k$  using weighted average
7. Remove  $C_i$  and  $C_j$
8. Termination: stop when just a single cluster remains

## Neighbor Joining, NJ



**Figure:** NJ starts with a star topology (i.e. no neighbors have been joined) and then uses the smallest distance in the distance matrix to find the next two pairs to move out of the multifurcation then recalculate the distance matrix that now contains a tip less.

1. Identify  $i, j$  as neighbor if their distance is the shortest.
2. Combine  $i, j$  into a new node  $u$ .
3. Update the distance matrix.
4. Distance of  $u$  from the rest of the tree is calculated
5. If only 3 nodes are left finish.

The distance between any taxon (=species) pair  $i$  and  $j$  is denoted as  $d(i, j)$  and can be obtained from the alignment. NJ iteratively selects a taxon pair, builds a new subtree, and agglomerates the pair of selected taxa to reduce the taxon set by one. Pair selection is based on choosing the pair  $i, j$  that minimizes the following  $Q$  (matrix) criterion:

$$Q(i, j) = (r - 2)d(i, j) - \sum_{k=1}^r d(i, k) - \sum_{k=1}^r d(j, k)$$

where  $r$  is the current number of taxa and the sums run on the taxon set. NJ estimates the length of the branch  $(f, u)$  using

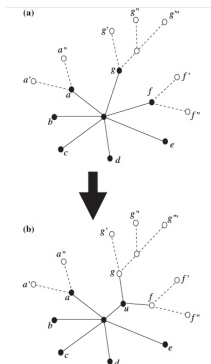
$$d(f, u) = \frac{1}{2}d(f, g) - \frac{1}{2(r-2)}[\sum_{k=1}^r d(f, k) - \sum_{k=1}^r d(g, k)]$$

and  $d(g, u)$  is obtained by symmetry. Finally, NJ replaces  $f$  and  $g$  by  $u$  in the distance matrix, using the reduction formula:

$$d(u, k) = \frac{1}{2}[d(f, k) - d(f, u)] + \frac{1}{2}[d(g, k) - d(g, u)]$$

NJ still reconstructs the correct tree when the distance matrix is perturbed by small noise and that NJ is optimal regarding tolerable noise amplitude.





**Figure:** One NJ agglomeration step. In the current tree (a), the taxon set contains a, b, c, d, e, f, and g; some are original taxa, whereas the others (i.e., a, f, and g) correspond to subtrees built during the previous steps. Tree (b): after selection of the (f, g) pair, a new subtree is built, and both f and g are replaced by a unique taxon denoted as u. NJ terminates when the central node is fully resolved. Neighbor joining on a set of  $r$  taxa requires  $r-3$  iterations. At each step one has to build and search a  $Q$  matrix. Initially the  $Q$  matrix is size  $r^2$ , then the next step it is  $(r-1)^2$ , etc. This leads to an algorithm with a time complexity of  $O(r^3)$ .

Distance matrix

	A	B	C	D	E
B	5				
C	4	7			
D	7	10	7		
E	6	9	6	5	
F	8	11	8	9	8

	U <sub>1</sub>	C	D	E
C	3			
D	6	7		
E	5	6	5	
F	7	8	9	8

	U <sub>1</sub>	C	U <sub>2</sub>
C	3		
U <sub>2</sub>	3	4	
F	7	8	6

	U <sub>2</sub>	U <sub>3</sub>
U <sub>3</sub>	2	
F	6	6

	U <sub>4</sub>
F	5

### Step 1

S calculations

$$S_A = (5+4+7+6+8)/4 = 7.5$$

$$S_B = (5+7+10+9+11)/4 = 10.5$$

$$S_C = (4+7+7+6+8)/4 = 8$$

$$S_D = (7+10+7+5+9)/4 = 9.5$$

$$S_E = (6+9+6+5+8)/4 = 8.5$$

$$S_F = (8+11+8+9+8)/4 = 11$$

$$S_{C,U_1} = (3+6+5+7)/3 = 7$$

$$S_{D,U_1} = (3+7+6+8)/3 = 8$$

$$S_{E,U_1} = (6+7+5+9)/3 = 9$$

$$S_{F,U_1} = (5+6+5+8)/3 = 8$$

$$S_{F,U_2} = (7+8+9+8)/3 = 10.6$$

$$S_{U_1,U_2} = (3+3+7)/2 = 6.5$$

$$S_{C,U_2} = (3+4+8)/2 = 7.5$$

$$S_{U_2,U_3} = (3+4+6)/2 = 6.5$$

$$S_{F,U_3} = (7+8+6)/2 = 10.5$$

$$S_{U_2,U_3} = (2+6)/1 = 8$$

$$S_{U_3,U_4} = (2+6)/1 = 8$$

$$S_F = (6+6)/1 = 12$$

Because  $N-2=0$ , we cannot do this calculation.

### Step 2

Calculate pair with smallest ( $M_{ij}$ ), where  $M_{ij} = D_{ij} - S_i - S_j$

Smallest are

$$M_{AB} = 5 - 7.5 - 10.5 = -13$$

$$M_{DE} = 5 - 9.5 - 8.5 = -13$$

Choose one of these (AB here).

Smallest is

$$M_{C,U_1} = 3 - 7 - 8 = -12$$

$$M_{E,U_1} = 5 - 9 - 8 = -12$$

Choose one of these (DE here).

Smallest is

$$M_{C,U_1} = 3 - 6.5 - 7.5 = -11$$

Smallest is

$$M_{U_2,U_3} = 6 - 8 - 12 = -14$$

$$M_{U_3,U_4} = 6 - 8 - 12 = -14$$

$$M_{U_2,U_4} = 2 - 8 - 8 = -14$$

Choose one of these ( $M_{U_2,U_3}$  here).

### Step 3

Create a node (U) that joins pair with lowest  $M_{ij}$  such that  $S_U = D_{ij}/2 + (S_i - S_j)/2$ .

U<sub>1</sub> joins A and B:

$$S_{U_1} = D_{AB}/2 + (S_A - S_B)/2 = 1$$

$$S_{U_1} = D_{DE}/2 + (S_D - S_E)/2 = 4$$

U<sub>2</sub> joins D and E:

$$S_{U_2} = D_{DE}/2 + (S_D - S_E)/2 = 3$$

$$S_{U_2} = D_{C,U_1}/2 + (S_C - S_{U_1})/2 = 2$$

U<sub>3</sub> joins C and U<sub>1</sub>:

$$S_{U_3} = D_{C,U_1}/2 + (S_C - S_{U_1})/2 = 2$$

$$S_{U_3,U_2} = D_{C,U_1}/2 + (S_{U_1} - S_{U_2})/2 = 1$$

U<sub>4</sub> joins U<sub>2</sub> and U<sub>3</sub>:

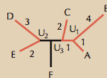
$$S_{U_4,U_2} = D_{U_2,U_3}/2 + (S_{U_2} - S_{U_3})/2 = 1$$

$$S_{U_4,U_4} = D_{U_2,U_3}/2 + (S_{U_2} - S_{U_3})/2 = 1$$

For last pair, connect U<sub>4</sub> and F with branch length = 5.

### Step 4

Join  $i$  and  $j$  according to S above and make all other taxa in form of a star. Branches in black are of unknown length. Branches in red are of known length.



### Step 5

Calculate new distance matrix of all other taxa to U with  $D_{iU} = D_{ik} + D_{jk} - D_{ij}$ , where  $i$  and  $j$  are those selected from above.

### Comments

Note this is the same tree we started with (drawn in unrooted form here).

**Figure:** [http://www.evolution-textbook.org/content/free/tables/Ch\\_27/T11\\_EVOW\\_Ch27.pdf](http://www.evolution-textbook.org/content/free/tables/Ch_27/T11_EVOW_Ch27.pdf)

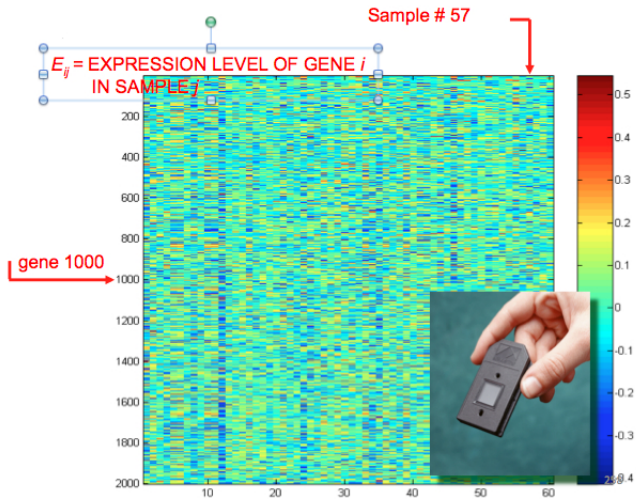
# Clustering

K means, Markov Clustering algorithm

## Algorithms for clustering: The Biological problem

We can use microarrays (DNA chips) to measure the activity (expression level) of the genes in different cells, tissues under varying conditions (with a drug) and at different time points. Expression level is estimated by measuring the amount of mRNA for that particular gene. More mRNA usually indicates more gene activity. Microarray data are usually transformed into a set of large matrices. The clustering analysis allows scientists to identify changes of activity in genes and functional similarity among genes.

# Clustering gene expression data



# Clustering gene expression data

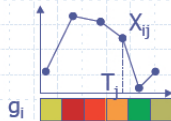
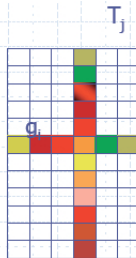
There are two typical experiments:

## ■ Differentiation

- Compare expression levels under different conditions
- A test  $T_j$  represents expression levels of a condition
- E.g., cancer or drug-treated cell vs. normal cell

## ■ Temporal expression

- Explore temporal evolution of expression levels
- A test  $T_j$  represents expression levels at a given time
- E.g., study cell response to heat-shock, starvation



Gene expression profile

**Figure:** The color of the spot indicates activation with respect to control (red) or repression with respect to the control (green) or absence of regulation (yellow) of a gene, or error in the technological process (black). The sample can be all the genes of an organism (example the 6000 genes of yeast), or a selection of genes of interest (+ control genes).

## K-Means Clustering: Lloyd Algorithm

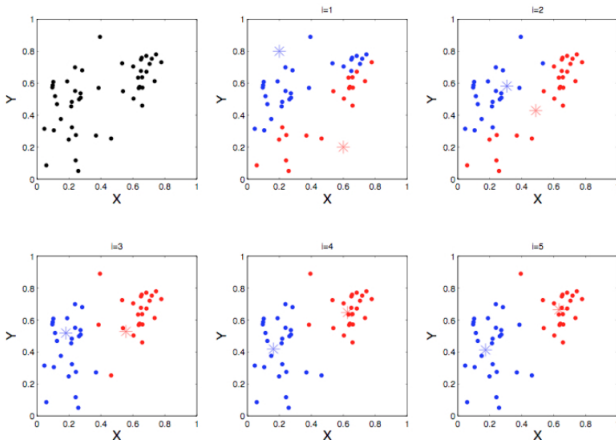
1. Arbitrarily assign the  $k$  cluster centers
2. while the cluster centers keep changing
3. Assign each data point to the cluster  $C_i$  corresponding to the closest cluster representative (center) ( $1 \leq i \leq k$ )
4. After the assignment of all data points, compute new cluster representatives according to the center of gravity of each cluster, that is, the new cluster representative is  $\frac{\sum v}{|C|}$  for all  $v$  in  $C$  for every cluster  $C$

## Progressive greedy K-means Algorithm

1. Select an arbitrary partition  $P$  into  $k$  clusters
2. while forever
3.  $\text{bestChange} \leftarrow 0$
4. for every cluster  $C$
5. for every element  $i$  not in  $C$
6. if moving  $i$  to cluster  $C$  reduces its clustering cost
7. if  $\text{cost}(P) - \text{cost}(P_{i \rightarrow C}) > \text{bestChange}$
8.  $\text{bestChange} \leftarrow \text{cost}(P) - \text{cost}(P_{i \rightarrow C})$
9.  $i' \leftarrow i$
10.  $C' \leftarrow C$
11. if  $\text{bestChange} > 0$
12. Change partition  $P$  by moving  $i'$  to  $C'$
13. else
14. return  $P$



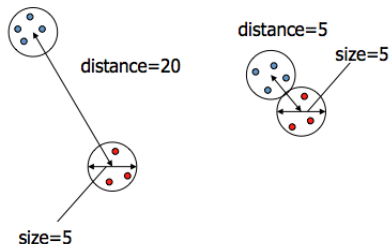
# Progressive greedy K-means Algorithm



**Figure:** K-means progression from left to right and top to bottom; stars are center points (the centers of the cluster).

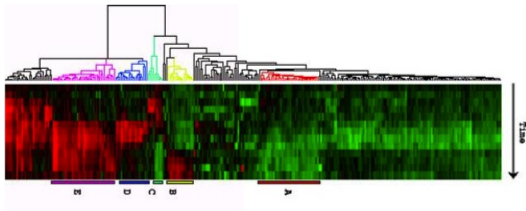
## Progressive greedy K-means Algorithm

The quality of the cluster results could be assessed by ratio of the distance to nearest cluster and cluster diameter. A cluster can be formed even when there is no similarity between clustered patterns. This occurs because the algorithm forces  $k$  clusters to be created. Linear relationship with the number of data points; the complexity is  $O(nKI)$  where  $n$  = number of points,  $K$  = number of clusters,  $I$  = number of iterations.



## Results of a clustering procedure on microarray data

The aim is clustering gene expression: visualising and analyzing vast amounts of biological data as a whole set can be difficult. It is easier to interpret the data if they are partitioned into clusters combining similar data points.



**Figure:** Clustering analysis obtained using Hierarchical clustering (UPGMA). The clusters are coloured differently.

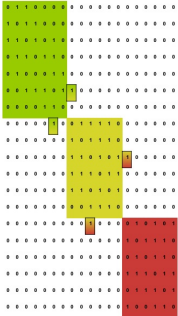
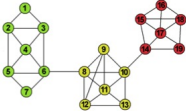
## Markov Clustering algorithm, MCL

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.

ANALOGY: 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. 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.

An **inflation parameter** is responsible for both strengthening and weakening of current. (Strengthens strong currents, and weakens already weak currents). An **expansion parameter**,  $r$ , controls the extent of this strengthening / weakening. In the end, this influences the granularity of clusters.

# The input of MCL could be an adjacency matrix



## MCL Algorithm

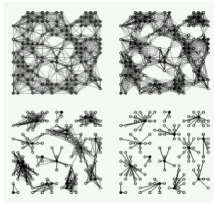
1. Input is an un-directed graph, with power parameter  $e$  (usually  $=2$ ), and inflation parameter  $r$  (usually  $=2$ ).
2. Create the associated adjacency matrix
3. Normalize the matrix;  $M'_{pq} = \frac{M_{pq}}{\sum_i M_{iq}}$
4. Expand by taking the  $e$ -th power of the matrix; for example, if  $e = 2$  just multiply the matrix by itself.
5. Inflate by taking inflation of the resulting matrix with parameter  $r$  :  $M_{pq} = \frac{(M'_{pq})^r}{\sum_i (M'_{iq})^r}$
6. Repeat steps 4 and 5 until a steady state is reached (convergence).

## MCL Algorithm complexity and entropy analysis

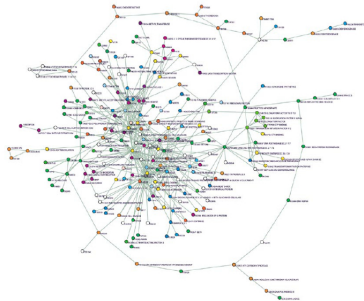
The number of steps to converge is not proven, but experimentally shown to be 10 to 100 steps, and mostly consist of sparse matrices after the first few steps. There are several distinct measures informing on the clustering and its stability such as the following clustering entropy:

$S = -1/L \sum_{ij} (P_{ij} \log_2 P_{ij} + (1 - P_{ij}) \log_2 (1 - P_{ij}))$  where the sum is over all edges and the entropy is normalized by the total number of edges. This might be used to detect the best clustering obtained after a long series of clusterings with different granularity parameters each time.

The expansion step of MCL has time complexity  $O(n^3)$ . The inflation has complexity  $O(n^2)$ . However, the matrices are generally very sparse, or at least the vast majority of the entries are near zero. Pruning in MCL involves setting near-zero matrix entries to zero, and can allow sparse matrix operations to improve the speed of the algorithm vastly.



**Figure:** Progression from left to right and top to bottom



**Figure:** Biological applications: protein families analysis (tribeMCL)

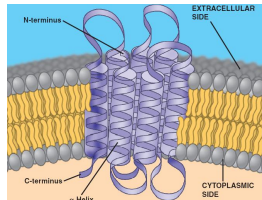


## Hidden Markov Models in Bioinformatics

Viterbi, Forward, Backward, Genscan, TMHMM, sensitivity,  
accuracy

## Hidden Markov Models: The biological problem

Identifying genes and their parts (exons and introns) in a genome is an important challenge; another problem we will work on is about membrane proteins that are important for cell import/export. In this case we would like to predict the location in the amino acid sequence of all the transmembrane helices. The prediction of gene parts and of the membrane protein topology (i.e. which parts are outside, inside and buried in the membrane) will require to train the model with a database of experimentally determined genes / transmembrane helices and to validate the model with an other database. The figure below describes a 7 helix membrane protein forming a sort of a cylinder (3D graph) across the cell membrane.



## Basic of Hidden Markov Models

HMMs form a useful class of probabilistic graphical models used to find genes, predict protein structure and classify protein families.

Definition: A hidden Markov model (HMM) has an Alphabet =  $b_1, b_2, \dots, b_M$ , set of states  $Q = 1, \dots, K$ , and transition probabilities between any two states

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

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

Start probabilities  $a_{0i}$

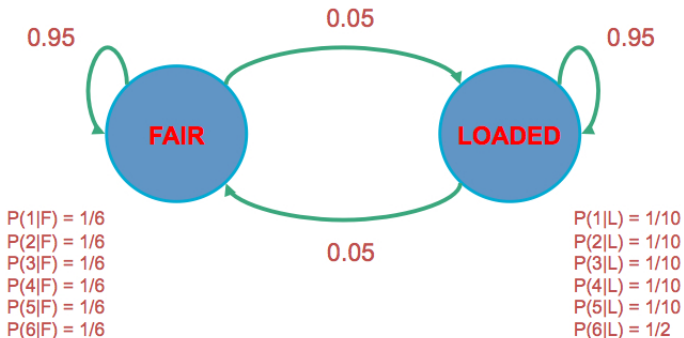
$a_{01} + \dots + a_{0K} = 1$

Emission probabilities within each state  $e_i(b) = P(x_i = b | \pi_i = k)$

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

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

# The dishonest casino model

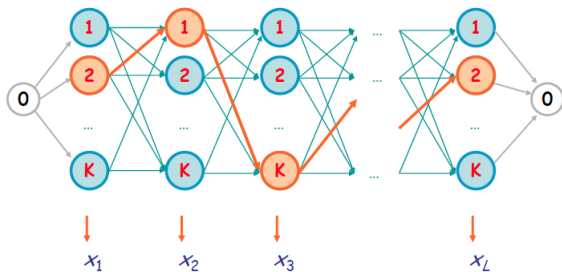


<http://ai.stanford.edu/serafim/>

## The dishonest casino

- ▶ Known: The structure of the model
- ▶ The transition probabilities
- ▶ Hidden: What the casino did (ex FFFFFLLLLLLLLFFFF)
- ▶ Observable: The series of die tosses, ex 3415256664666153...
- ▶ What we must infer:
- ▶ When was a fair die used?
- ▶ When was a loaded one used?

## A “parse” of a sequence



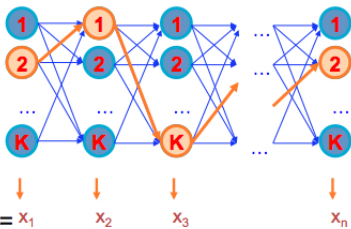
$$\Pr(x, \pi) = a_{0\pi_1} \prod_{i=1}^L e_{\pi_i}(x_i) \cdot a_{\pi_i\pi_{i+1}}$$

Given a sequence  $x = x_1 x_2 \dots x_N$ , a parse of  $x$  is a sequence of states  $\pi = \pi_1, \pi_2, \dots, \pi_N$

Given a sequence  $\mathbf{x} = x_1, \dots, x_N$

and a parse  $\pi = \pi_1, \dots, \pi_N$ ,

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



$$P(\mathbf{x}, \pi) = P(x_1, \dots, x_N, \pi_1, \dots, \pi_N) = x_1 \quad x_2 \quad x_3 \quad \dots \quad x_n$$

$$P(x_N, \pi_N | \pi_{N-1}) P(x_{N-1}, \pi_{N-1} | \pi_{N-2}) \dots P(x_2, \pi_2 | \pi_1) P(x_1, \pi_1) =$$

$$P(x_N | \pi_N) P(\pi_N | \pi_{N-1}) \dots P(x_2 | \pi_2) P(\pi_2 | \pi_1) P(x_1 | \pi_1) P(\pi_1) =$$

$$a_{0\pi_1} a_{\pi_1\pi_2} \dots a_{\pi_{N-1}\pi_N} e_{\pi_1}(x_1) \dots e_{\pi_N}(x_N)$$

# The three main questions on HMMs

## 1. Evaluation

GIVEN a HMM  $M$ , and a sequence  $x$ ,

FIND  $\text{Prob}[x | M]$

## 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  $\theta = (e_i(\cdot), a_{ij})$  that maximize  $P[x | \theta]$

Evaluation: forward algorithm or the backwards algorithm;  
decoding: Viterbi; Learning: Baum Welch = forward-backward  
algorithm (not here)



## Lets 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  $\theta = a_{ij}, e_i(\cdot)$

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

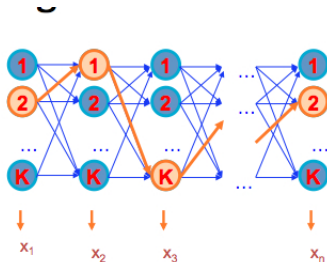
Similarly,  $P[x, \pi | M]$  and  $P[x, \pi]$  are the same

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

GIVEN  $x = x_1 x_2 \dots x_N$

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

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



We can use dynamic programming!

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

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

What is  $V_k(i+1)$ ?

From definition,

$$\begin{aligned} V_l(i+1) &= \max_{\{\pi_1, \dots, i\}} P[x_1 \dots x_i, \pi_1, \dots, \pi_i, x_{i+1}, \pi_{i+1} = l] \\ &= \max_{\{\pi_1, \dots, i\}} P(x_{i+1}, \pi_{i+1} = l \mid x_1 \dots x_i, \pi_1, \dots, \pi_i) P[x_1 \dots x_i, \pi_1, \dots, \pi_i] \\ &= \max_{\{\pi_1, \dots, i\}} P(x_{i+1}, \pi_{i+1} = l \mid \pi_i) P[x_1 \dots x_{i-1}, \pi_1, \dots, \pi_{i-1}, x_i, \pi_i] \\ &= \max_k P(x_{i+1}, \pi_{i+1} = l \mid \pi_i = k) \max_{\{\pi_1, \dots, i-1\}} P[x_1 \dots x_{i-1}, \pi_1, \dots, \pi_{i-1}, x_i, \pi_i = k] = \\ & e_l(x_{i+1}) \max_k a_{kl} V_k(i) \end{aligned}$$

# The Viterbi Algorithm

Input:  $x = x_1, \dots, x_N$

**Initialization:**

$$V_0(0) = 1 \quad (0 \text{ is the imaginary first position})$$

$$V_k(0) = 0, \text{ for all } k > 0$$

**Iteration:**

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

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

**Termination:**

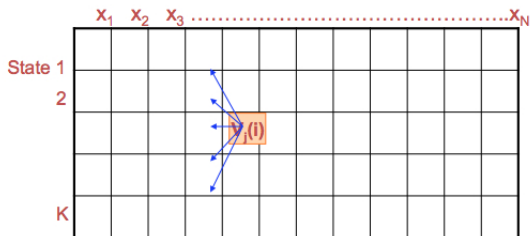
$$P(x, \pi^*) = \max_k V_k(N)$$

**Traceback:**

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

$$\pi_{i-1}^* = \text{Ptr}_{\pi_i}(i)$$

# The Viterbi Algorithm



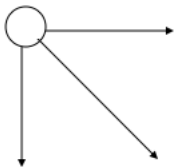
Similar to “aligning” a set of states to a sequence

**Time:**

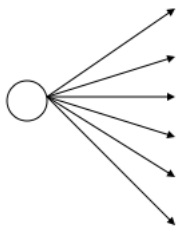
$$O(K^2N)$$

**Space:**

$$O(KN)$$



Valid directions in the  
*alignment problem.*



Valid directions in the  
*decoding problem.*

## Generating a sequence by the model

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

1. Start at state  $\pi_1$  according to prob  $a_{0\pi_1}$
2. Emit letter  $x_1$  according to prob  $e_{\pi_1}(x_1)$
3. Go to state  $\pi_2$  according to prob  $a_{\pi_1\pi_2}$
4. until emitting  $x_n$

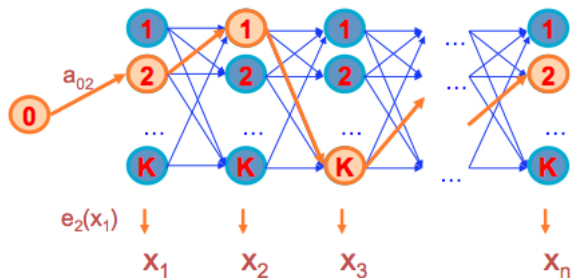


Figure:

## Evaluation

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

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

$P(\pi_i = k \mid x)$  Probability that the  $i^{\text{th}}$  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  $x$ :

$$P(x) = \sum_{\pi} P(x, \pi) = \sum_{\pi} P(x | \pi) P(\pi)$$

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

$$f_k(i) = P(x_1 \dots x_i, \pi_i = k) \quad (\text{the forward probability})$$

## The Forward Algorithm derivation

Define the forward probability:

$$\begin{aligned}f_i(i) &= P(x_1 \dots x_i, \pi_i = l) \\&= \sum_{\pi_1 \dots \pi_{i-1}} P(x_1 \dots x_{i-1}, \pi_1, \dots, \pi_{i-1}, \pi_i = l) e_l(x_i) \\&= \sum_k \sum_{\pi_1 \dots \pi_{i-2}} P(x_1 \dots x_{i-1}, \pi_1, \dots, \pi_{i-2}, \pi_{i-1} = k) a_{kl} e_l(x_i) \\&= e_l(x_i) \sum_k f_k(i-1) a_{kl}\end{aligned}$$

## The Forward Algorithm

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

### Initialization:

$$f_0(0) = 1$$

$$f_k(0) = 0, \text{ for all } k > 0$$

### Iteration:

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

### Termination:

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

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

## Comparison between Viterbi and Forward

### VITERBI

#### Initialization:

$$V_0(0) = 1$$

$$V_k(0) = 0, \text{ for all } k > 0$$

#### Iteration:

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

#### Termination:

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

### FORWARD

#### Initialization:

$$f_0(0) = 1$$

$$f_k(0) = 0, \text{ for all } k > 0$$

#### Iteration:

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

#### Termination:

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

## Motivation for the Backward Algorithm

We want to compute

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

the probability distribution on the  $i^{\text{th}}$  position, given  $x$

We start by computing

$$\begin{aligned} P(\pi_i = k, x) &= P(x_1 \dots x_i, \pi_i = k, x_{i+1} \dots x_N) \\ &= P(x_1 \dots x_i, \pi_i = k) P(x_{i+1} \dots x_N \mid x_1 \dots x_i, \pi_i = k) \\ &= P(x_1 \dots x_i, \pi_i = k) P(x_{i+1} \dots x_N \mid \pi_i = k) \end{aligned}$$

Forward,  $f_k(i)$       Backward,  $b_k(i)$

## The Backward Algorithm derivation

Define the backward probability:

$$\begin{aligned}b_k(i) &= P(x_{i+1} \dots x_N \mid \pi_i = k) \\&= \sum_{\pi_{i+1} \dots \pi_N} P(x_{i+1}, x_{i+2}, \dots, x_N, \pi_{i+1}, \dots, \pi_N \mid \pi_i = k) \\&= \sum_l \sum_{\pi_{i+1} \dots \pi_N} P(x_{i+1}, x_{i+2}, \dots, x_N, \pi_{i+1} = l, \pi_{i+2}, \dots, \pi_N \mid \pi_i = k) \\&= \sum_l e_l(x_{i+1}) a_{kl} \sum_{\pi_{i+1} \dots \pi_N} P(x_{i+2}, \dots, x_N, \pi_{i+2}, \dots, \pi_N \mid \pi_{i+1} = l) \\&= \sum_l e_l(x_{i+1}) a_{kl} b_l(i+1)\end{aligned}$$

## The Backward Algorithm

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

### Initialization:

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

### Iteration:

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

### Termination:

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

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

Time:  $O(K^2N)$

Space:  $O(KN)$

Useful implementation technique to avoid underflows

Viterbi:        sum of logs

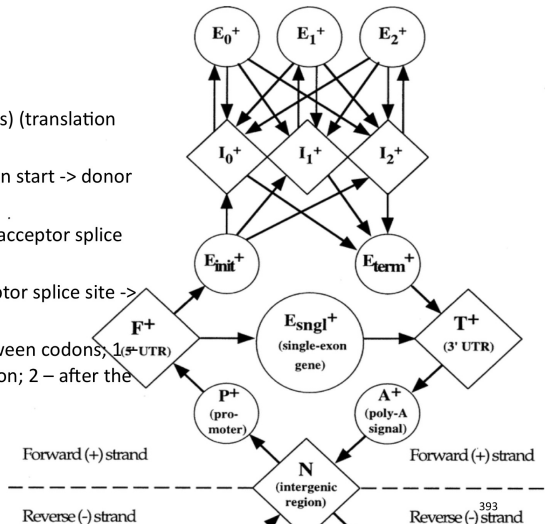
Forward/Backward: rescaling at each position by multiplying by a constant



# Identifying genes and their parts (exons and introns)

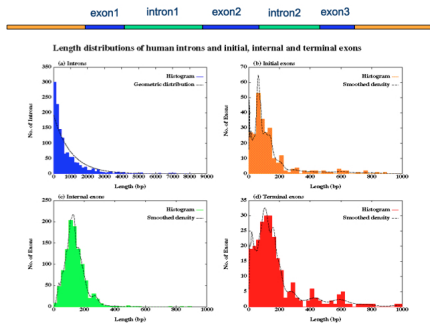
## GenScan

- N - intergenic region
- P - promoter
- F - 5' untranslated region
- $E_{\text{sngl}}$  - single exon (intronless) (translation start -> stop codon)
- $E_{\text{init}}$  - initial exon (translation start -> donor splice site)
- $E_k$  - phase k internal exon (acceptor splice site -> donor splice site)
- $E_{\text{term}}$  - 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



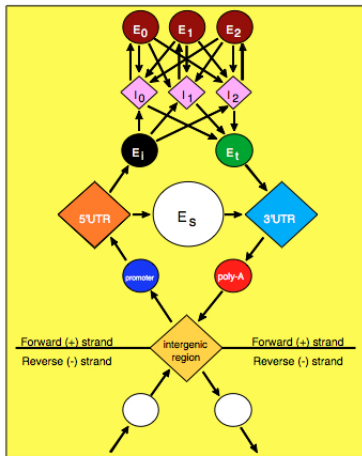
## Identifying genes and their parts (exons and introns)

In order to identify genes and their parts (exons and introns) we need to know their length distribution (see example in figures below). Human genes comprise about 3% of the human genome; average length:  $\sim 8,000$  DNA base pairs (bp); 5-6 exons/gene; average exon length:  $\sim 200$  bp; average intron length:  $\sim 2,000$  bp;  $\sim 8\%$  genes have a single exon and some exons can be as small as 1 or 3 bp. Below the statistics we could implement into a HMM.



# Identifying genes and their parts (exons and introns)

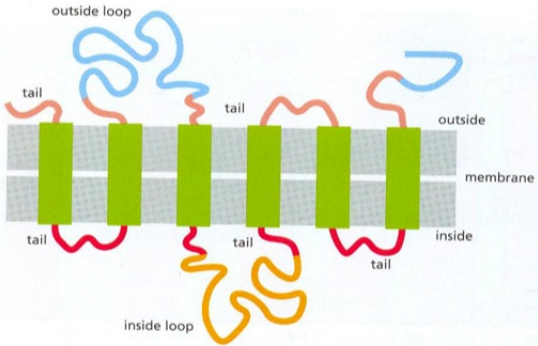
## GENSCAN (Burge & Karlin)



62001	AGGACAGGTA CCGCTGTCAAT CACTTAGACC TCACCCTGTG GAGCCACACC
62051	CTAGGTTTGG CCAATCTACT CCCAGGAGCA GGGAGGGCAC GAGCCAGGGC
62101	TGGGCATAAA AGCCAGGGCA GAGCCACTTA TTGCTTCAAT TTGCTCTGAA
62151	CCCACTGTGG TTCCTAGCA ACTTCAACCA GAGC
62201	
62251	CTT TGTGATCAAG GTTACAAGAC
62301	AGGTTTAGG AGACCAATAG AACTGGGCA TGGGAGACA GAGAGACTC
62351	TTGGTTTTCT GATAGGCATC GACTCTGCTT GCCTATTGGT CTATTTGGC
62401	ACCTTATTC TCCCTGGTGG CTACCTTGGG ACCGAGAGGG TCTTGGAGGG
62451	CTCTGGGAT CTGCCCACGC CAGATCTGCT TATGGGCAC GCTAAAGTGG
62501	AGGCTAAGG CAGAAGAGG CCGGGGCGCT TTAGTAAAGG CCGGGTCAAC
62551	CTGACAAAG TCAGGGGAC CTGCGGACA CTGAGGAGG GAGAGTGG
62601	CGAGGTGAC GTGATGTTGG AGAGCTTAGG TGGAGCTTA TGGGACCTTT
62651	GATGTTTTCT TTCCCCTGCT TERCATAGGT TAAGTTCATG TCATAGGAAG
62701	GGGAGAGTA ACAGGGTACA GTTTAGATG GGAACAGAC GAATGATTCG
62751	ATCAGTGGG AAGTCTCAGG ATCCGTTTAG TTCTTTTAA TTGCTGTGCA
62801	TAACAATGTT TTCTTTGTTT TTAATCTTGG CTCTCTTTTCT TTCTCTCTCC
62851	CGCAATTTTT ACVATTATAC TTAAGGCTCT AACAFTGGGG ATAAACAAGG
62901	GRAATATCTC TGAGATACAT TAAGTACTTT AAAAATAAAC TTACACAGTT
62951	CTCGTAGTA CATTACTATT TGGAAATAT TTGTCCTTAT TTGCATATCG
63001	ATAAECTCCG TACTTTATTT TCTTTTATTT TTAATTGATA CATATCATTT
63051	ATACATATTT ATGGTTTAAA GTGATAGTGT TAAATATGTT TACACATATTT
63101	GACCAATACA GGTAATTTT GCATTTGTAA TTTTAAAAA TGCTTTCTTC
63151	TTTTAAATA CTTTTTGT TTACTTATTT CTAATACTTT CCTTAACTCC
63201	TTTTCTTCAG GGCATAAATG ATACATGTA TCAAGCTCTT TTGCACATTT
63251	CTAAAGATA ACAGTGATAA TTCTGTGTT AAGGCAATAG CAATATTTCT
63301	GCATATAAAT ATTTCTGCAT ATAAATGTA ACCGATGTAA GAGGTTTCA
63351	ATTTGTAATA GAGCTAGACA TCCAGCTACC ATTCCTCTTT TATTITAGG
63401	TTGGGATAAG GCTGGATAT TCCGAGTCCA AGCTAGGCCC TTTTGTAAAT
63451	CATGTTGATA CCTTATATCT TCTCTCCACA CTCCCTGGGC AACTGCTGAG
63501	TCTGTGGTCT GGCCCAACAC TTGGGCAAG AATTCACCCC ACCAGTGCAG
63551	GCTGCTTATC AGAAAGTGT GCTGTGGTGG GCTAATGCC TGGGCCACAA
63601	GTATCACTAA GCTGCTCTTC TTGCTGTGCA ATTTCTATTA AAGTTCCTTT
63651	TGTTCCCTAA GTCCAACTAC TAACTGGG GATATATGA AGGCTCTTGA
63701	GCATCTGAT TCTGCTCAAT AAAAAGATT TATTTCATTT CAATGATGTT

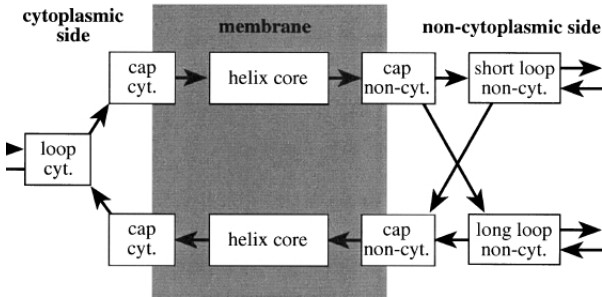
**Figure:** The model (left) and the output (right) of Genscan prediction of a genomic region; the result is a segmentation of a genome sequence, i.e. the colours map the HMM states with the predicted functional genomic

# Prediction of amino acid segments included in membrane proteins



amino acid sequence	MGDVCDETEFGILVA...SVALRPRKHGRWIV...FWVDNGTEQ...PEHMTKLHMM...
state sequence	ooooooooohhhhh...hhhhiiiiiihhhh...hhhoooo00...000oooohhh...
topology	

**Figure:** top: the 3D graph previous figure could be represented as a 2D graph; bottom, 3 state prediction: each amino acid could be in the membrane (h), outside the cell (o) or inside the cell (i)

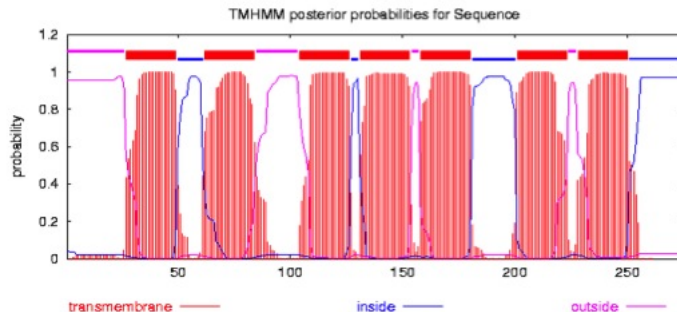


**Figure:** The THMM model: a three state prediction model (h,o,i) could be then refined adding more states, for example caps, i.e. the boundary between outside and membrane and inside and membrane. This refinement improves the prediction of the topology of the protein.

# TMHMM <http://www.cbs.dtu.dk/services/TMHMM/>

```

# Sequence Length: 274
# Sequence Number of predicted TMs: 7
# Sequence Exp number of AAs in TMs: 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
    
```



## Assessing performances: Sensitivity and specificity

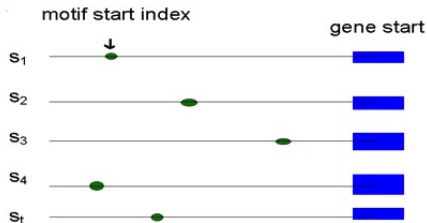
1. be predicted to occur: Predicted Positive (PP)
2. be predicted not to occur: Predicted Negative (PN)
3. actually occur: Actual Positive (AP)
4. actually not occur: Actual Negative (AN)
5. True Positive  $TP = PP \cap AP$
6. True Negative  $TN = PN \cap AN$
7. False Negative  $FN = PN \cap AP$
8. False Positive  $FP = PP \cap AN$
9. Sensitivity: probability of correctly predicting a positive example  $S_n = TP / (TP + FN)$
10. Specificity: probability of correctly predicting a negative example  $S_p = TN / (TN + FP)$  or
11. probability that positive prediction is correct  $S_p = TP / (TP + FP)$

## Gibbs sampling: the Biological problem



## Gibbs sampling: the Biological problem

given a set of sequences, find the motif shared by all or most sequences; while its starting position in each sequence is unknown, each motif appears exactly once in one sequence and it has fixed length.



**Figure:** Several genes are co-regulated (activated or repressed) by same protein that binds before the gene start (transcription factor)

Gibbs Sampling is an example of a Markov chain Monte Carlo algorithm; it 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. It could be used to identify short strings, motifs, common to all co-regulated genes which are not co-aligned. The algorithm in brief:

1. Randomly choose starting positions  $s = (s_1, \dots, s_t)$  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 at random based on the probabilities calculated in step 4.
6. Repeat steps 2-5 until there is no improvement

## Gibbs sampling

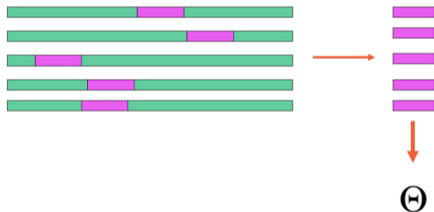
Considering a set of unaligned sequences, we choose initial guess of motifs

1. Select a **random** position in each sequence

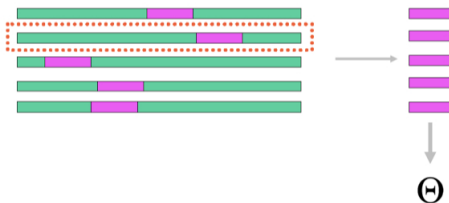


**Figure:** motifs in purple, the rest of the sequences in green; next figures:  $\theta$  is the weight matrix i.e. the frequency of each base in the aligned set of motifs; red the best fitting motif; in y axis the likelihood of each motif with respect to the current weight matrix.

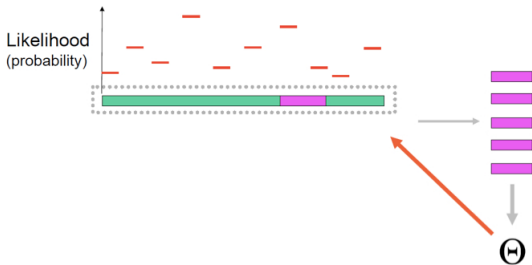
## 2. Build a weight matrix



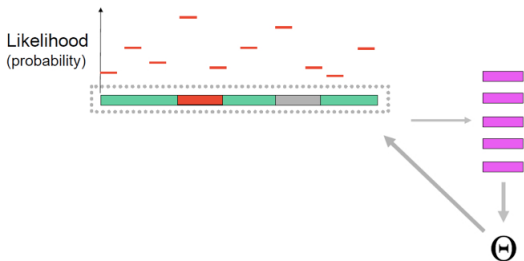
## 3. Select a sequence at random



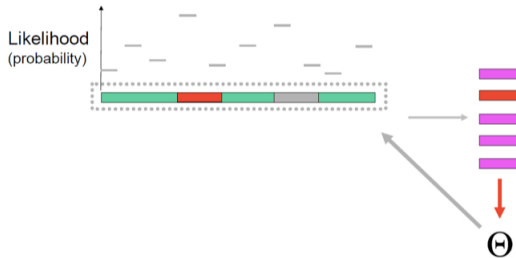
#### 4. Score possible sites in seq using weight matrix



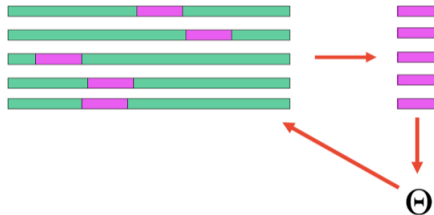
#### 5. Sample a new site proportional to likelihood



## 6. Update weight matrix



## 7. Iterate until convergence (no change in sites/ $\Theta$ )

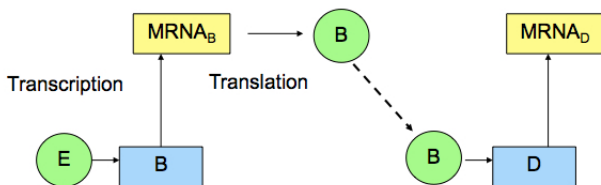


# Algorithms for Biological Networks

Wagner algorithm

## Biological Networks: the biological problem

A biological network is a group of genes in which individual genes can influence the activity of other genes. Let us assume that there are two related genes, B and D, neither of which 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, but both CX and E will have elevated levels of  $mRNA_B$  and low levels of  $mRNA_D$ .

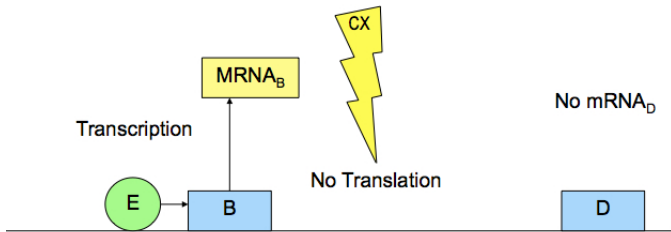


**Figure:** We have E only; B is a Primary Target of E; Production of  $mRNA_B$  is enhanced by E; D is a Secondary Target of E; Production of  $mRNA_D$  is enhanced by B



What is a genetic perturbation? it is an experimental manipulation of gene activity by manipulating either a gene itself or its product.

Such perturbations include point mutations, gene deletions, overexpression, inhibition of translation, or any other interference with the activity of the product.



**Figure:** E and CX both present; B is a Primary Target; Production of  $RNA_B$  is enhanced by E; Production of  $RNA_D$  is decreased (prevented)

## Network reconstruction: direct and indirect effects

When manipulating a gene and finding that this manipulation affects the activity of other genes, the question often arises as to whether this is caused by a direct or indirect interaction?

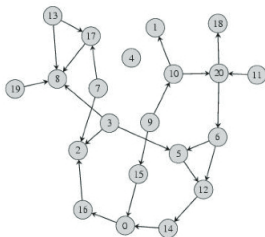
An algorithm to reconstruct a genetic network from perturbation data should be able to distinguish direct from indirect regulatory effects.

Consider a series of experiments in which the activity of every single gene in an organism is manipulated. (for instance, non-essential genes can be deleted, and for essential genes one might construct conditional mutants.) The effect on mRNA expression of all other genes is measured separately for each mutant.

- ▶ How to reconstruct a large genetic network from  $n$  gene perturbations in fewer than  $n^2$  steps?
- ▶ 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: For each gene  $g_i$ , compare the control experiment to perturbed experiment and identify the differentially expressed genes Use the most parsimonious graph that yields the graph as its reachable graph

The nodes of the graph correspond to genes, and two genes are connected by a directed edge if one gene influences the activity of the other.

(a)



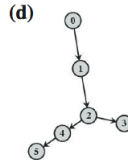
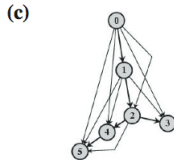
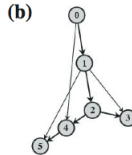
(b)

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

(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

(a)  
 0: 1 2 3 4 5  
 1: 2 3 4 5  
 2: 3 4 5  
 3:  
 4: 5  
 5:



**Figure:** 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. Lets call Gpars the most parsimonious network compatible with Acc.

Figure A shows a graph representation of a hypothetical genetic network of 21 genes. Figure B shows an alternative representation of the network shown in A. For each gene  $i$ , it simply shows which genes activity state the gene influences directly. In graph theory, a list like that shown in Fig. B is called the adjacency list of the graph. We will denote it as  $Adj(G)$ , and will refer to  $Adj(i)$  as the set of nodes (genes) adjacent to (directly influenced by) node  $i$ . One might also call it the list of nearest neighbors in the gene network, or the list of direct regulatory interactions.

When perturbing each gene in the network shown in Figure A, one would get the list of influences on the activities of other genes shown in Figure C.

Starting from a graph representation of the network in Figure A, one arrives at the list of direct and indirect causal interactions in Figure C by following all paths leaving a gene. That is, one follows all arrows emanating from the gene until one can go no further.

## The adjacency list completely defines the structure of a gene network

In graph theory, the list  $\text{Acc}(G)$  is called the accessibility list of the graph  $G$ , because it shows all nodes (genes) that can be accessed (influenced in their activity state) from a given gene by following paths of direct interactions.

In the context of a genetic network one might also call it the list of perturbation effects or the list of regulatory effects.

$\text{Acc}(i)$  is the set of nodes that can be reached from node  $i$  by following all paths of directed edges leaving  $i$ .  $\text{Acc}(G)$  then simply consists of the accessibility list for all nodes  $i$

The adjacency matrix of a graph  $G$ ,  $A(G) = (a_{ij})$  is an  $n$  by  $n$  square matrix, where  $n$  is the number of nodes (genes) in the graph. An element  $a_{ij}$  of this matrix is equal to one if and only if a directed edge exists from node  $i$  to node  $j$ . All other elements of the adjacency matrix are zero.

The accessibility matrix  $P(G) = p_{ij}$  is also an  $n$  by  $n$  square matrix. An element  $p_{ij}$  is equal to one if and only if a path following directed edges exists from node  $i$  to node  $j$ . otherwise  $p_{ij}$  equals zero.

Adjacency and accessibility matrices are the matrix equivalents of adjacency and accessibility lists.

Lets first consider only graphs without cycles, where cycles are paths starting at a node and leading back to the same node.

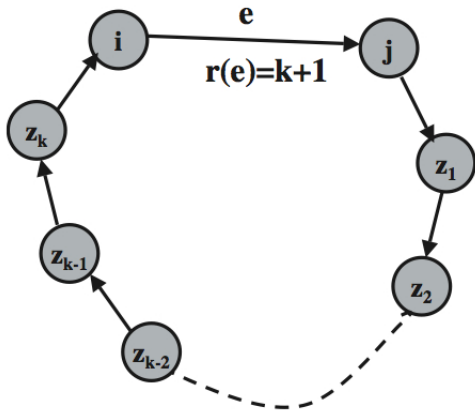
Graphs without cycles are called acyclic graphs.

Later generalize to graphs with cycles.

An acyclic directed graph defines its accessibility list, but the converse is not true.

In general, if  $Acc$  is the accessibility list of a graph, there is more than one graph  $G$  with the same accessibility list.



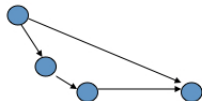


**Figure:** A shortcut is an edge connecting two nodes,  $i$  and  $j$  that are also connected via a longer path of edges. The shortcut  $e$  is a shortcut range  $k+1$ . That is, when eliminating  $e$ ,  $i$  and  $j$  are still connected by a path of length  $k+1$ .

## Wagner 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.

## Theorem

- ▶ Let  $Acc$  be the accessibility list of an acyclic digraph. Then there exists exactly one graph  $G_{pars}$  that has  $Acc$  as its accessibility list and that has fewer edges than any other graph  $G$  with  $Acc$  as its accessibility list.
- ▶ This means that for any list of perturbation effects there exists exactly one genetic network  $G$  with fewer edges than any other network with the same list of perturbation effects.
- ▶ Definition: An accessibility list  $Acc$  and a digraph  $G$  are compatible if  $G$  has  $Acc$  as its accessibility list.  $Acc$  is the accessibility list induced by  $G$ .
- ▶ Definition: Consider two nodes  $i$  and  $j$  of a digraph that are connected by an edge  $e$ . The range  $r$  of the edge  $e$  is the length of the shortest path between  $i$  and  $j$  in the absence of  $e$ . If there is no other path connecting  $i$  and  $j$ , then  $r := \infty$ .

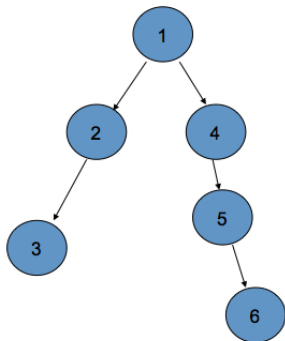
## Theorem

Let  $Acc(G)$  be the accessibility list of an acyclic directed graph,  $G_{pars}$  its most parsimonious graph, and  $V(G_{pars})$  the set of all nodes of  $G_{pars}$ . Then the following equation (1):

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

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  $Acc(i)$ .

## Example



**Figure:**  $Adj(1) = Acc(1) - (Acc(2) + Acc(3) + Acc(4) + Acc(5) + Acc(6)) = (2, 3, 4, 5, 6) - (3 \cup (5, 6) \cup 6) = (2, 4)$

Proof: I will first prove that every node in  $\text{Adj}(i)$  is also contained in the set defined by the right hand side of (1).

Let  $x$  be a node in  $\text{Adj}(i)$ . This node is also in  $\text{Acc}(i)$ . Now take, without loss of generality any node  $j \in \text{Acc}(i)$ . Could  $x$  be in  $\text{Acc}(j)$ ? If  $x$  could be in  $\text{Acc}(j)$  then we could construct a path from  $i$  to  $j$  to  $x$ . But because  $x$  is also in  $\text{Adj}(i)$ , there is also an edge from  $i$  to  $x$ . This is a contradiction to  $G$  pars being shortcut-free. Thus, for no  $j \in \text{Acc}(i)$  can  $x$  be in  $\text{Acc}(j)$ .  $x$  is therefore also not an element of the union of all  $\text{Acc}(j)$  shown on the right-hand side of (1). Thus, subtracting this union from  $\text{Acc}(i)$  will not lead to the difference operator in (1) eliminating  $x$  from  $\text{Acc}(i)$ . Thus  $x$  is contained in the set defined by the right-hand side of (1).

Next to prove: Every node in the set of the right-hand side of (1) is also in  $\text{Adj}(i)$ .

Let  $x$  be a node in the set of the right-hand side of (1). Because  $x$  is in the right hand side of (1),  $x$  must a fortiori also be in  $\text{Acc}(i)$ . That is,  $x$  is accessible from  $i$ . But  $x$  can not be accessible from any  $j$  that is accessible from  $i$ .

For if it were, then  $x$  would also be in the union of all  $\text{Acc}(j)$ . Then taking the complement of  $\text{Acc}(i)$  and this union would eliminate  $x$  from the set in the right hand side of (1). In sum,  $x$  is accessible from  $i$  but not from any  $j$  accessible from  $i$ . Thus  $x$  must be adjacent to  $i$ .

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  $i$ .

Proof: Let  $j$  be a node accessible from node  $i$ . Assume that there is a node  $k$  accessible from  $j$ , such that  $k$  is adjacent to  $i$ . That is,  $j \in \text{Acc}(i)$ ,  $k \in \text{Acc}(j)$  and  $k \in \text{Adj}(i)$ . That  $k$  is accessible from  $j$  implies that there is a path of length at least one from  $j$  to  $k$ . For the same reason, there exists a path of length at least one connecting  $i$  to  $j$ . In sum, there must exist a path of length at least two from  $i$  to  $k$ . However, by assumption, there also exists a directed edge from  $i$  to  $k$ . Thus, the graph  $G$  can not be short-cut free.



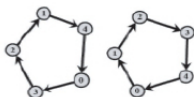
## Step 2: How about graphs with cycles?

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

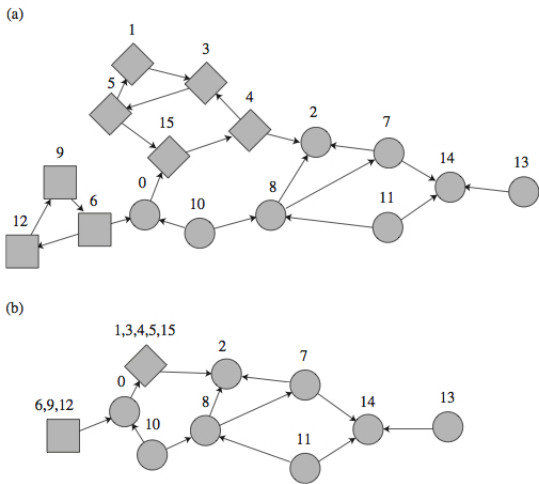
Can't decide a unique graph if cycle happens

Not an algorithmic but an experimental limitation



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



**Figure:** Basic idea: Shrink each cycles (strongly connected components) into one node and apply the algorithm of step 1. A graph after shrinking all the cycles into nodes is called a condensation graph

## How good is this algorithm?

1. Unable to resolve cycled graphs
2. Require more data than conventional methods using gene expression correlations.
3. There are many networks consistent with the given accessibility list. The algorithm construct the most parsimonious one.
4. The same problem was proposed around 1980 which is called transitive reduction.
5. 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.
6. 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  $G$ )

## Complexity

- ▶ Measures of algorithmic complexity are influenced by the average number of entries in a nodes accessibility list. Let  $k < n - 1$  be that number.
- ▶ For all practical purposes, there will be many fewer entries than that, not only because accessibility lists with nearly  $n$  entries are not accessibility lists of acyclic digraphs, but also because most real-world graphs are sparse.
- ▶ During execution, each node accessible from a node  $j$  induces one recursive call of PRUNEACC, after which the node accessed from  $j$  is declared as visited.
- ▶ Thus, each entry of the accessibility list of a node is explored no more than once.
- ▶ However, line 15 of the algorithm loops over all nodes  $k$  adjacent to  $j$ . If  $a = |Adj(j)|$ , on average, then overall computational complexity becomes  $O(nka)$ .

## Comments on the code

The algorithm itself takes the accessibility list of a graph and eliminates entries inconsistent with Theorem 2 and Corollary 2.

It does so recursively until only the adjacency list of the shortcut-free graph is left.

The algorithm is shown as pseudocode. Because it operates on lists, programming languages such as perl or library extensions of other languages permitting list operations will facilitate its implementation.

(In Appendix a perl implementation of the algorithm, where accessibility and adjacency list are represented by a two-dimensional hashing array.)

```
1  for all nodes  $i$  of  $G$ 
2       $Adj(i) = Acc(i)$ 
3
4  for all nodes  $i$  of  $G$ 
5      if node  $i$  has not been visited
6          call PRUNE_ACC( $i$ )
7          end if
8
9  PRUNE_ACC( $i$ )
10     for all nodes  $j \in Acc(i)$ 
11         if  $Acc(j) = \emptyset$ 
12             declare  $j$  as visited.
13         else
14             call PRUNE_ACC( $j$ )
15         end if
16
17     for all nodes  $j \in Acc(i)$ 
18         for all nodes  $k \in Adj(j)$ 
19             if  $k \in Acc(i)$ 
20                 delete  $k$  from  $Adj(i)$ 
21             end if
22     end for
23     declare node  $i$  as visited
24 end PRUNE_ACC( $i$ )
```

The algorithm needs an accessibility list for each node  $i$ ,  $Acc(i)$ , which would be obtained from gene perturbation data and subsequent gene activity measurements for a genetic network.

In lines one and two, for each node  $i$  the adjacency list  $Adj(i)$  is initialized as equal to the accessibility list.

The algorithm will delete elements from this  $Adj(i)$  until the adjacency list of the most parsimonious network of  $Acc(G)$  is obtained.

```
1  for all nodes  $i$  of  $G$ 
2       $Adj(i) = Acc(i)$ 
3  for all nodes  $i$  of  $G$ 
4      if node  $i$  has not been visited
5          call PRUNE_ACC( $i$ )
6      end if

7  PRUNE_ACC( $i$ )
8      for all nodes  $j \in Acc(i)$ 
9          if  $Acc(j) = \emptyset$ 
10             declare  $j$  as visited.
11          else
12             call PRUNE_ACC( $j$ )
13          end if

14      for all nodes  $j \in Acc(i)$ 
15          for all nodes  $k \in Adj(j)$ 
16             if  $k \in Acc(i)$ 
17                 delete  $k$  from  $Adj(j)$ 
18             end if
19      declare node  $i$  as visited
20  end PRUNE_ACC( $i$ )
```

The master loop in lines 3-6 cycles over all nodes of  $G$ , and calls the routine PRUNE\_ACC for each node  $i$ .

In the last statement of this routine (line 19) the calling node is declared as visited.

A visited node is a node whose adjacency list  $Adj(i)$  needs not be modified any further.

This is the purpose of the conditional statement in the master loop (line 4), which skips over nodes that have already been visited.

```
1  for all nodes  $i$  of  $G$ 
2       $Adj(i)=Acc(i)$ 
3  for all nodes  $i$  of  $G$ 
4      if node  $i$  has not been visited
5          call PRUNE_ACC( $i$ )
6      end if

7  PRUNE_ACC( $i$ )
8      for all nodes  $j \in Acc(i)$ 
9          if  $Acc(j)=\emptyset$ 
10             declare  $j$  as visited.
11          else
12             call PRUNE_ACC( $j$ )
13          end if

14      for all nodes  $j \in Acc(i)$ 
15          for all nodes  $k \in Adj(j)$ 
16             if  $k \in Acc(i)$ 
17                 delete  $k$  from  $Adj(i)$ 
18             end if
19  declare node  $i$  as visited
20  end PRUNE_ACC( $i$ )
```

Aside from storing  $Acc$  and  $Adj$ , the algorithm thus also needs to keep track of all visited nodes.

In an actual implementation,  $Acc$ ,  $Adj$ , and any data structure that keeps track of visited nodes would need to be either global variables or passed into the routine PRUNE\_ACC, preferably by reference.

In contrast, the calling node  $i$  needs to be a local variable because of the recursivity of PRUNE\_ACC.

```
1  for all nodes  $i$  of G
2       $Adj(i) = Acc(i)$ 
3
4  for all nodes  $i$  of G
5      if node  $i$  has not been visited
6          call PRUNE_ACC( $i$ )
7          end if
8
9  PRUNE_ACC( $i$ )
10     for all nodes  $j \in Acc(i)$ 
11         if  $Acc(j) = \emptyset$ 
12             declare  $j$  as visited.
13         else
14             call PRUNE_ACC( $j$ )
15         end if
16
17     for all nodes  $j \in Acc(i)$ 
18         for all nodes  $k \in Adj(j)$ 
19             if  $k \in Acc(i)$ 
20                 delete  $k$  from  $Adj(i)$ 
21             end if
22     end if
23     declare node  $i$  as visited
24     end PRUNE_ACC( $i$ )
```



## Function PRUNE\_ACC

It contains of two loops. The first loop (lines 8-13) cycles over all nodes  $j$  accessible from the calling node  $i$ . If there exists a node accessible from  $j$ , then PRUNE\_ACC is called from  $j$ . If no node is accessible from  $j$ , that is, if  $Acc(j) = \emptyset$ , then  $j$  is declared as visited.

Because its accessibility list is empty, its adjacency list must be empty as well ( $Adj(i) \subseteq Acc(i)$ ), and needs no further modification.

```
1  for all nodes  $i$  of  $G$ 
2       $Adj(i) = Acc(i)$ 
3  for all nodes  $i$  of  $G$ 
4      if node  $i$  has not been visited
5          call PRUNE_ACC( $i$ )
6      end if
7  PRUNE_ACC( $i$ )
8      for all nodes  $j \in Acc(i)$ 
9          if  $Acc(j) = \emptyset$ 
10             declare  $j$  as visited.
11          else
12             call PRUNE_ACC( $j$ )
13          end if
14      for all nodes  $j \in Acc(i)$ 
15          for all nodes  $k \in Adj(j)$ 
16             if  $k \in Acc(i)$ 
17                 delete  $k$  from  $Adj(i)$ 
18             end if
19      declare node  $i$  as visited
20  end PRUNE_ACC( $i$ )
```

Thus, through the first loop PRUNE\_ACC calls itself recursively until a node is reached whose accessibility list is empty.

There always exists such a node, otherwise the graph would not be acyclic.

This also means that infinite recursion is not possible for an acyclic graph. Thus, the algorithm always terminates.

More precisely, the longest possible chain of nested calls of PRUNE\_ACC is  $(n-1)$  if G has  $n$  nodes.

For any node  $i$  calling PRUNE\_ACC, the number of nested calls is at most equal to the length of the longest path starting at  $i$ .

```
1  for all nodes  $i$  of G
2       $Adj(i) = Acc(i)$ 
3  for all nodes  $i$  of G
4      if node  $i$  has not been visited
5          call PRUNE_ACC( $i$ )
6      end if

7  PRUNE_ACC( $i$ )
8      for all nodes  $j \in Acc(i)$ 
9          if  $Acc(j) = \emptyset$ 
10             declare  $j$  as visited.
11          else
12             call PRUNE_ACC( $j$ )
13          end if

14      for all nodes  $j \in Acc(i)$ 
15          for all nodes  $k \in Adj(j)$ 
16             if  $k \in Acc(i)$ 
17                 delete  $k$  from  $Adj(i)$ 
18             end if
19      declare node  $i$  as visited
20  end PRUNE_ACC( $i$ )
```

The second loop of PRUNE\_ACC (lines 14-18) only starts once the algorithm has explored all nodes accessible from the calling node  $i$ , that is, as the function calls made during the first loop return.

In the second loop the principle of Corollary 2 is applied.

Specifically, the second loop cycles over all nodes  $j$  accessible from  $i$  in line 14.

```
1  for all nodes  $i$  of  $G$ 
2       $Adj(i) = Acc(i)$ 
3  for all nodes  $i$  of  $G$ 
4      if node  $i$  has not been visited
5          call PRUNE_ACC( $i$ )
6      end if
7  PRUNE_ACC( $i$ )
8      for all nodes  $j \in Acc(i)$ 
9          if  $Acc(j) = \emptyset$ 
10             declare  $j$  as visited.
11          else
12             call PRUNE_ACC( $j$ )
13          end if
14      for all nodes  $j \in Acc(i)$ 
15          for all nodes  $k \in Adj(j)$ 
16             if  $k \in Acc(i)$ 
17                 delete  $k$  from  $Adj(j)$ 
18             end if
19      declare node  $i$  as visited
20  end PRUNE_ACC( $i$ )
```

In a slight deviation from what Corollary 2 suggests, line 15 cycles not over all nodes  $k \in Acc(j)$ , but only over  $k \in Adj(j)$ .

All nodes  $k \in Adj(j)$  are deleted from  $Adj(i)$  in lines 16-18. Cycling only over  $k \in Adj(j)$  saves time, but does not compromise the requirement that all nodes  $k \in Adj(i)$  be removed, because line 14 covers all nodes  $j$  accessible from  $i$ .

Because of the equality proven in Theorem 2, once this has been done, the adjacency list need not be modified further. This is why upon leaving this routine, the calling node is declared as visited.

Notice also that if a node  $j$  with  $Acc(j) = \epsilon$  is encountered, the loop in line 15 is not executed.

```
1  for all nodes  $i$  of  $G$ 
2     $Adj(i) = Acc(i)$ 
3  for all nodes  $i$  of  $G$ 
4    if node  $i$  has not been visited
5      call PRUNE_ACC( $i$ )
6    end if
7  PRUNE_ACC( $i$ )
8    for all nodes  $j \in Acc(i)$ 
9      if  $Acc(j) = \emptyset$ 
10         declare  $j$  as visited.
11      else
12         call PRUNE_ACC( $j$ )
13      end if
14    for all nodes  $j \in Acc(i)$ 
15      for all nodes  $k \in Adj(j)$ 
16        if  $k \in Acc(i)$ 
17          delete  $k$  from  $Adj(i)$ 
18        end if
19    declare node  $i$  as visited
20  end PRUNE_ACC( $i$ )
```

```

1   for all nodes  $i$  of  $G$ 
2       if  $component[i]$  has not been defined
3           create new node  $x$  of  $G^*$ 
4            $component[i]=x$ 
5           for all nodes  $j \in Acc(i)$ 
6               if  $i \in Acc(j)$ 
7                    $component[j]=x$ 
8               end if
9           end if

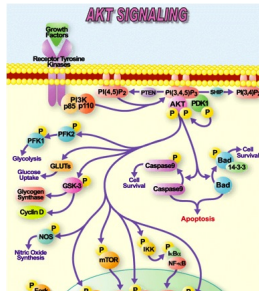
10  for all nodes  $i$  of  $G^*$ 
11       $Acc_{G^*}(i)=\emptyset$ 
12  for all nodes  $i$  of  $G$ 
13      for all nodes  $j \in Acc(i)$ 
14          if  $component[i] \neq component[j]$ 
15              if  $component[j] \notin Acc_{G^*}(component[i])$ 
16                  add  $component[j]$  to  $Acc_{G^*}(component[i])$ 
17              end if
18          end if

```

# Algorithms for Biological Networks

## Gillespie algorithm: The Biological problem

Many studies have reported occurrence of stochastic fluctuations and noise in living systems. Observation of gene expression in individual cells has clearly established the stochastic nature of transcription and translation. When using deterministic modeling approaches, for examples differential equations, we assume that the biological system evolves along a fixed path from its initial state. Such an approach cannot be taken for modeling stochastic processes such as gene regulation. Also, using deterministic methods, it is not possible to capture emergent phenomena that arise from inherent randomness. The Gillespie algorithm has been applied to many in silico biological simulations recently.



## Gillespie algorithm

Consider a system of  $N$  molecular species  $S_1, \dots, S_N$  interacting through  $M$  elemental chemical reactions  $R_1, \dots, R_M$ .

We assume that the system is confined to a constant volume  $W$  and is well stirred and at a constant temperature. Under these assumptions, the state of the system can be represented by the populations of the species involved.

We denote these populations by  $X(t) = (X_1(t), \dots, X_N(t))$ , where  $X_i(t)$  is the number of molecules of species  $S_i$  in the system at time  $t$ . The well stirred condition is crucial. For each reaction  $R_j$ , we define a propensity function  $a_j$ , such that  $a_j(x)dt$  is the probability, given  $X(t) = x$ , that one  $R_j$  reaction will occur in time interval  $[t, t + dt)$ . State change vector  $v_j$ , whose  $i$ th component is defined by  $v_{j,i}$  the change in the number of  $S_i$  molecules produced by one  $R_j$  reaction.



The most important method to simulate a network of biochemical reactions is the Gillespie stochastic simulation algorithm (SSA)

- ▶ The Gillespie algorithm is widely used to simulate the behavior of a system of chemical reactions in a well stirred container
- ▶ The key aspects of the algorithm is the drawing of two random numbers at each time step, one to determine after how much time the next reaction will take place, the second one to choose which one of the reactions will occur.
- ▶ Each execution of the Gillespie algorithm will produce a calculation of the evolution of the system. However, any one execution is only a probabilistic simulation, and the chances of being the same as a particular reaction is vanishingly small.
- ▶ Therefore to garner any useful information from the algorithm, it should be run many times in order to calculate a stochastic mean and variance that tells us about the behaviour of the system.
- ▶ the complexity of the Gillespie algorithm is  $O(M)$  where  $M$  is the number of reactions.

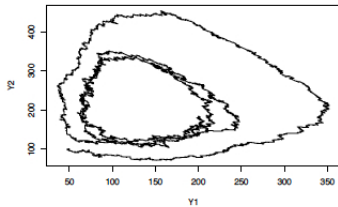
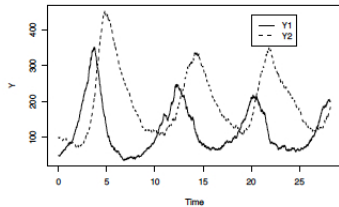
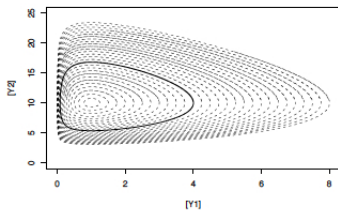
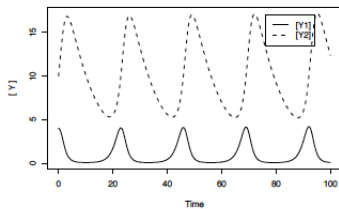
## Gillespie Algorithm

1. Initialise: set the initial molecule copy numbers, set time  $t = 0$ .
2. Calculate the propensity function  $a_i$  for each reaction, and the total propensity according to equation  $a_0(x) \equiv \sum_{j=1}^M a_j(x)$ ,  $i = 1, \dots, M$ .
3. Generate two uniformly distributed random numbers  $r_1$  and  $r_2$  from the range (0, 1).
4. Compute the time  $\tau$  to the next reaction using equation 
$$\tau = \frac{1}{a_0(x)} \ln \left( \frac{1}{r_1} \right) .$$
5. Decide which reaction  $R_\mu$  occurs at the new time using equation  $r_2 > \sum_{k=1}^{\mu-1} a_k \dots \text{and} \dots r_2 < \frac{1}{a_0} \sum_{k=1}^{\mu-1} a_k$ .
6. Update the state vector  $v$  by adding the update vector :  
$$v(t + \tau) = v(t) + (\nu)_\mu$$
7. Set  $t = t + \tau$ . Return to step 2 until  $t$  reaches some specified limit  $t_{MAX}$ .

In each step, the SSA starts from a current state  $x(t) = x$  and asks two questions: When will the next reaction occur? We denote this time interval by  $\tau$ . When the next reaction occurs, which reaction will it be? We denote the chosen reaction by the index  $j$ . To answer the above questions, one needs to study the joint probability density function  $p(\tau, j | x, t)$  that is the probability, given  $X(t) = x$ , that the next reaction will occur in the infinitesimal time interval  $[t + \tau, t + \tau + dt]$ . The theoretical foundation of SSA is given by  $p(\tau, j | x, t) = a_j(x) \exp(-a_0(x)\tau)$ , where  $a_0(x) \equiv \sum_{j=1}^M a_j(x)$ . It implies that the time  $\tau$  to the next occurring reaction is an exponentially distributed random variable with mean  $1/a_0(x)$ , and that the index  $j$  of that reaction is the integer random variable with point probability  $a_j(x)/a_0(x)$ . The  $\tau$  is  $\tau = \frac{1}{a_0(x)} \ln\left(\frac{1}{r_1}\right)$ .

The system state is then updated according to  $X(t + \tau) = x + \nu_j$  and this process is repeated until the simulation final time or until some other terminating condition is reached.

## Example: ODE versus Gillespie



## References and acknowledgements

- <http://www.thomas-schlitt.net/Bioproject.html>;
- <http://www.biostat.wisc.edu/craven/hunter.pdf>
- G. Church Y Gao, S Kosuri Next-Generation Digital Information Storage in DNA Science 22 September 2012
- N. Jones, P. Pevzner An Introduction to Bioinformatics Algorithms
- T.F. Smith, M.S. Waterman, Identification of common molecular subsequences, J Mol Biol 147, 195-197, 1981.
- affine gaps:  
<http://courses.cs.washington.edu/courses/cse527/00wi/lectures/lect05.ppt>
- D.S. Hirschberg, A linear space algorithm for computing longest common subsequences, Communications of the ACM 18, 341-343, 1975; <http://drp.id.au/align/2d/AlignDemo.shtml>.
- Nussinov, R., Pieczenik, G., Griggs, J. R. and Kleitman, D. J. (1978). Algorithms for loop matchings, SIAM J. Appl. Math
- Altschul, S.F. and Gish, W. (1996) "Local alignment statistics." Meth. Enzymol. 266:460-480 - Altschul, S.F., et al. (1990) "Basic local alignment search tool." J. Mol. Biol. 215:403-410

## References and acknowledgements

- Li, H and Durbin, R (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* 25:1754-60.
- Compeau P, Pevzner P and Tesler G. How to apply de Bruijn graphs to genome assembly. *Nature Biotechnology* 29: 987 2011
- Schatz, M., Delcher, A. and Salzberg, S. *Genome Res.* 20, 1165-1173 (2010).
- UPGMA:  
<http://www.southampton.ac.uk/re1u06/teaching/upgma/>
- Gascuel O. and Steel M. Neighbor-Joining Revealed *Molecular Biology and Evolution* 2006  
<http://mbe.oxfordjournals.org/content/23/11/1997.full.pdf>
- Atteson K. 1999. The performance of the neighbor-joining methods of phylogenetic reconstruction. *Algorithmica* 25:251-78.
- <http://www.cs.princeton.edu/mona/Lecture/phylogeny.pdf>
- MCL: <http://micans.org/mcl/ani/mcl-animation.html>
- Enright AJ, Van Dongen S, Ouzounis CA. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res.* 2000;28:1575-84.

## References and acknowledgements

- Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison

[http://books.google.co.uk/books/about/Biological\\_Sequence\\_Analysis.htm](http://books.google.co.uk/books/about/Biological_Sequence_Analysis.htm)

- Viterbi: <http://www.cs.umb.edu/~sreivilak/viterbi/>

- Lenwood S. Heath Naren Ramakrishnan (Eds) Problem Solving Handbook in Computational Biology and Bioinformatics

- <http://www.bio.davidson.edu/courses/genomics/chip/chip.html>

- Lawrence, Altschul, Boguski, Liu, Neuwald, Wootton, Detecting Subtle Sequence Signals: a Gibbs Sampling Strategy for Multiple Alignment Science, 1993

- A. Wagner Bioinformatics 17, 2001

- Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics

## Software and interesting websites for practicals

- Data Repository: <http://www.ncbi.nlm.nih.gov/> ; Human Genome Browser Gateway <http://genome.ucsc.edu/> [www.ensembl.org](http://www.ensembl.org) ; <http://www.ebi.ac.uk>
- Progressive alignment:  
<http://www.ebi.ac.uk/Tools/msa/clustalw2/>;
- Phylogenetic software:  
<http://evolution.genetics.washington.edu/phylip/software.html>
- HMM: <http://www.cbs.dtu.dk/services/TMHMM/>
- <http://genes.mit.edu/GENSCAN.html>
- Gibbs sampling [http://bayesweb.wadsworth.org/cgi-bin/gibbs.8.pl?data\\_type=DNA](http://bayesweb.wadsworth.org/cgi-bin/gibbs.8.pl?data_type=DNA)
- Various libraries to help with Biological data : [www.biojava.org](http://www.biojava.org);  
[www.bioperl.org](http://www.bioperl.org); [www.biopython.org](http://www.biopython.org); C++  
[www.ncbi.nlm.nih.gov/IEB/ToolBox/](http://www.ncbi.nlm.nih.gov/IEB/ToolBox/); Bioconductor



## Some papers with applications of the algorithms described here

- Venter JC et al. The sequence of the human genome. *Science*. 2001 Feb 16;291(5507):1304-51.
- International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*. 2004 Oct 21;431(7011):931-45.
- Hatem A, Bozda D, Toland AE, atalyrek V. Benchmarking short sequence mapping tools. *BMC Bioinformatics*. 2013 Jun 7;14:184. doi: 10.1186/1471-2105-14-184.
- Glazko GV, Nei M. Estimation of divergence times for major lineages of primate species. *Mol Biol Evol*. 2003 Mar;20(3):424-34.
- Enright AJ, Kunin V, Ouzounis CA. Protein families and TRIBES in genome sequence space. *Nucleic Acids Res*. 2003 Aug 1;31(15):4632-8.
- Shah SP, McVicker GP, Mackworth AK, Rogic S, Ouellette BF. GeneComber: combining outputs of gene prediction programs for improved results. *Bioinformatics*. 2003 Jul 1;19(10):1296-7.

## Figures acknowledgement (also orally during the lecture; it refers to the numbers in the copies distributed during the lectures, i.e. 4 slides/pages)

- pag 3, Church et al, Next-Generation Digital Information Storage in DNA, 28 SEPTEMBER 2012 VOL 337 - pag 4 from Bower and Boulori Computational modeling of genetic and biochemical networks, MIT Press, from Shalini Venkataraman and Vidhya Gunaseelan CS 594: An Introduction to Computational Molecular Biology, Nir Friedmans and Shlomo Moran lectures. - pag 5 H Koepl's Lecture at ETH. - pag 11 from Pevzner P Computational Molecular Biology: An Algorithmic Approach, MIT Press. - pag 14 from Volker Sperschneider Bioinformatics: Problem Solving Paradigms, springer. - pag 18,19 from Ming Li, University of Waterloo. - pag 21,22 from Wall lab website, Harvard. - pag 23,24 from Compeau Nature Biotechnology 29, 987991, (2011). - pag 22,26 from Alicia Clum, DOE Joint Genome Institute. - pag 33 from G Caldarelli at the European Complex Systems Conference. - pag 34 from paper on TribeMCL (see previous slide). - pag 36,37,38 from Nir Friedmans and Shlomo Moran lectures at [www.cs.huji.ac.il](http://www.cs.huji.ac.il). - pag 40, 41 from C. Burge and S. Karlin Prediction of Complete Gene Structures in Human Genomic DNA. J. Mol. Biol. (1997) 268, 78-94. - pag 42,43 Chris Burge's lecture on DNA Motif Modeling and Discovery at MIT. - pag 52 from Wilkinson Stochastic Modeling, CRC Press.

## Examples of Exam Questions

- ▶ Align the two strings: ACGCTG and CATGT, with match score =1 and mismatch, gap penalty equal -1
- ▶ Describe with one example the difference between Hamming and Edit distances
- ▶ Discuss the complexity of an algorithm to reconstruct a genetic network from microarray perturbation data
- ▶ Discuss the properties of the Markov clustering algorithm and the difference with respect to the k-means and hierarchical clustering algorithms

**Examples of Answers** Align the two strings: ACGCTG and CATGT, with match score =1 and mismatch, gap penalty equal -1

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

Describe with one example the difference between Hamming and Edit distances  $TGCATAT \rightarrow ATCCGAT$  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).

## Examples of Answers

Discuss the complexity of an algorithm to reconstruct a genetic network from microarray perturbation data

Reconstruction:  $O(nka)$  where  $n$  is the number of genes,  $k$  is the average number of entries in the accession list;  $a$  is the average number of entries in adjacency list. Large scale experimental gene perturbations in the yeast *Saccharomyces cerevisiae* ( $n=6300$ ) suggests that  $k < 50$ ,  $a < 1$ , and thus that  $nka \ll n^2$ .

Discuss the properties of the Markov clustering algorithm and the difference with respect to the k-means and hierarchical clustering algorithms

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

The k-means algorithm is composed of the following steps: 1) Place  $K$  points into the space represented by the objects that are being clustered. These points represent initial group centroids. 2) Assign each object to the group that has the closest centroid. 3) When all objects have been assigned, recalculate the positions of the  $K$  centroids. 4) Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Hierarchical clustering: Start with each point its own cluster. At each iteration, merge the two clusters; with the smallest distance. Eventually all points will be linked into a single cluster. The sequence of mergers can be represented with a rooted tree.