Bioinformatics

Computer Laboratory

Computer Science Tripos, Part II

Michaelmas Term 2014

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).

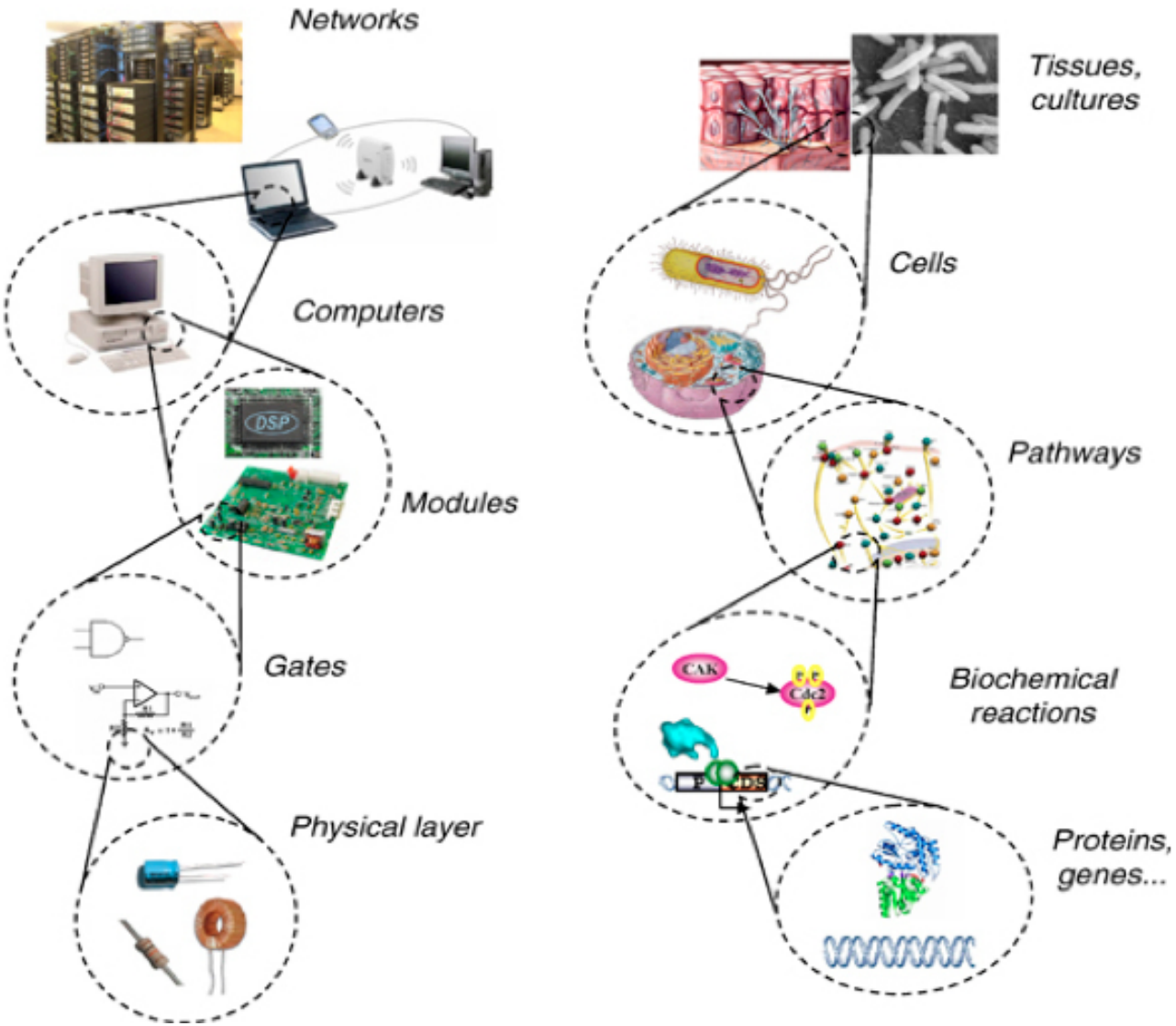
Overview of the course

First we provide an overview of the most important biological concepts. Then we learn how to compare 2 strings representing DNA sequences (or different parts of the same string). Searching a database for nearly exact matches is a key task in a Bioinformatics lab. The big efforts in sequencing human genomes and also single cell genomes require new algorithms to deal with big sequence data. We learn how to build trees to study sequences relationship and how to cluster biological data. We use hidden Markov models to predict properties of sequence parts such as exon/intron arrangements in a gene or the structure of a membrane protein. Patterns dispersed in sequences could be identified by iterated techniques. Then we study how to reconstruct genetic networks from data. A large set of biochemical reactions could be simulated by using an algorithm. Material and figure acknowledgements at the end of this notes and during the lectures.

Topics and List of algorithms

- Basic concepts in Genetics-Genomics.
- Dynamic programming (Longest Common Subsequence, Needleman-Wunsch, Smith-Waterman, Hirschberg, Nussinov RNA folding).
- Homology database search (Blast, Patternhunter).
- Progressive alignment (Clustal).
- Genome Assembly (Burrows-Wheeler transform)
- Next Generation sequencing (De Bruijn graph)
- 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 sequence alignment (Gibbs sampling).
- Biological Networks reconstruction (Wagner) and simulation (Gillespie).

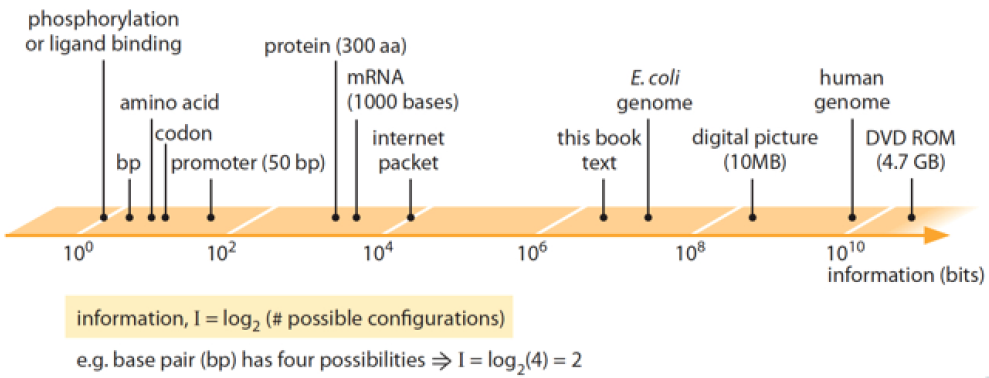
1 Comparison between system networks and biological systems

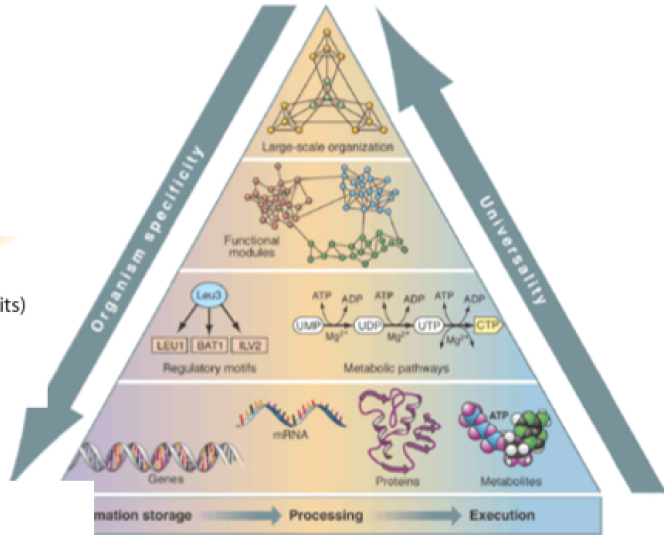


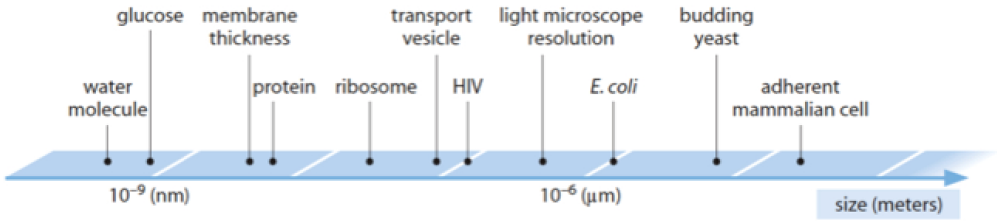
from Andrianantoandro et. al. Mol Syst Biol (2006)

(what is in the previous figure)

from bottom: Molecules (for example genes (strings of bases) and proteins (strings of amino acids)), reactions (modification of molecules), pathways (networks of reactions), cells (networks of pathways), tissues (networks of cells), organs (networks of tissues), organisms (networks of organs?). The figures below (from http://bionumbers.hms.harvard.edu/ and Oltvai and Barabasi) make the point of biological numeracy and that there is plenty of variability at the ”bottom, i.e. lower scales”.



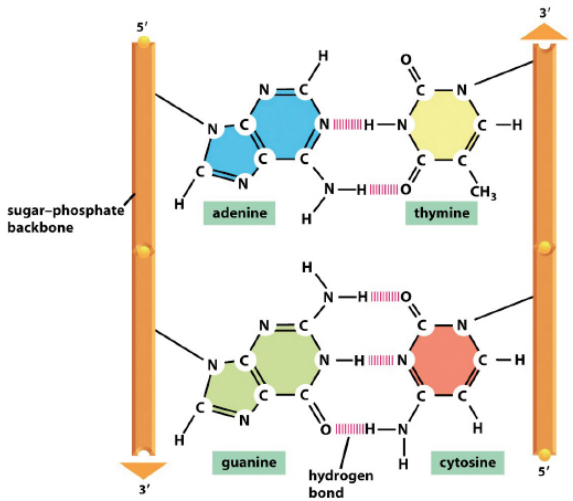




Basic concepts in genetics

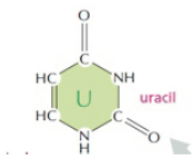
DNA could be thought as a string of symbols from a 4-letter (bases) alphabet, A (adenine), T (thymine), C (cytosine) and G (guanine). In the double helix A pairs with T, C with G (so only the sequence of one filament is vital to keep). A gene is a string of DNA that contains information for a specific 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 graph. The genetic code is a map between 61 triplets of DNA bases and 20 amino acids.



A to T

G to C



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

Comparing CS and Biological information

from www.nsta.org/publications/news/story.aspx?id=47561

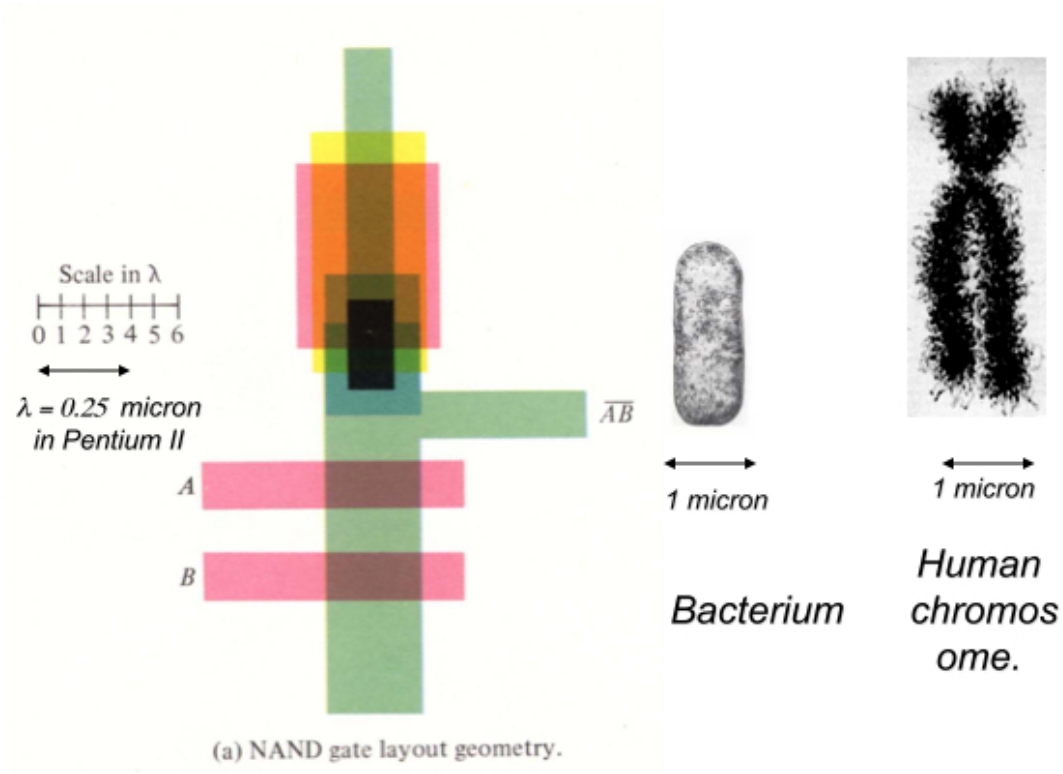
Biology	Computer science
1. Digital alphabet consists of bases A, C, T, G	1. Digital alphabet consists of 0, 1
2. Codons consist of three bases	2. Computer bits form bytes
3. Genes consist of codons	3. Files consist of bytes
4. Promoters indicate gene locations	4. File-allocation table indicates file locations
5. DNA information is transcribed into hnRNA and processed into mRNA	5. Disc information is transcribed into RAM
6. mRNA information is translated into proteins	6. RAM information is translated onto a screen or paper
7. Genes may be organized into operons or groups with similar promoters	7. Files are organized into folders
8. "Old" genes are not destroyed; their promoters become nonfunctional	8. "Old" files are not destroyed; references to their location are deleted
9. Entire chromosomes are replicated	9. Entire discs can be copied
10. Genes can diversify into a family of genes through duplication	10. Files can be modified into a family of related files
11. DNA from a donor can be inserted into host chromosomes	11. Digital information can be inserted into files
12. Biological viruses disrupt genetic instructions	12. Computer viruses disrupt software instructions
13. Natural selection modifies the genetic basis of organism design	13. Natural selection procedures modify the software that specifies a machine design
14. A successful genotype in a natural population outcompetes others	14. A successful website attracts more "hits" than others

Useful concepts : Strings, polymers, graphs

Unit: DNA base (A (adenine), T (thymine), C (cytosine) and G (guanine))
Polymer: DNA molecule
Unit: RNA base (A (adenine), U (uracil), C (cytosine) and G (guanine)) Polymer: RNA molecule
Unit: amino acid (there are 20 amino acids) Polymer: the protein (a linear, unbranched chain of amino acids)
Polymers can be thought as strings (the information is the sequence of symbols) or as graph (the information is the 3 dimensional structure)
the strings undergo modifications of length n-units (mutations (base or amino acid replacement), insertions (adding more bases or amino acids), deletions (loss of n bases or amino acids)); strings algorithms on alignment, tree, searching for conserved motifs.
The graph topology determines the 3D connectivity with other graphs (forming networks).

DNA and chromosomes

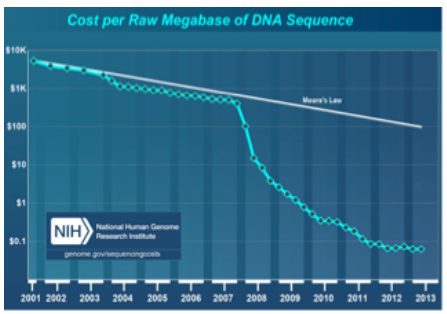
In eukaryotes the genetic information is distributed over different DNA molecules. A human cell contains 24 different such chromosomes. If all DNA of a human cell would be laid out end-to-end it would reach approximately 2 meters. The nucleus however measures only 6μm. Equivalent of packing 40 km of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

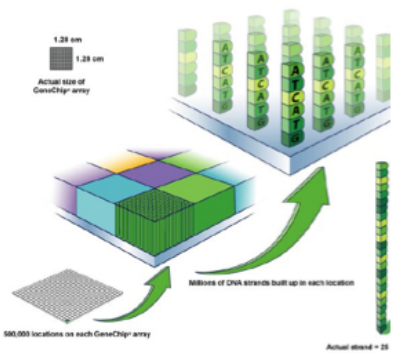


(a) NAND gate layout geometry.

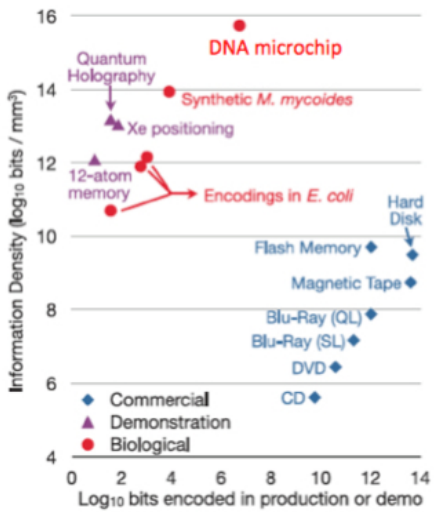
DNA is something we can read and write

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.



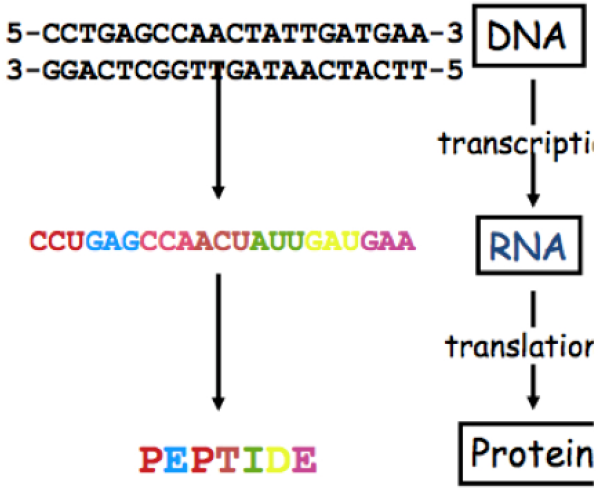


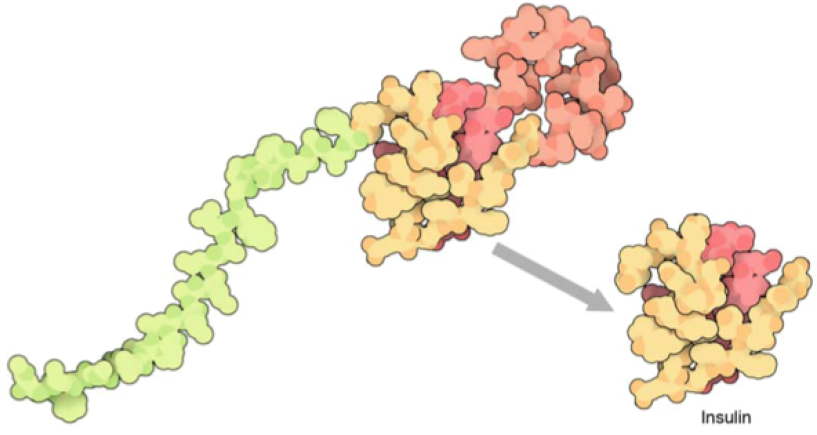
Left: DNA reading; bottom left: use of DNA sequences placed in DNA chip to write DNA or other information. Bottom right: advantage of using DNA chips.



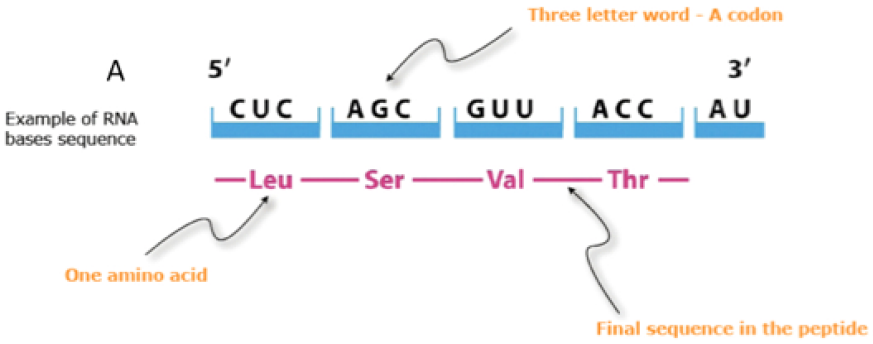
The flow of biological information

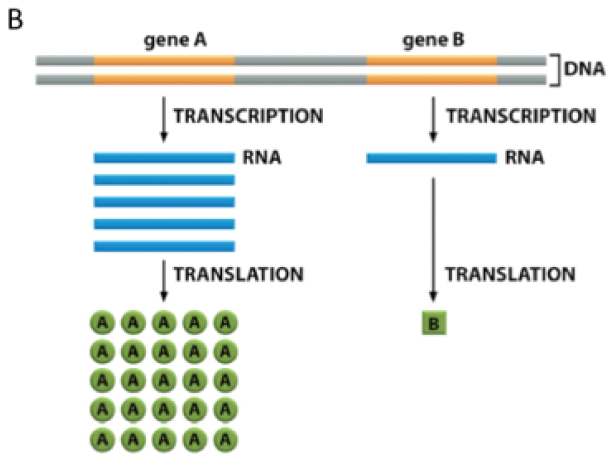
DNA makes RNA makes proteins (the 3D graph below); given the pairing rule in a DNA double strands molecule, all the information is in each single strand. The RNA is termed mRNA (messenger); triplets of bases of mRNA are copied into a chain of amino acids (the protein) according to the genetic code.

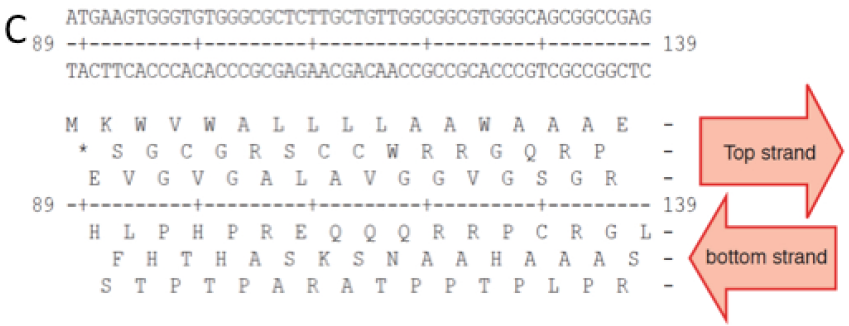




One gene (a set of bases with begin and end signals) contains information for one protein (at least)







A) Each triplet of bases codes for one amino acid.

B) Genes differ for the amount of messenger RNA and protein molecules they produce (variable among cells type, position and time regulation).

C) Potentially the DNA strands could code for 6 proteins.

The Genetic code: a mapping function between DNA and Proteins

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

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

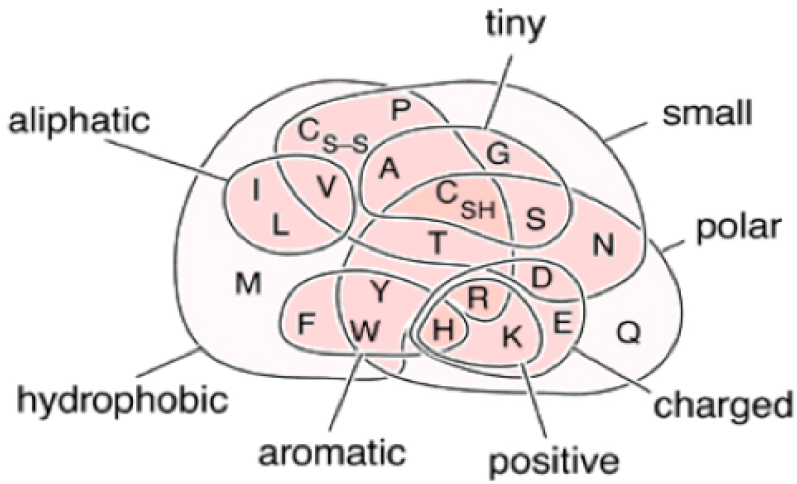


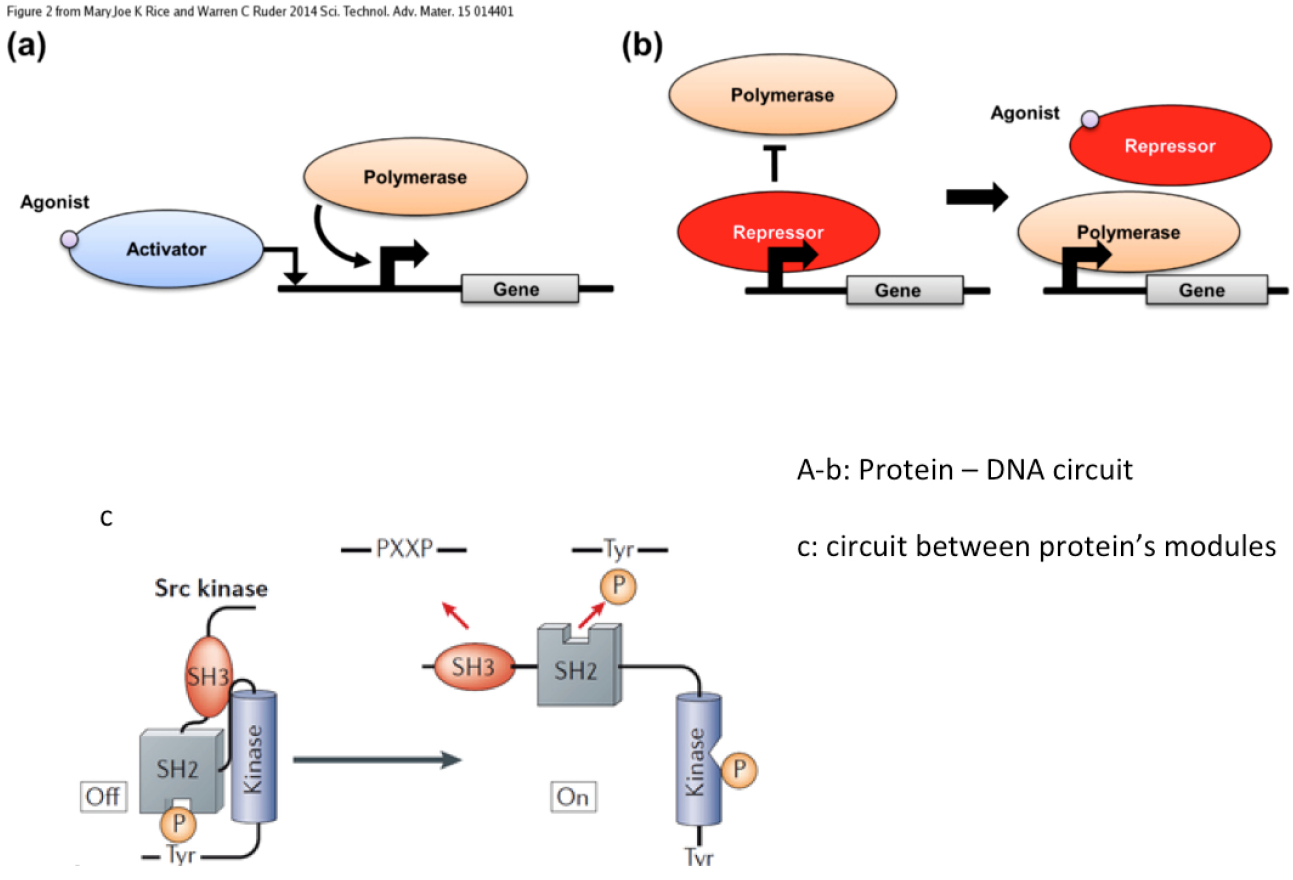
Figure : The genetic code provides the information for the translation of codons (triplets of bases, in black) into amino acids (single and triple letter code in red) that are chained together to form a protein; 61 codons code for 20 amino acids (differences on the right); 3 specific codons code a “stop” signal; note that C exists in two states.

Basic concepts in biocircuits

Genes are activated or repressed by regulatory proteins which are coded the same or other genes.
A chemical reaction converts chemical compounds (analogous to a production rule)
An enzyme is a protein that accelerates chemical reactions.
Each enzyme is encoded by one or more genes.
A pathway is a linked set of reactions within a cell (analogous to a chain of rules)
A pathway is a conceptual unit of the metabolism; it represents an ordered set of interconnected, directed biochemical reactions
The set of metabolic pathways makes the metabolic network which makes the cell phenotype.

The structure of biocircuits

The top figure shows the regulation of genes through proteins acting as activators (a) and repressors (b); the bottom figure shows that the regulation of proteins could occur via activation or inactivation of other proteins or parts of the same protein. Large scale network of circuits can be designed (see for instance http://2014.igem.org/)

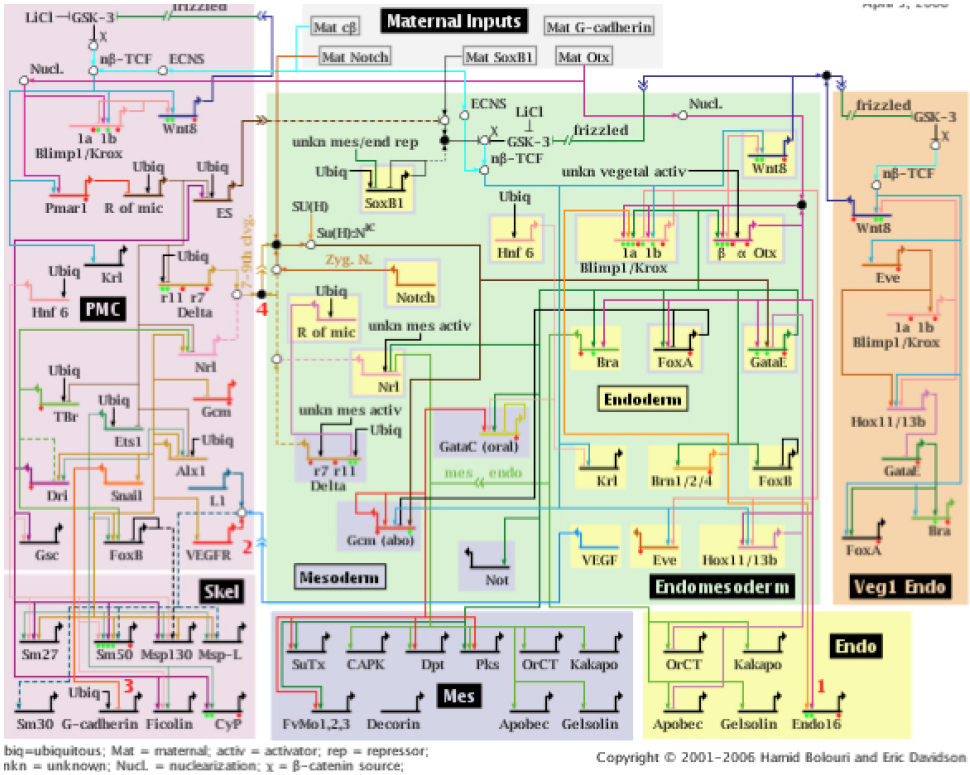


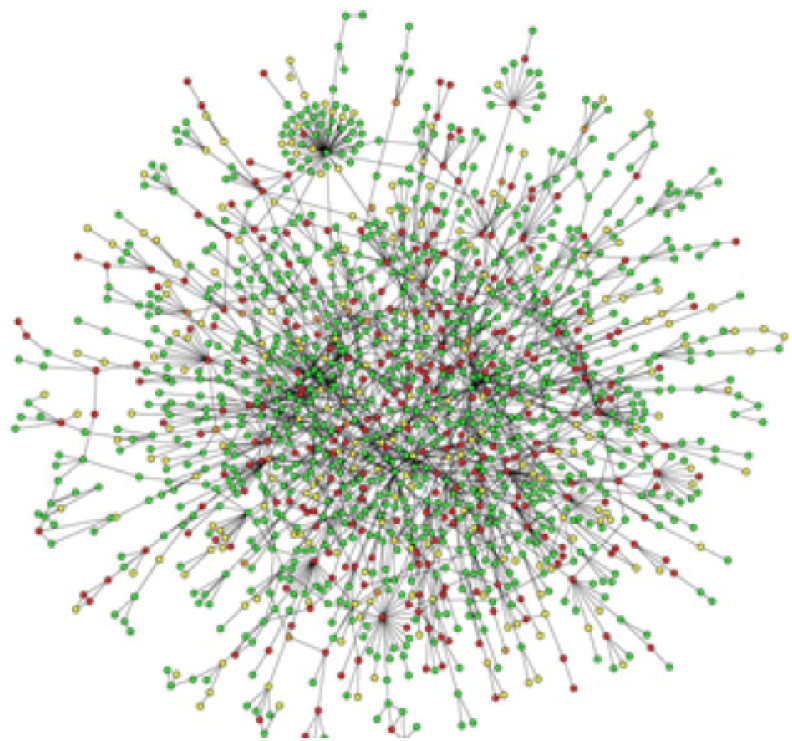
A-b: Protein – DNA circuit

c: circuit between protein's modules

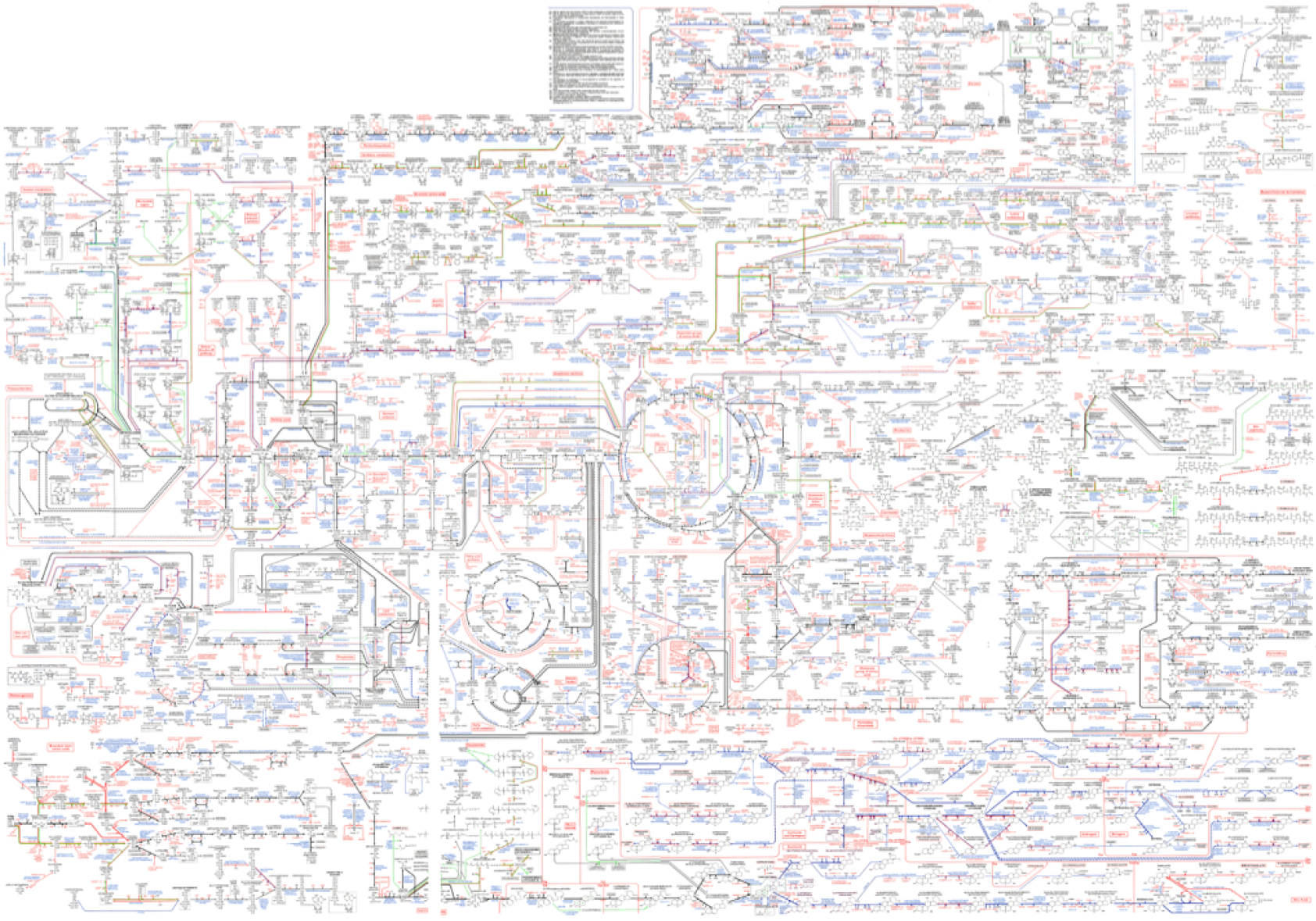
Genes form networks

Cells contain large gene and protein networks (left; from Bolouri). Gene connectivity could be studied (from Barabasi); some genes could be more important than others (red circles in the figure in the right)





Pathways form (metabolic) reaction Networks (from expasy.org)

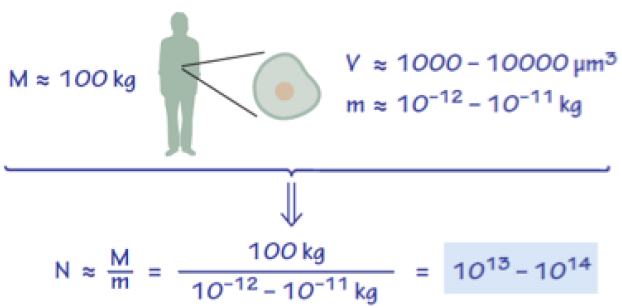


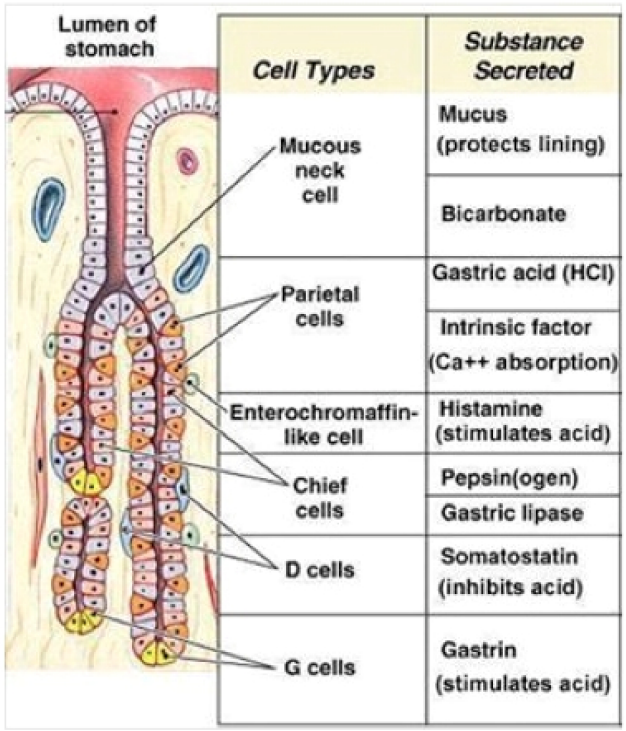
Cells make networks (tissues and organs)

A human cell (in average it measures $10\mu m$ across); the stomach tissue (figure below) is a network of cells

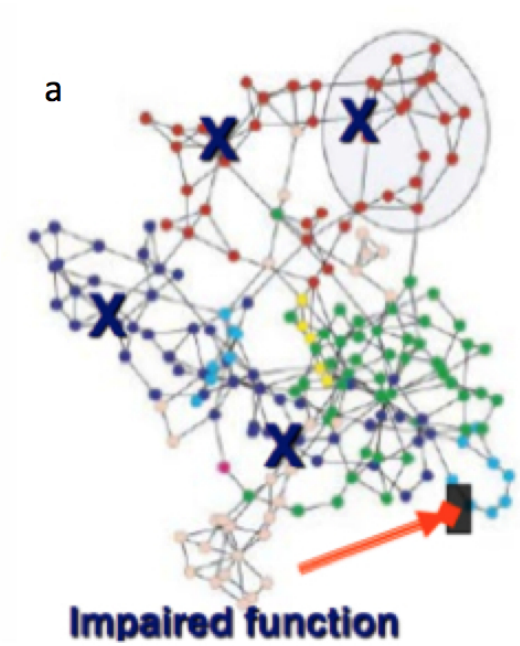
Organism	Stage in life cycle or organ	Estimated cell count
Human	Adult	$3.7\pm0.8\times10^{13}$
Human	Fat cells – lean adult	4×10^{10}
Human	Fat cells – obese adult	10^{11}
Drosophila melanogaster	Embryo cycle 14	6,000 (nuclei)
Drosophila melanogaster	Wing disk at metamorphosis	50,000
Arabidopsis Thaliana	Leaf layer	20,000 (leaf 1) to 100,000 (leaf 6)
C. elegans	Adult male (somatic)	1031

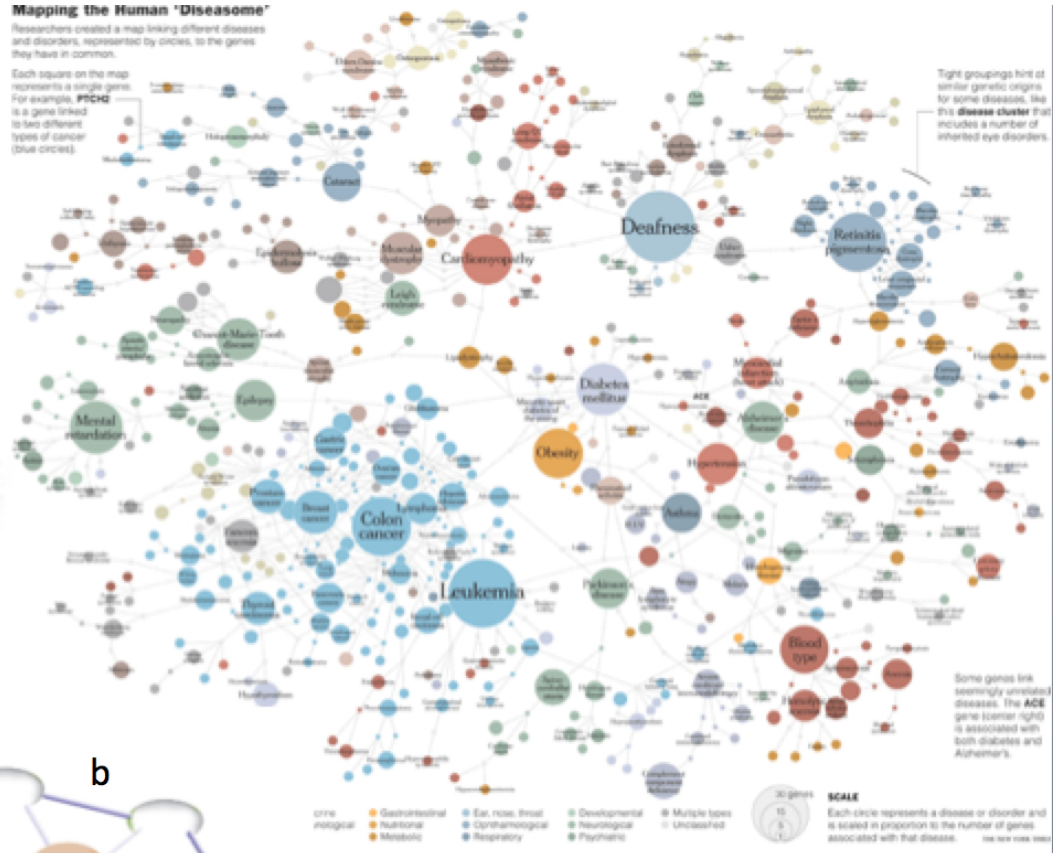
number of cells in the human body



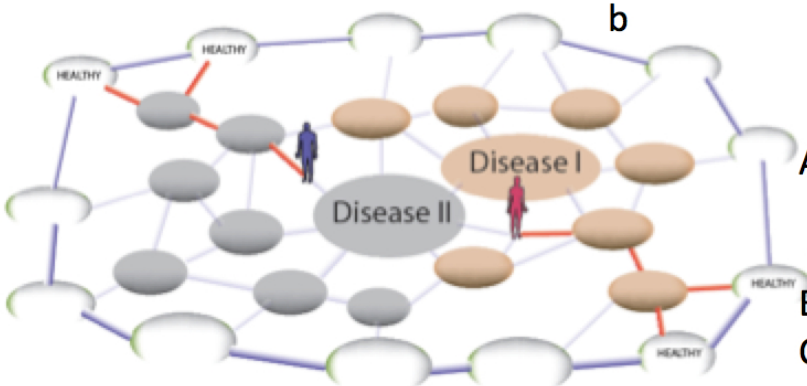


Diseases form networks





b



c

A: Cancer, heart disease, obesity, diabetes are managed at best, not cured because they are network diseases

B: disease map from www.diseasome.eu

C: patient disease risk map

Comparing regulatory networks in genomes and Operating Systems

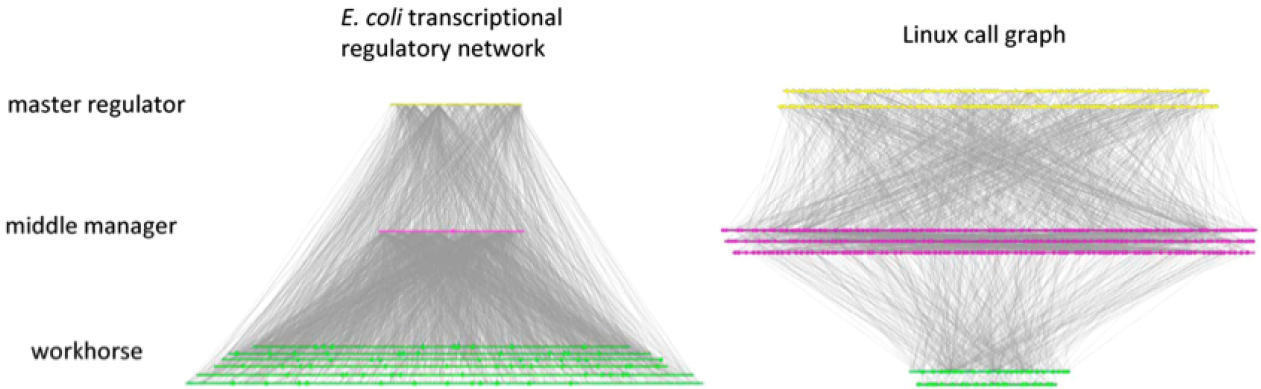
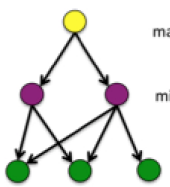


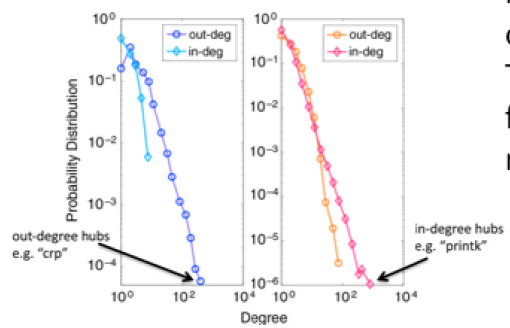
Fig. 1. The hierarchical layout of the *E. coli* transcriptional regulatory network and the Linux call graph. (*Left*) The transcriptional regulatory network of *E. coli*. (*Right*) The call graph of the Linux Kernel. Nodes are classified into three categories on the basis of their location in the hierarchy: master regulators (nodes with zero in-degree, *Yellow*), workhorses (nodes with zero out-degree, *Green*), and middle managers (nodes with nonzero in- and out-degree, *Purple*). Persistent genes and persistent functions (as defined in the main text) are shown in a larger size. The majority of persistent genes are located at the workhorse level, but persistent functions are underrepresented in the workhorse level. For easy visualization of the Linux call graph, we sampled 10% of the nodes for display. Under the sampling, the relative portion of nodes in the three levels and the ratio between persistent and nonpersistent nodes are preserved compared to the original network. The entire *E. coli* transcriptional regulatory network is displayed.

A



	percentage in E. coli regulatory network	percentage in Linux call graph
master regulator	4.6	29.6
middle manager	5.1	58.2
workhorse	90.2	12.3

B



The transcriptional regulatory network (1,378 nodes) follows a conventional hierarchical picture, with a few top regulators and many workhorse proteins. The Linux call graph (12,391 nodes), on the other hand, possesses many regulators; the number of workhorse routines is much lower in proportion. The regulatory network has a broad out-degree distribution but a narrow in-degree distribution. The situation is reversed in the call graph, where we can find in-degree hubs, but the out-degree distribution is rather narrow. Yan et al. PNAS 2010, 107, 20.

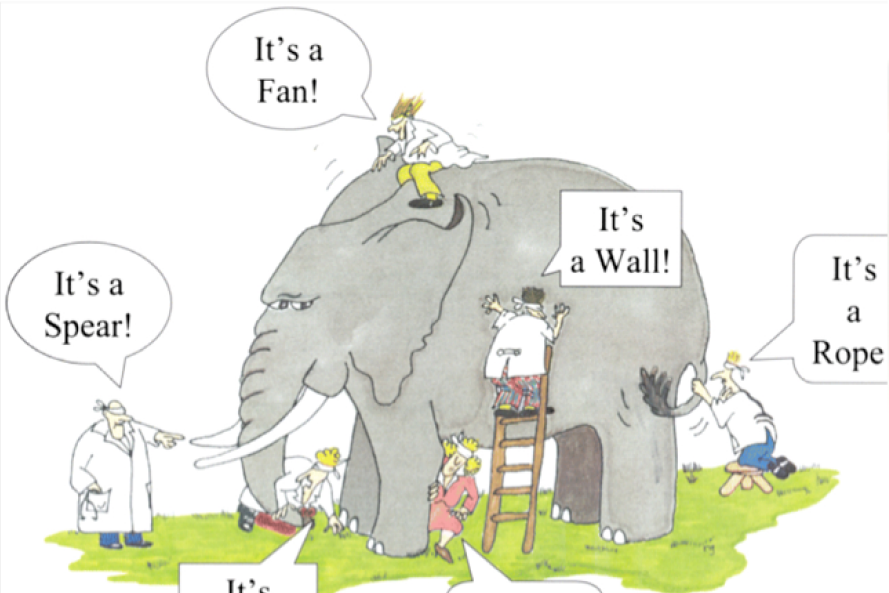
Data repositories (few examples to try):

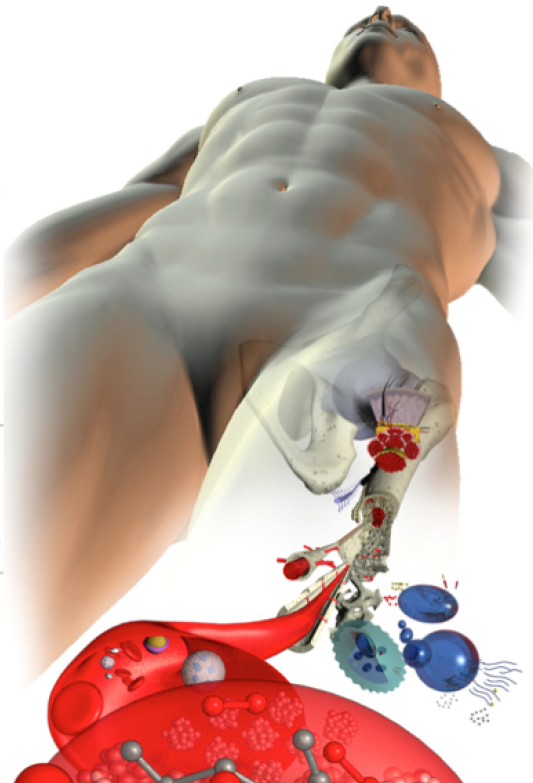
www.ebi.ac.uk
www.ensembl.org
www.ncbi.nlm.nih.gov/
www.ddbj.nig.ac.jp/ etc

1) strings for DNA and protein sequences (see
   a. http://www.ensembl.org/Homo_sapiens/Gene/Summary?db=core;g=ENSG00000260629;r=11:5244554-5245547;t=ENST00000564523
   b. http://www.ensembl.org/Homo_sapiens/Transcript/ProteinSummary?db=core;g=ENSG00000213934;r=11:5244554-5245547;t=ENST00000330597
   c. http://www.ncbi.nlm.nih.gov/nuccore/U01317.1
   d. http://www.ncbi.nlm.nih.gov/protein/AAA16334.1)
2) gene expression (intensity of gene activity) matrix (csv format) (see
   a. http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE53655
   b. http://www.genome.jp/kegg/expression/
3) 3d graphs for protein structure (see
   a. http://www.ncbi.nlm.nih.gov/Structure/mmdb/mmdbsrv.cgi?uid=122520
   b. http://prosite.expasy.org/PS01033 http://prosite.expasy.org/cgi-bin/pdb/pdb_structure_viewer.cgi?pdb=101M&ps=PS01033
4) medical literature
   ww.ncbi.nlm.nih.gov/pubmed/?term=human+beta+globin
5) metabolic pathways (see http://www.genome.jp/kegg/pathway.html)
6) Browsers (http://en.wikipedia.org/wiki/Genome_browser; http://www.ensembl.org/index.html; http://genome.ucsc.edu/)

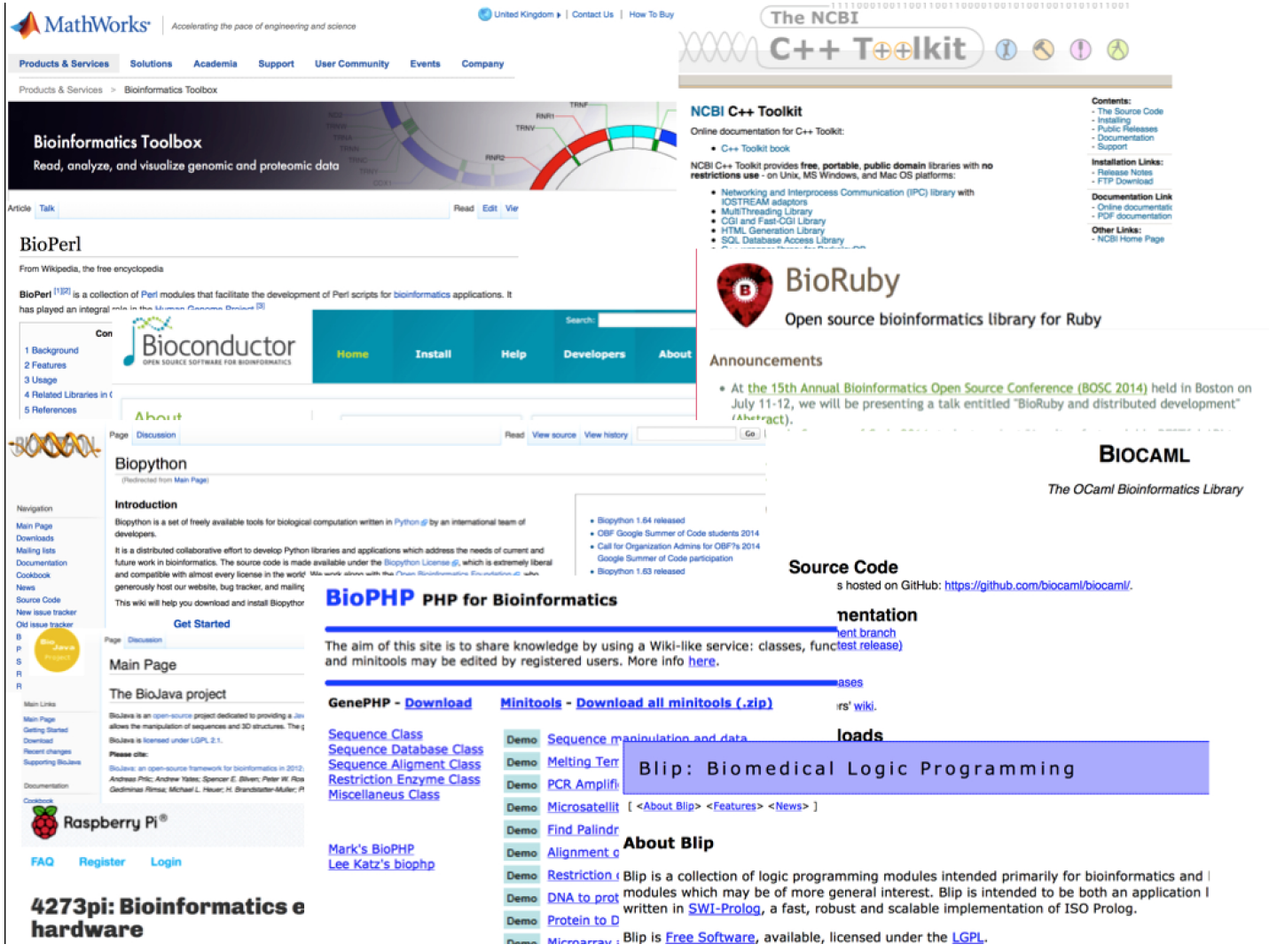
Bioinformatics future challenges: integrating data and scales

Often bioinformaticians are specialised in one type of biological data (for example sequence or gene expression data) or on biological processes at one specific scale. We can observe what happens at almost all scales, from the whole organism down to the molecular level; however, putting things together in order to obtain real understanding is much more difficult and less developed.



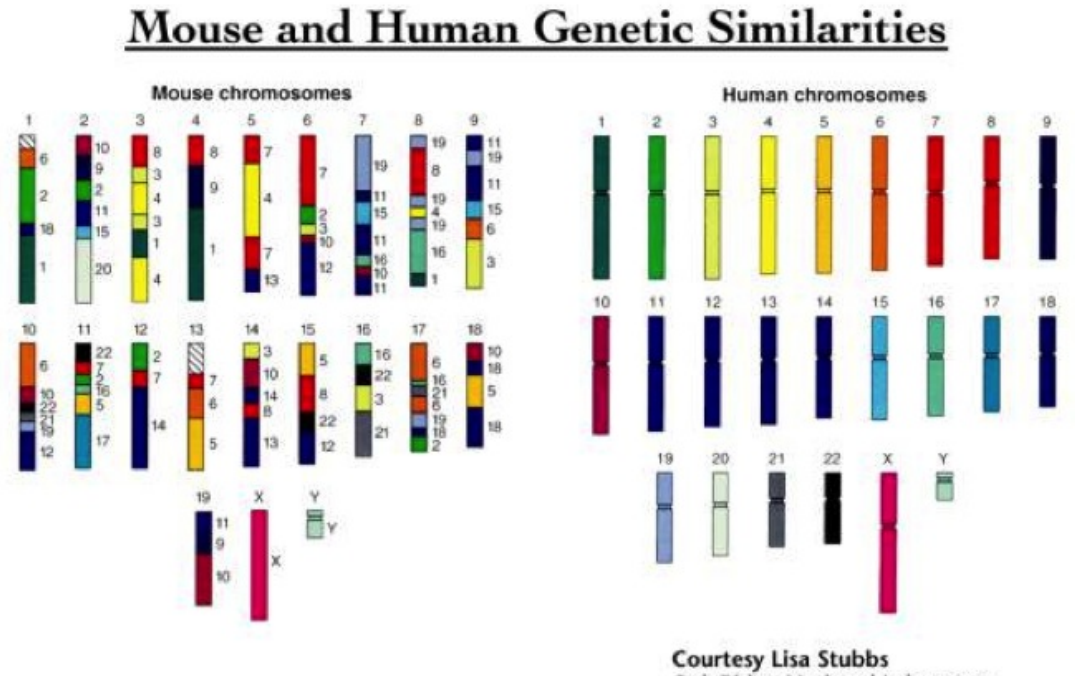


Bioinformatics library



2: Dynamic programming algorithms for sequence alignment

Typical tasks: align genome and protein sequences (below the results of the analysis of the differences between human and mouse genome sequences, each of 3 billion DNA bases); the task is to detect all differences at the single base to block of bases levels. In the RNA folding problem we want to align a molecule with itself. Algorithms in this lecture: Longest common subsequence, Needleman-Wunsch, Smith-Waterman, Affine gap, Hirschberg, Nussinov RNA folding.



2 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

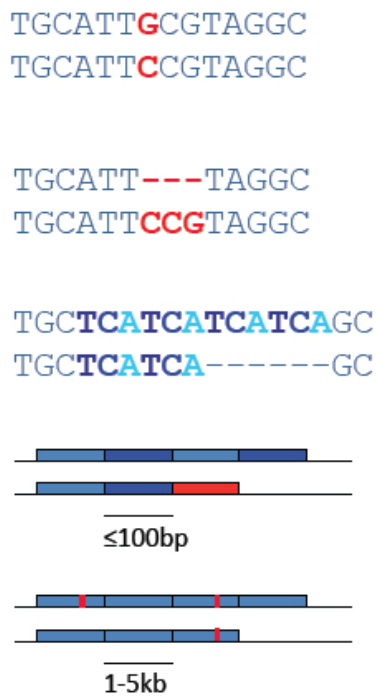


Figure : Type and frequency of mutations (replacements, insertions, deletions) in the human genome per generation; mutations change single DNA bases (SNP polymorphism) or rearrange DNA strings at different length scales. In sequence alignment we compare sequences that are different because of mutations.

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 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 $x = x_1, x_2, , x_M$, $y = y_1, y_2, , y_N$, an alignment is an assignment of gaps to positions $0, , M$ in x, and $0, , N$ in y, so as to line up each letter in one sequence with either a letter, or a gap in the other sequence.

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC

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

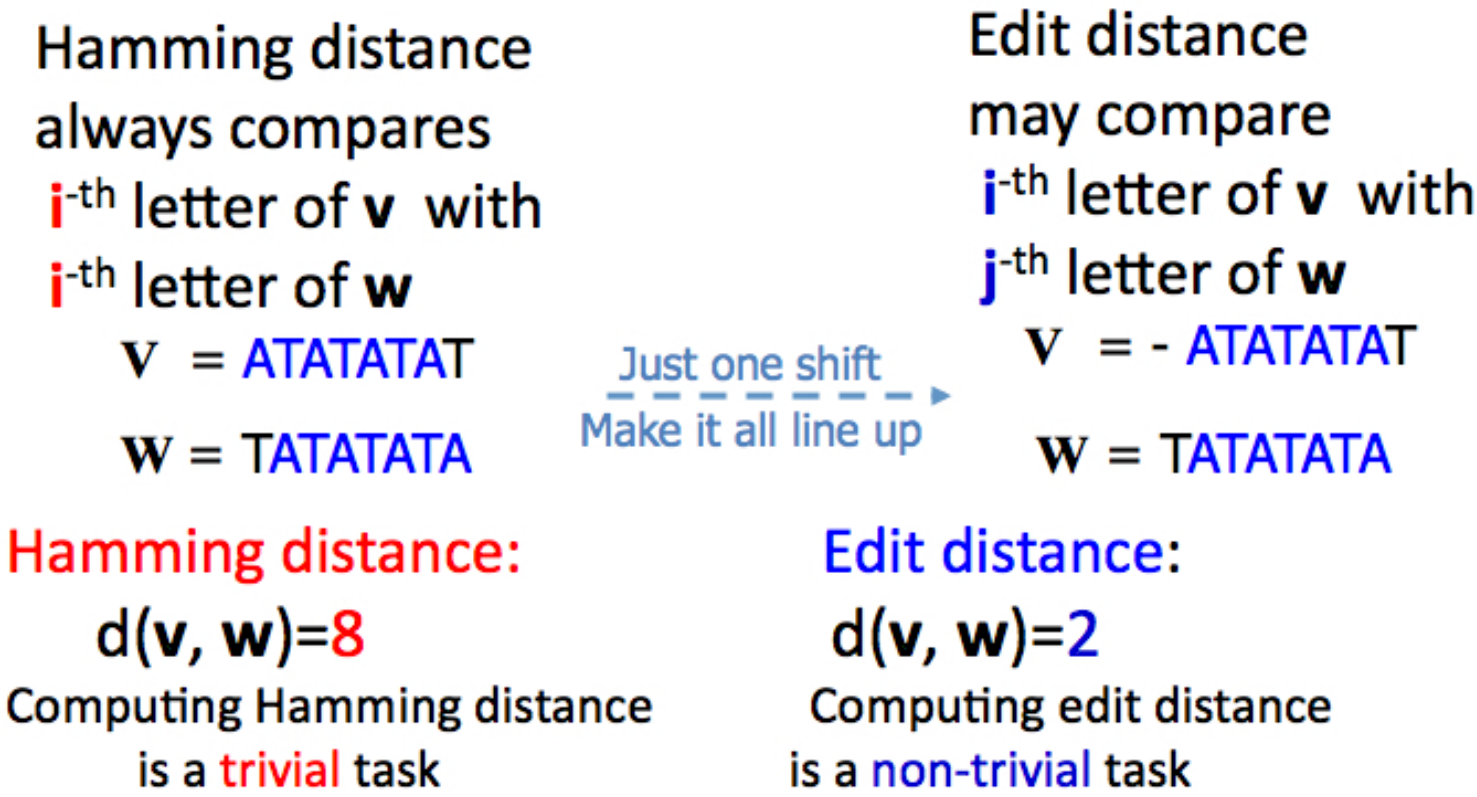
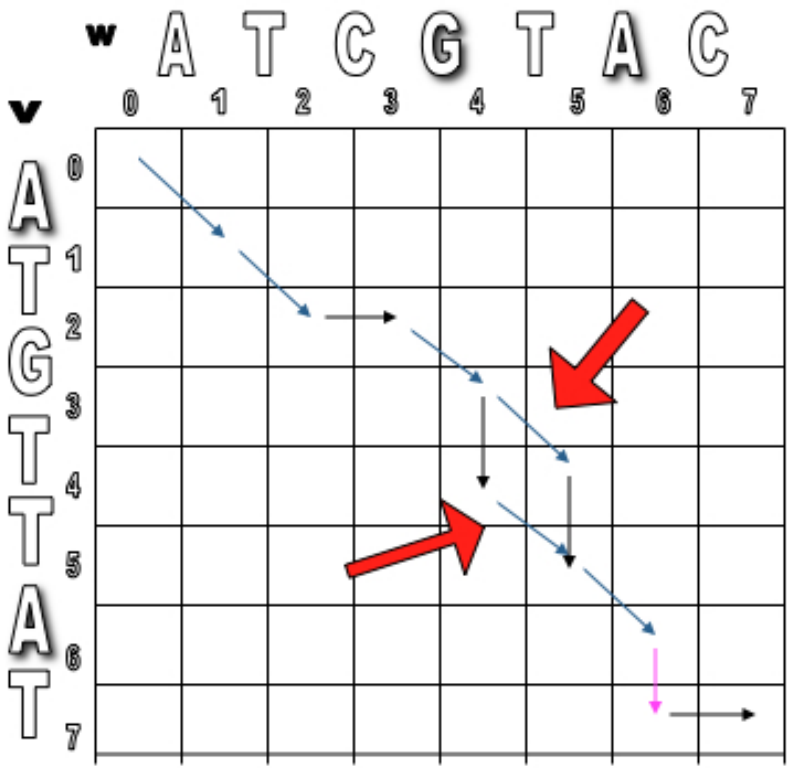


Figure : The Hamming distance is a column by column number of mismatches; the Edit distance between two strings is the minimum number of operations (insertions, deletions, and substitutions) to transform one string into the other.

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 ONE where the column and row site matches (diagonal segments), a 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.

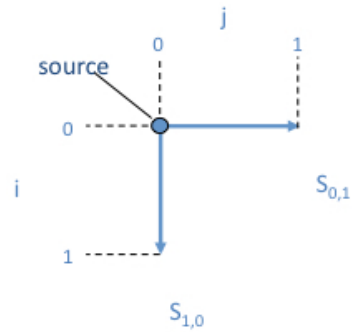
The Longest Common Subsequence (LCS)

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 =AAACCGTGAGTTATTCGTTCTAGAA
$v2$ =CACCCCTAAGGTACCTTTGGTTC
LCS is 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 $v = v_1\ v_2\ , v_m$ and $w = w_1\ w_2\ , w_n$. The LCS of v and w is a sequence of positions in v: $1 < i_1 < i_2 < < i_t < m$ and a sequence of positions in w: $1 < j_1 < j_2 < < j_t < n$ 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 (we will see that in DNA sequence alignment we use different scores for match, mismatch and gap).

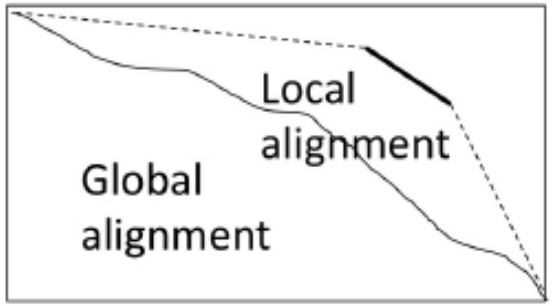
The Longest Common Subsequence




```
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
                       ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

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……M

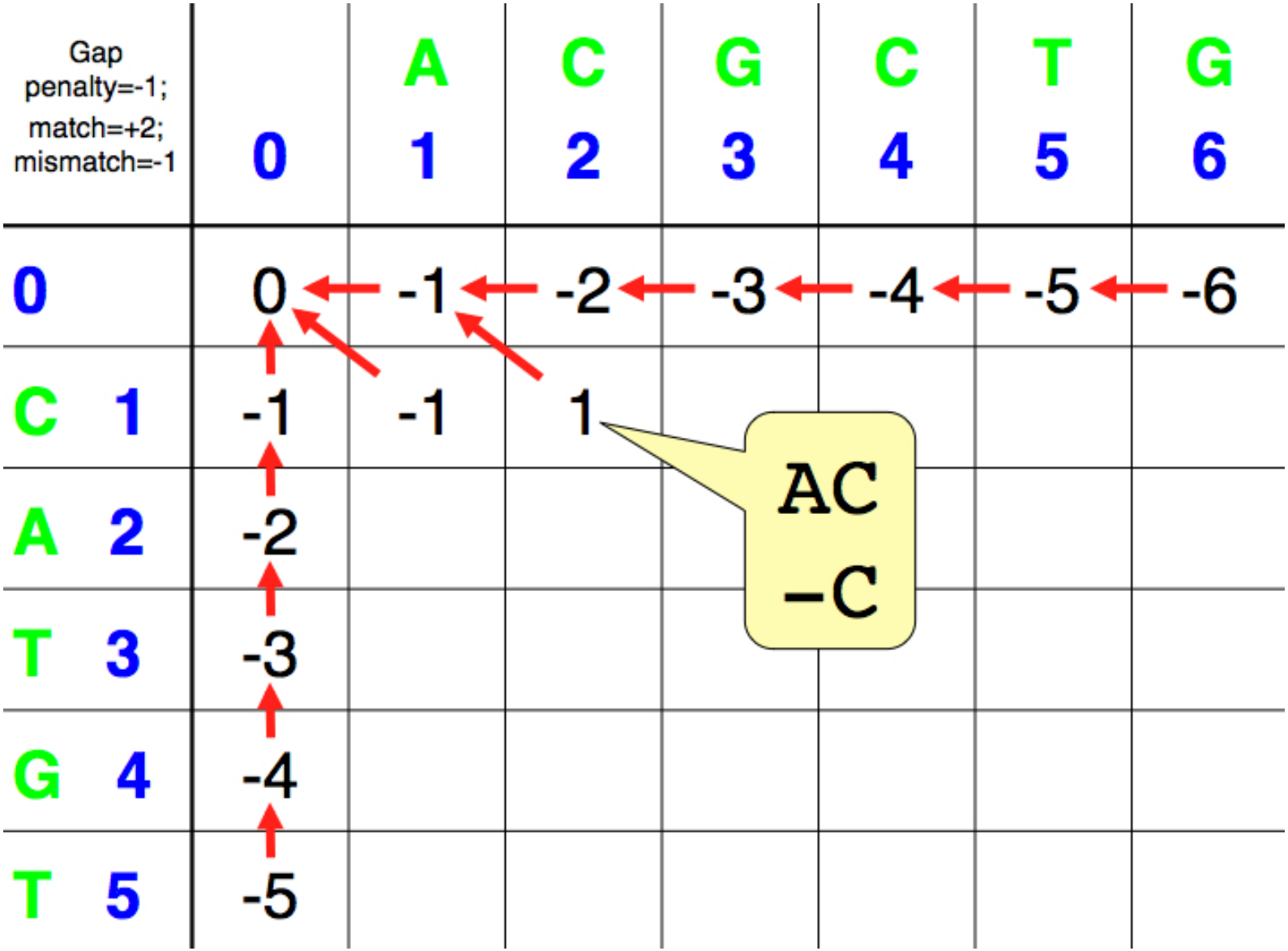
   For each j = 1……N

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

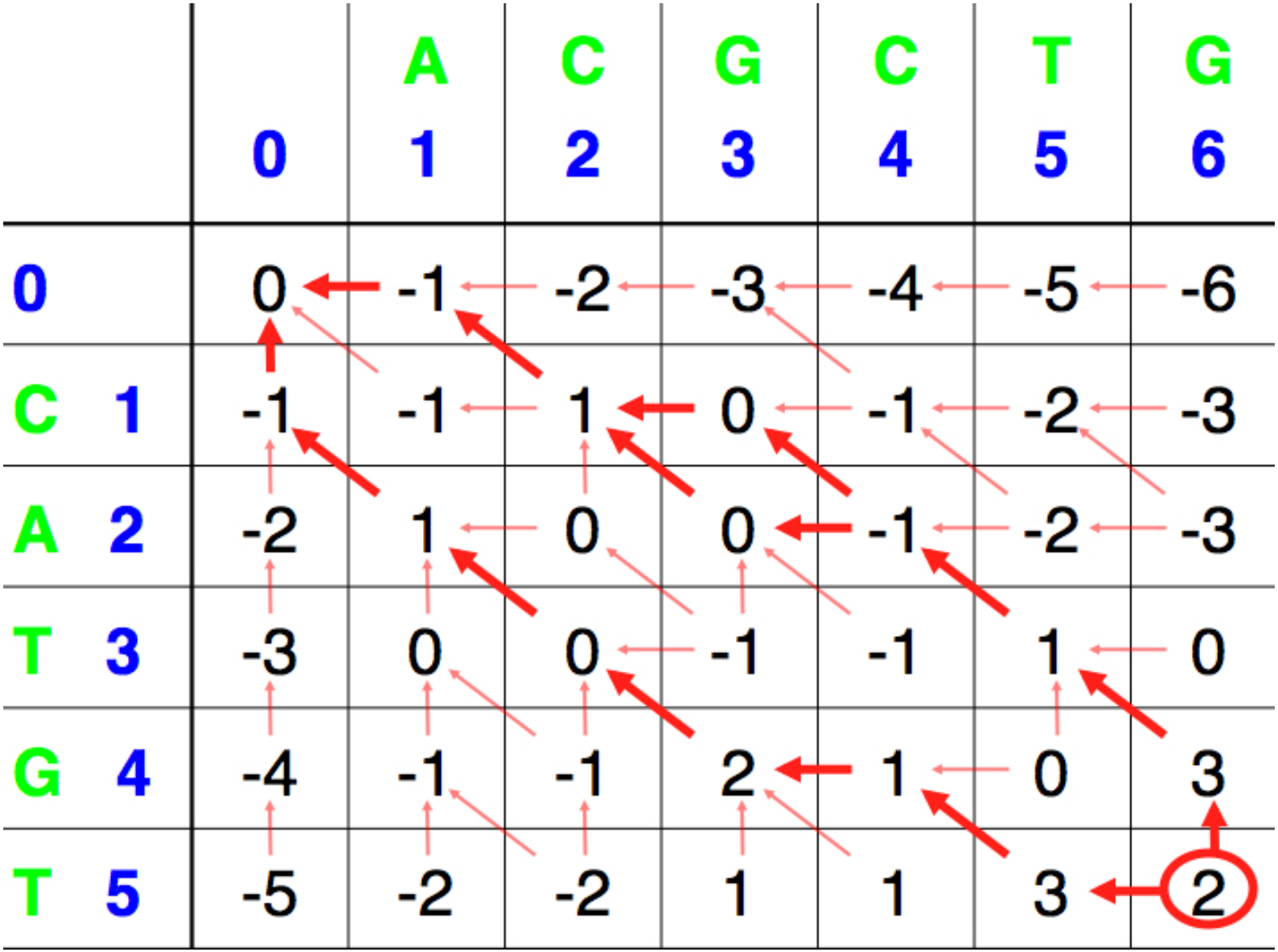
$$Ptr(i,j) = \begin{cases} UP, & \text{if [case 1]} \\ LEFT & \text{if [case 2]} \\ DIAG & \text{if [case 3]} \end{cases}$$

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

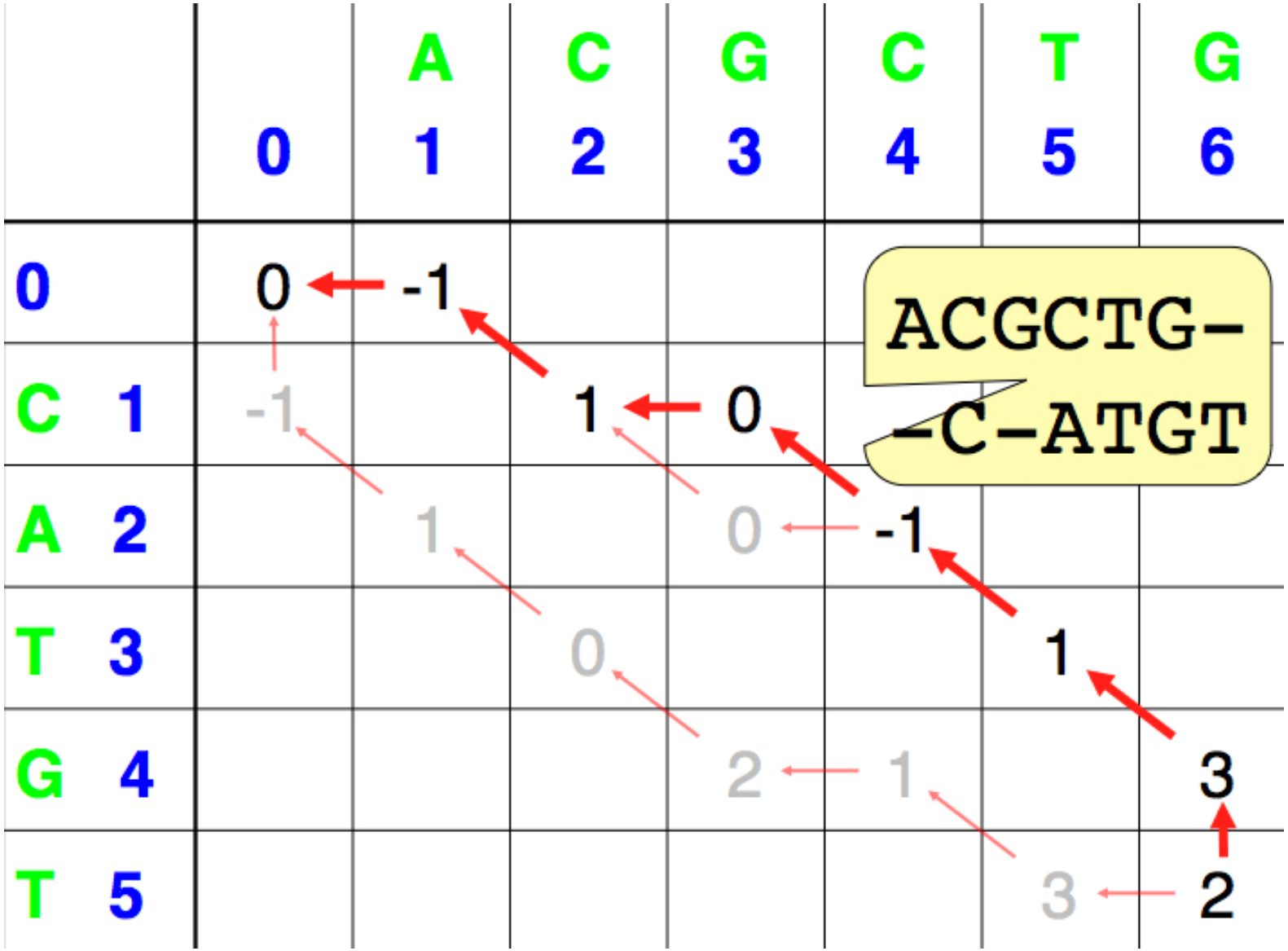
Match= 2 (s=2); Gap= -1 (d=1); Mismatch=-1 (s=1)



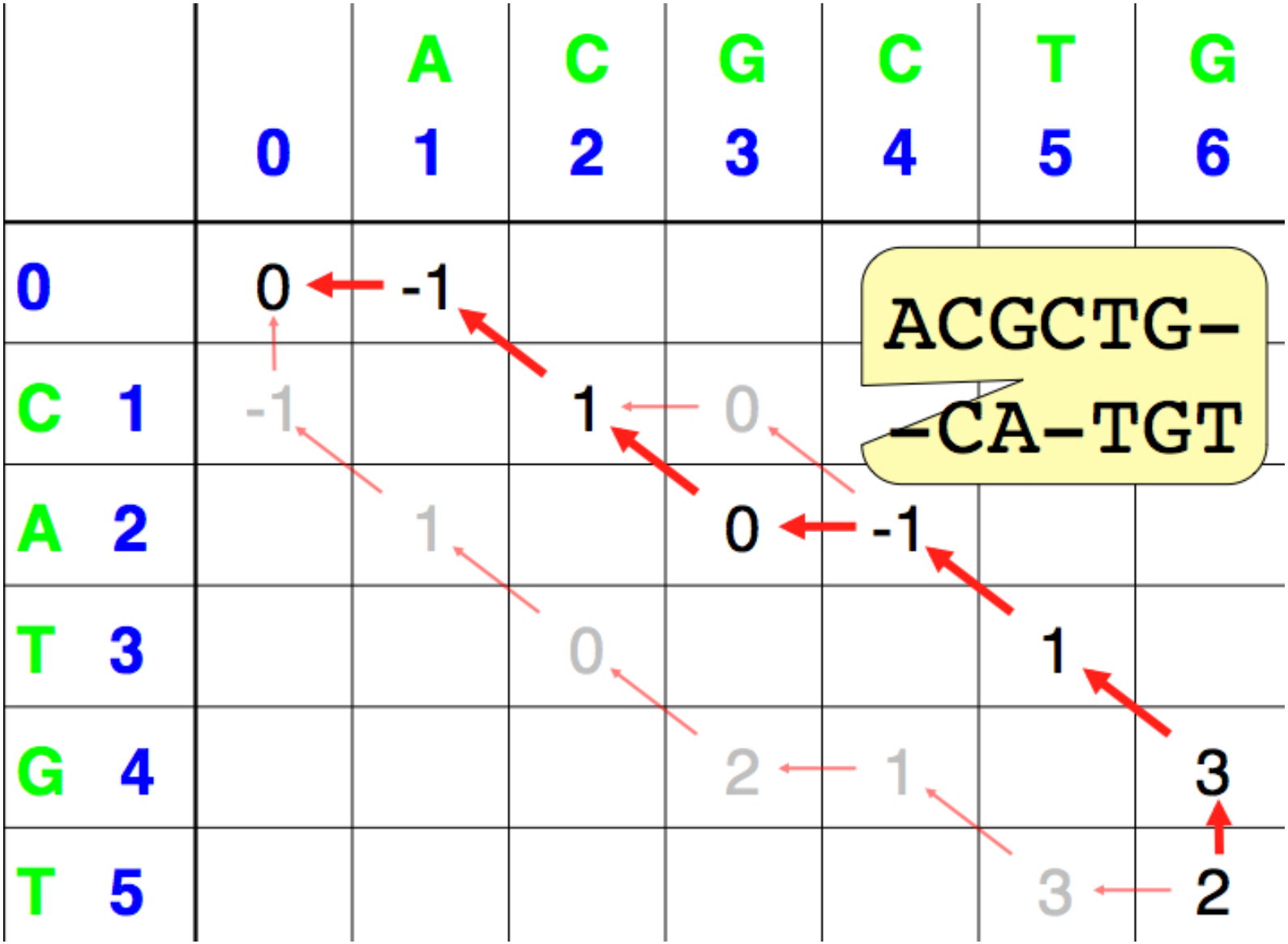
Match= 2 (s=2); Gap= -1 (d=1); Mismatch=-1 (s=1)



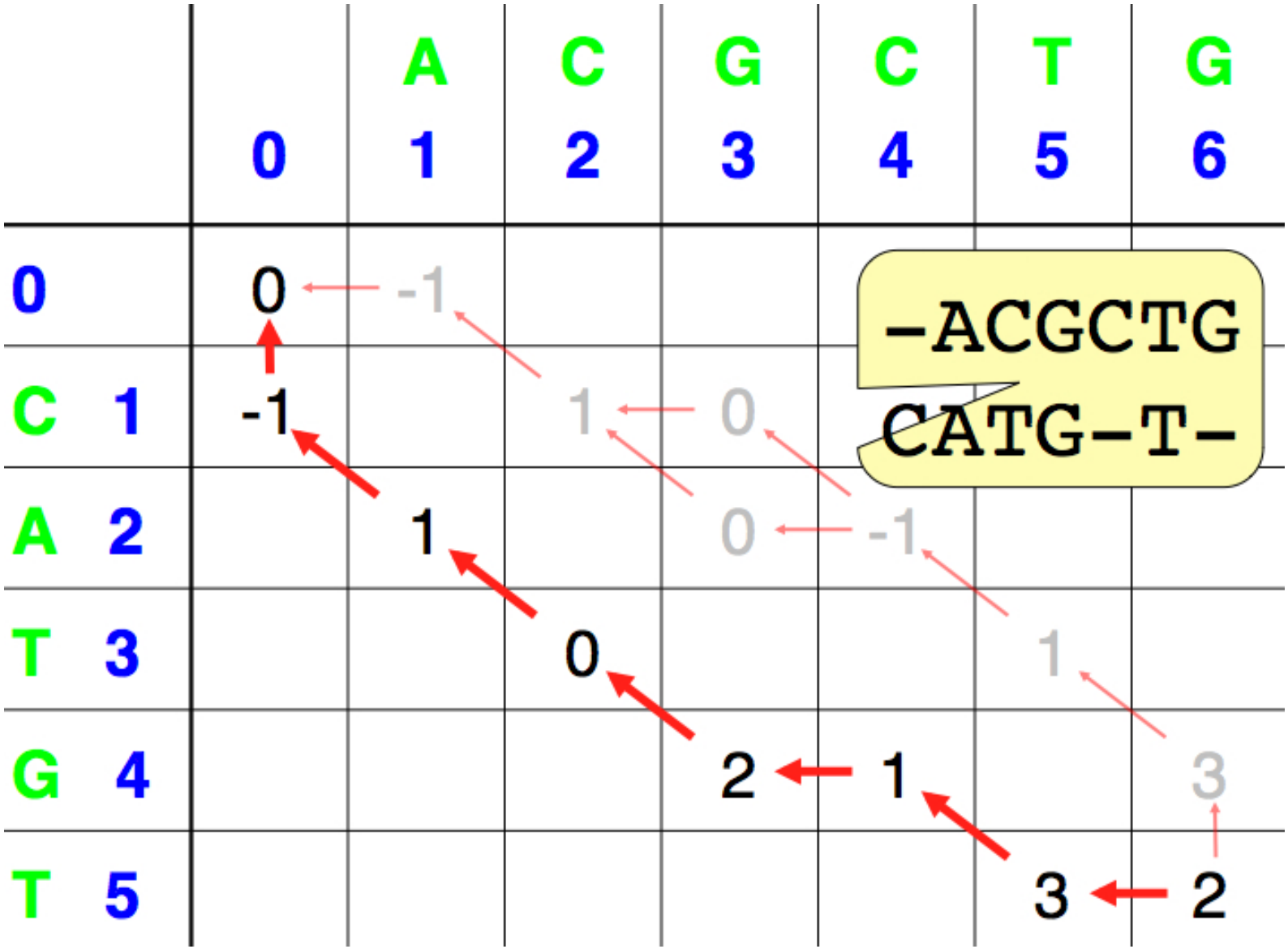
Match= 2; Gap= -1; Mismatch=-1



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



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



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

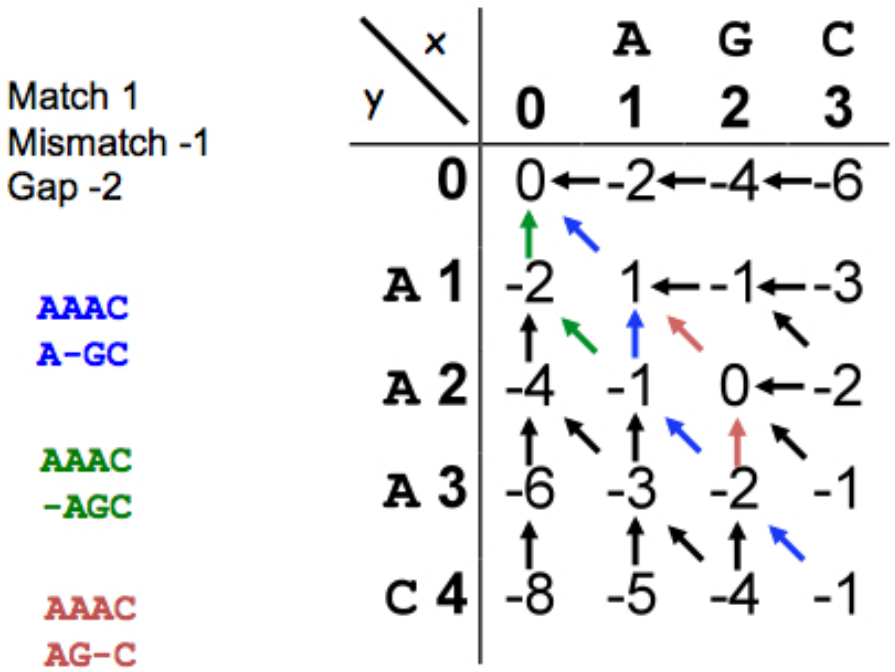


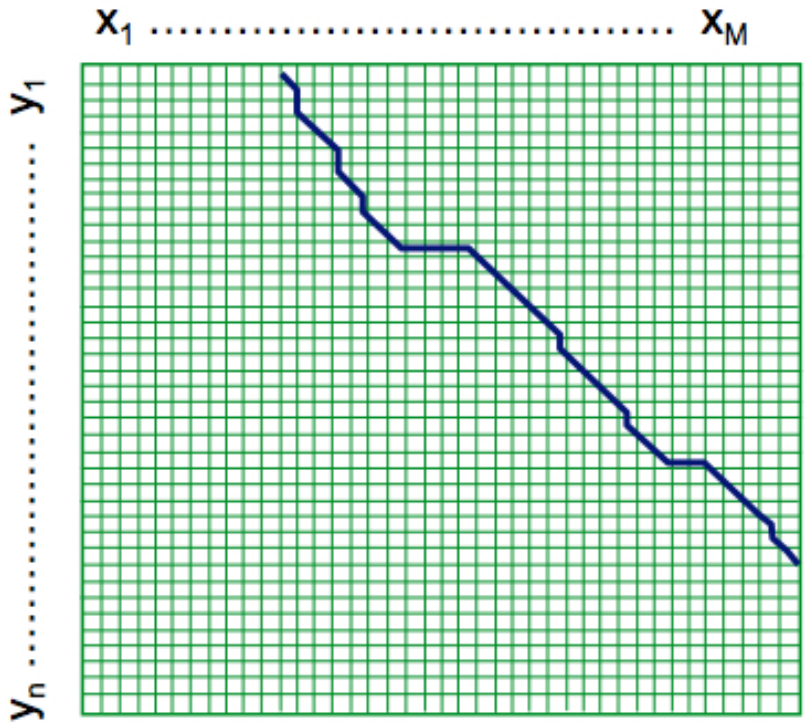
Figure : Given a m x n matrix, the overall complexity of computing all sub-values is $O(nm)$. The final optimal score is the value at position n,m. In this case we align the sequences AGC and AAAC.

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 between two sequences.

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

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



Changes:

1. **Initialization**

   For all i, j,

   $F(i, 0) = 0$

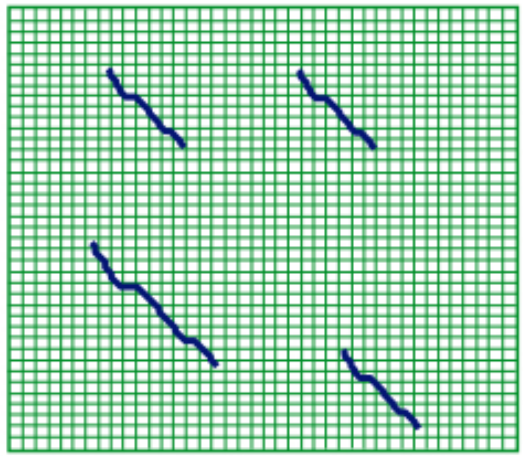
   $F(0, j) = 0$

2. **Termination**

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

The local alignment: the Smith-Waterman algorithm

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



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

y = **cccggg**aaccaacc

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

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

Termination:

1. If we want the best local alignment…

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

2. If we want all local alignments scoring > t

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

Example, Local alignment TAATA vs TACTAA

y = TAATA
x = TACTAA

y \ x		T	A	C	T	A	A
	0	1	2	3	4	5	6
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

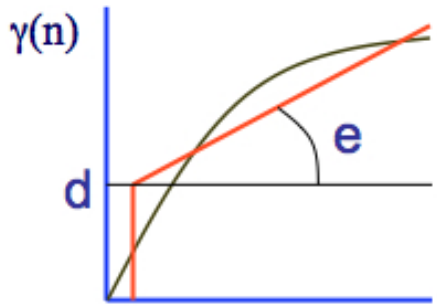
y \ x		T	A	C	T	A	A
	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
T 1	0	1	0	0	1	0	0
A 2	0	0	2	0	0	2	1
A 3	0	0	1	1	0	1	3
T 4	0	0	0	0	2	0	1
A 5	0	0	1	0	0	3	1

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.

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

gap open	gap 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 $x_1 \ldots x_i$ to $y_1 \ldots y_j$
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 $x_1 \ldots x_i$ to $y_1 \ldots y_j$

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:

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

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

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

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

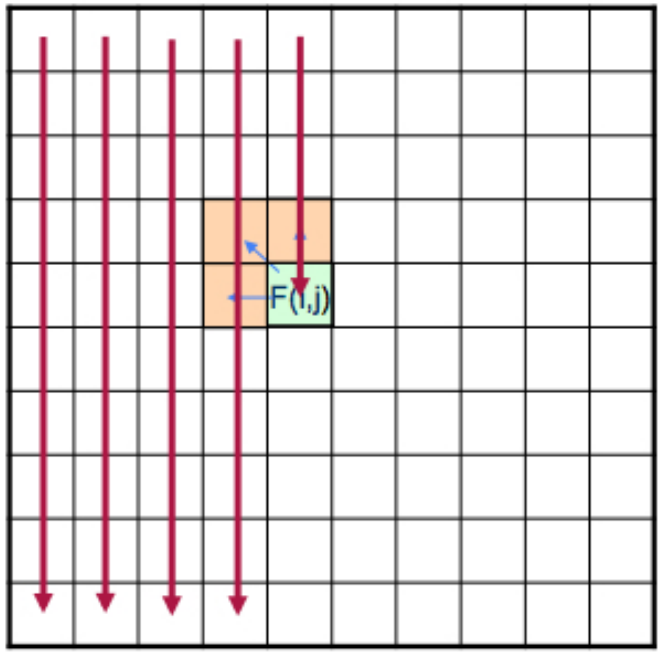
Termination: similar

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., 0(nm). The space complexity is roughly the number of vertices in the edit graph, i.e., 0(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

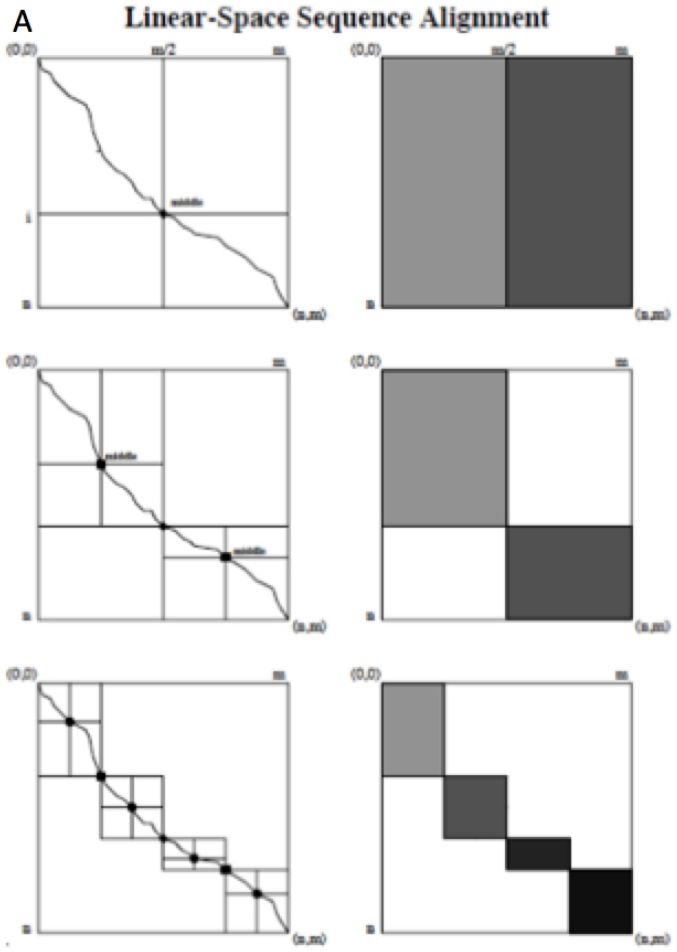
It is easy to compute F(M, N) in linear space




```
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) = …
```

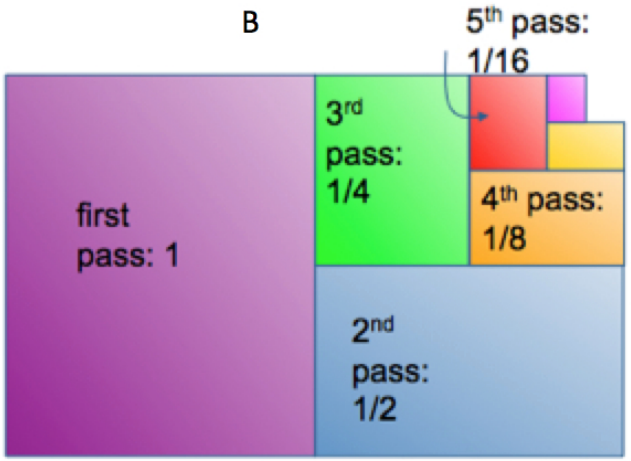
Figure : Space complexity of computing just the score itself is O(n); we only need the previous column to calculate the current column, and we can then throw away that previous column once we have done using it

Space-Efficient Sequence Alignment, Hirschberg algorithm



A: space complexity

B: time complexity



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 $s_{*,j-i}$ (column $j-1$). Therefore, the alignment scores in the columns before $j-1$ can be discarded while computing alignment scores for columns j, $j+1$,.... 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.

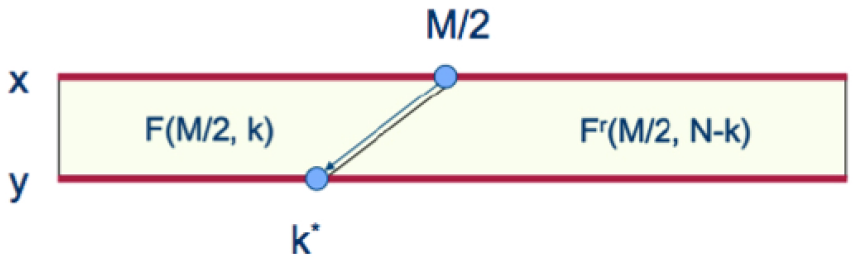
Space-Efficient Sequence Alignment, Hirschberg algorithm

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 +.. $<$2 * area and therefore compute the longest path in time 0(nm) and space O(n).

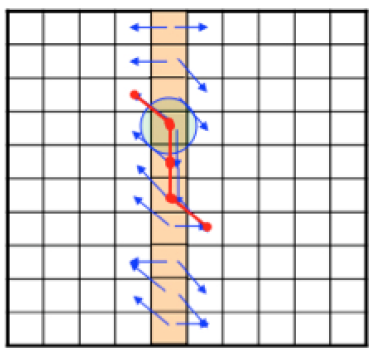
Space-Efficient Sequence Alignment, Hirschberg algorithm

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)

Hirschberg algorithm: details

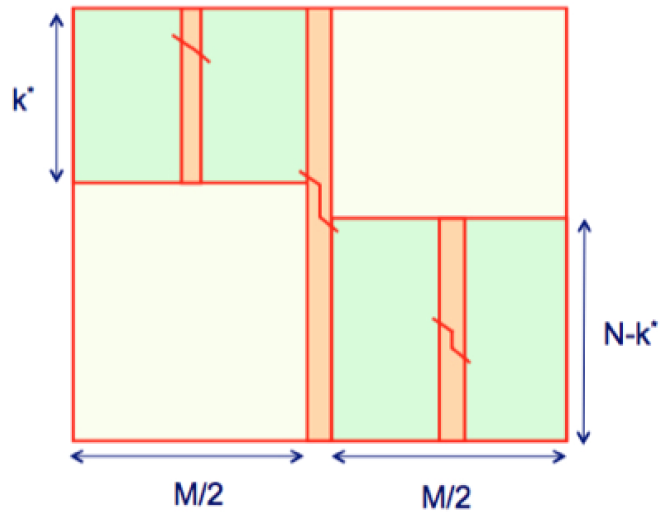


Now, we can find k^* maximizing $F(M/2, k) + F^r(M/2, k)$
Also, we can trace the path exiting column M/2 from k^*



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

- Iterate this procedure to the left and right!



Nussinov Algorithm: The Biological problem

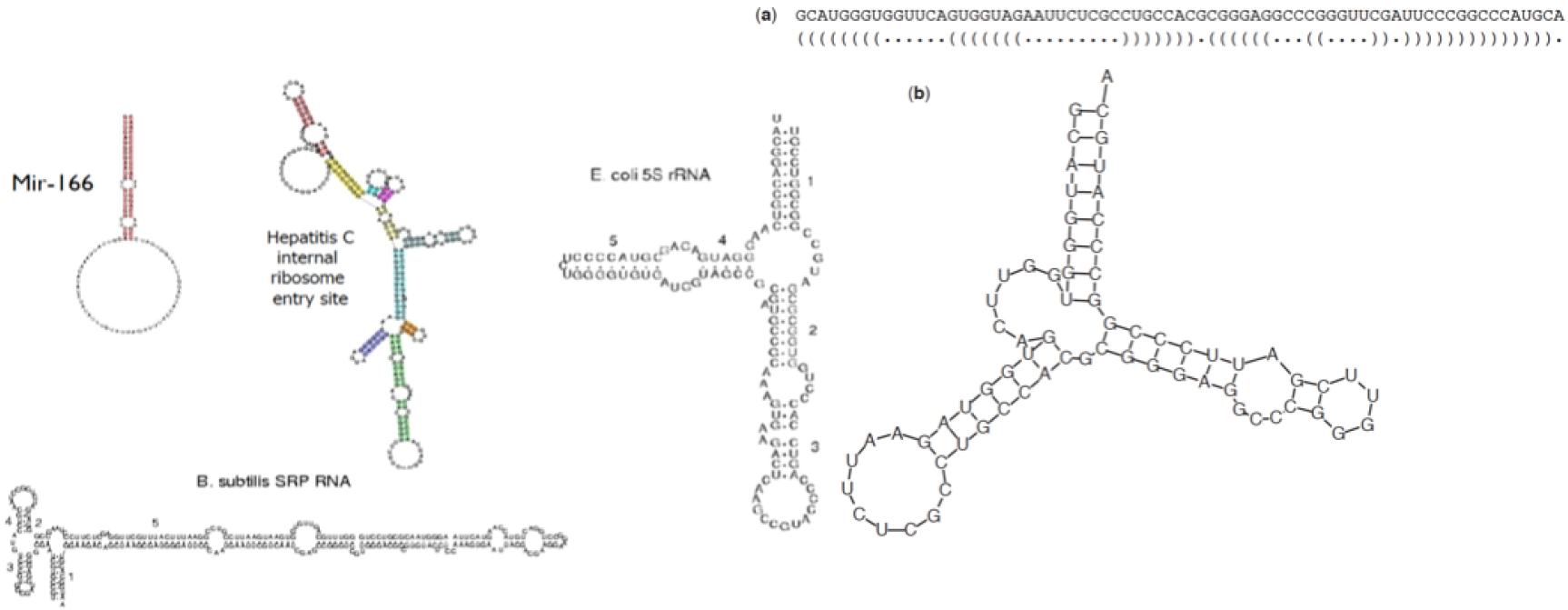


Figure : Examples of RNA molecules in nature; many molecules of RNA do not translate into proteins; using the pairing rule A-T, C-G, the molecule could find regions of perfect pairing so to have intrachain interactions. Therefore, the molecule folds into 2 Dimensional shape (termed secondary structure)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

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

The intrachain folding of RNA reveals the RNA Secondary Structure

This tells us which bases are paired in the subsequence from x_i to x_j Every optimal structure can be built by extending optimal substructures.

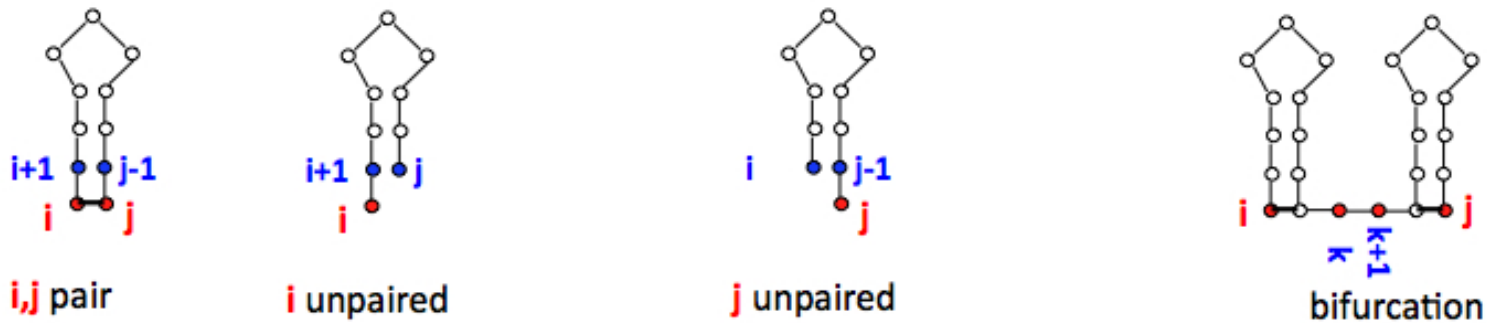


Figure : Set of paired positions on interval $[i, j]$. Suppose we know all optimal substructures of length less than $j - i + 1$. The optimal substructure for $[i, j]$ must be formed in one of four ways: i,j paired; i unpaired; j unpaired; combining two substructures. Note that each of these consists 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 . . . j].
2. for $1 \leq i \leq n$ and $i < j \leq n$:
   for $i = 1, ..., n : \gamma(i, i) = 0$;
   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

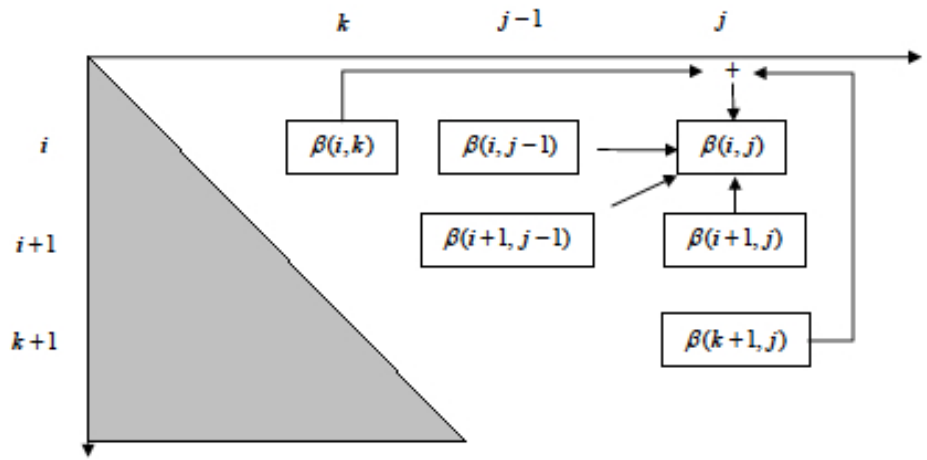


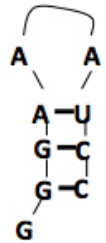
Figure : Details of matrix filling

Nussinov algorithm: example

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

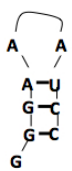
Example:

GGGAAAUCC



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

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



	G	G	G	A	A	A	U	C	C
G	0	0	0	0					
G	0	0	0	0	0				
G		0	0	0	0	0			
A			0	0	0	0	?		
A				0	0	0	1		
A					0	0	1	1	
U						0	0	0	0
C							0	0	0
C								0	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.

Challenges in RNA folding

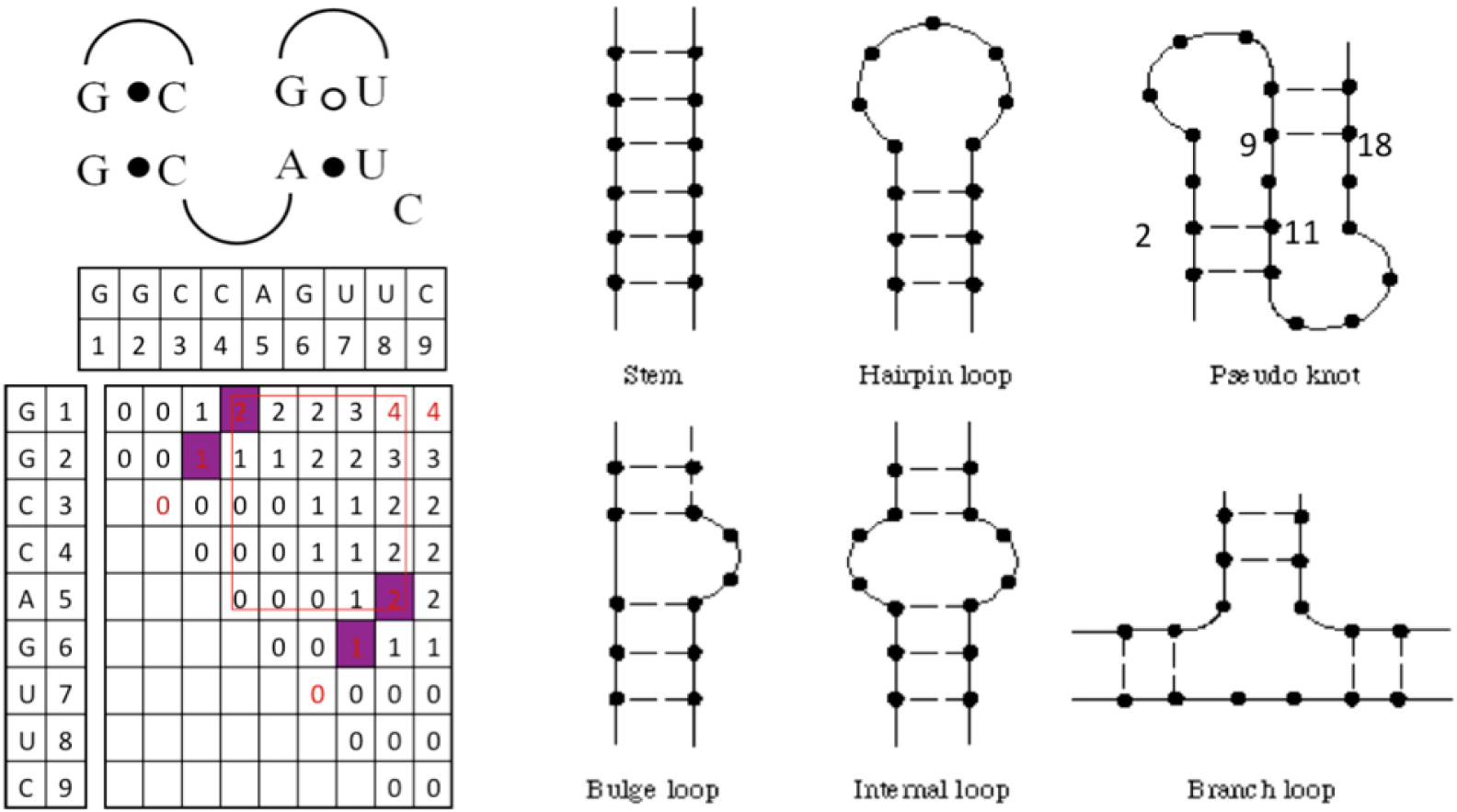


Figure : left: the case of bifurcation; right: from easy to difficult cases

Challenges in alignment: repeats and inversions

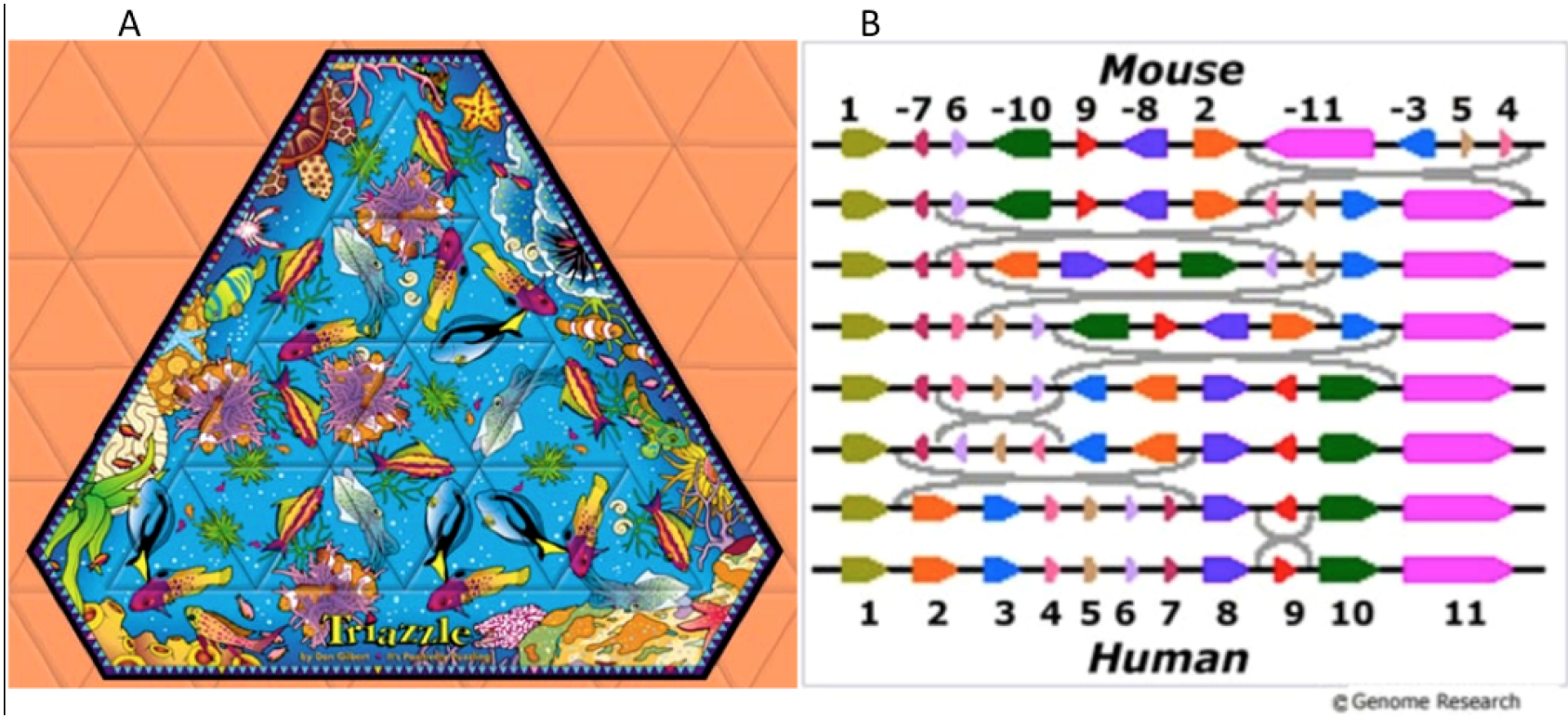
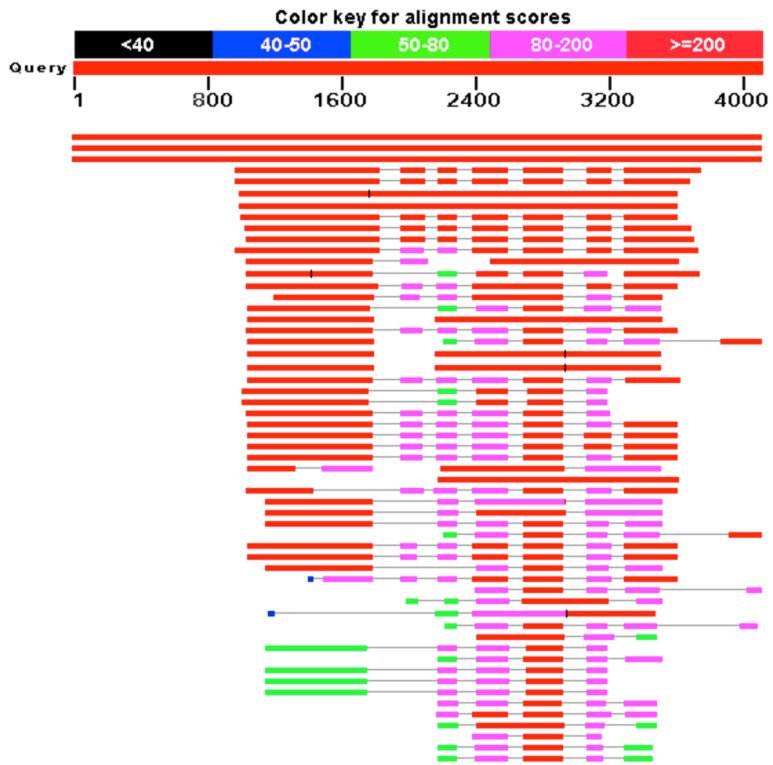


Figure : difficulty when there are repeats of different length and inverted blocks (also nested)

3 Approximate Search algorithms

The problem is to find in a database all sequences with interesting similarities. Below there is an example of output for the following task: a query (an unknown gene sequence) is compared with other sequences with known functions in a database. Perfect hits are red colored. Regions that were weaker in match are pink, green, or blue; alignment details are also available. Algorithms considered: Blast, Patternhunter.



3 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 $9x10^9$ length sequences would take about $9x10^{18}$ 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.

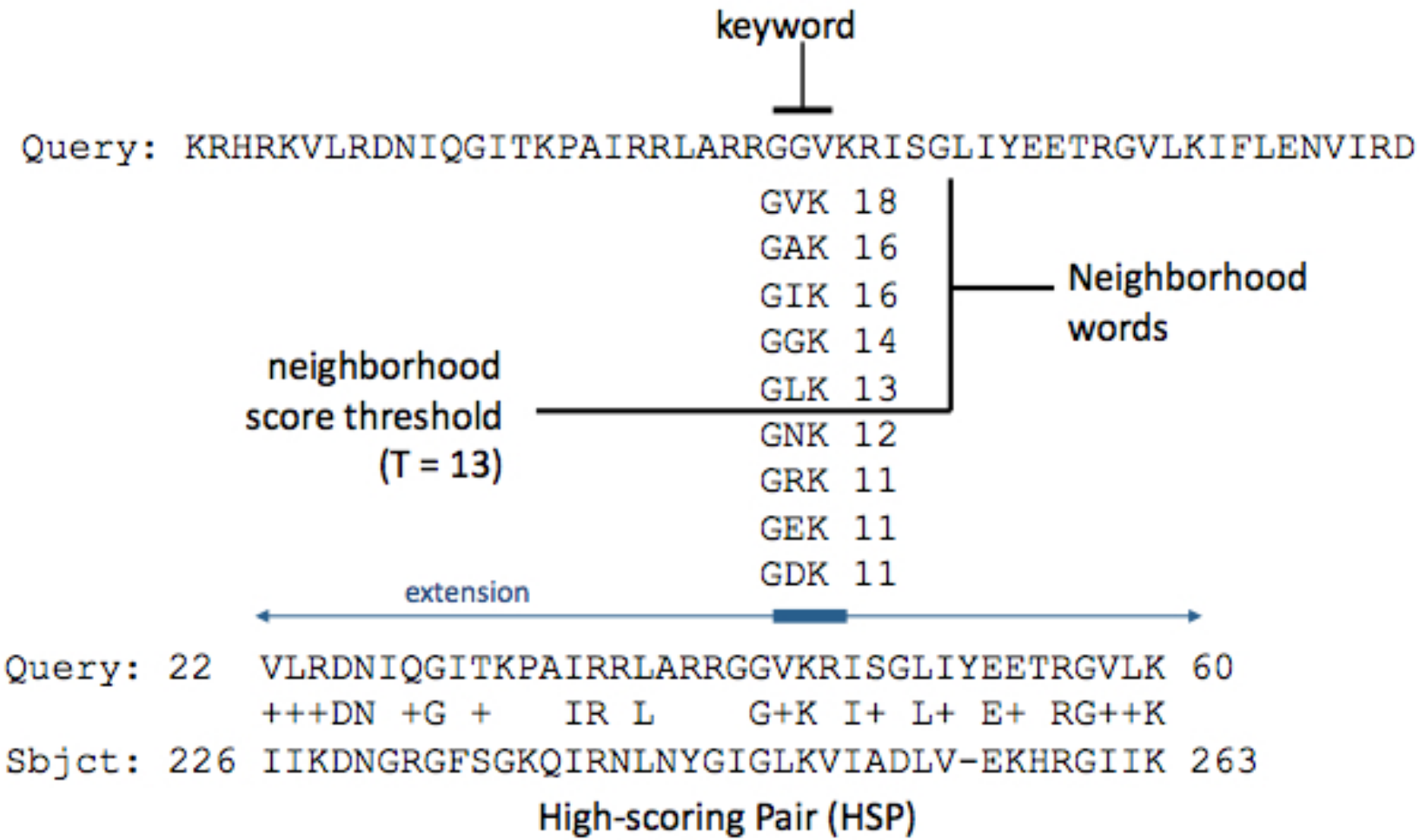
BLAST (Basic Local Alignment Search Tools)

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 seed 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 (http://blast.ncbi.nlm.nih.gov/Blast.cgi)

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 alignments (in the sequences) with score 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.
- BLAST outputs a list of HSPs together with E-values that measure how frequent such HSPs would occur by chance. This is calculated with respect of a database with similar size and random data.

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 (amino acids) 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.

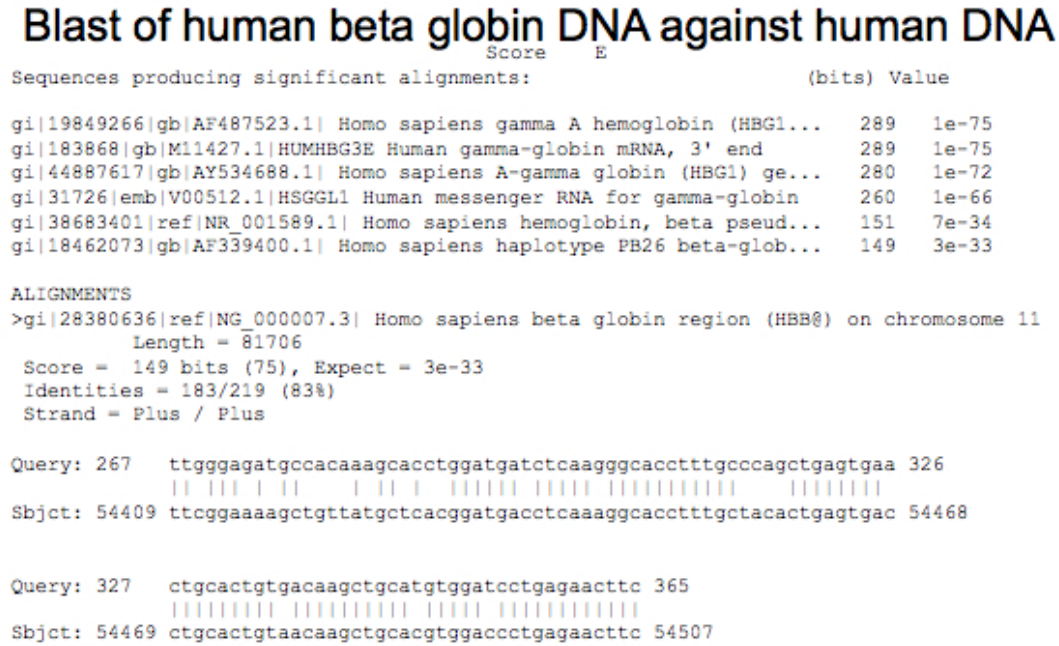
Note that the use of seeds of length W and the termination of extensions with fading scores are both steps that speed up the algorithm, but also imply that BLAST is not guaranteed to find all HSPs.

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 λ. 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 λ 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 λ, 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: $Z' = \frac{\lambda Z - lnW}{ln2}$. 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 $\geq Z$ is given by $P(k) = \frac{E^k}{k!} e^{-E}$ (see also www.ncbi.nlm.nih.gov/blast/tutorial/Altschul-1.html)
- 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
Sequences producing significant alignments:                       (bits) Value

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   ttgggagatgccacaaagcacctggatgatctcaagggcacctttgcccagctgagtgaa 326
              || ||| | ||    | || |  |||||| ||||| ||||||||||    ||||||||
Sbjct: 54409 ttcggaaaagctgttatgctcacggatgacctcaaaggcacctttgctacactgagtgac 54468


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

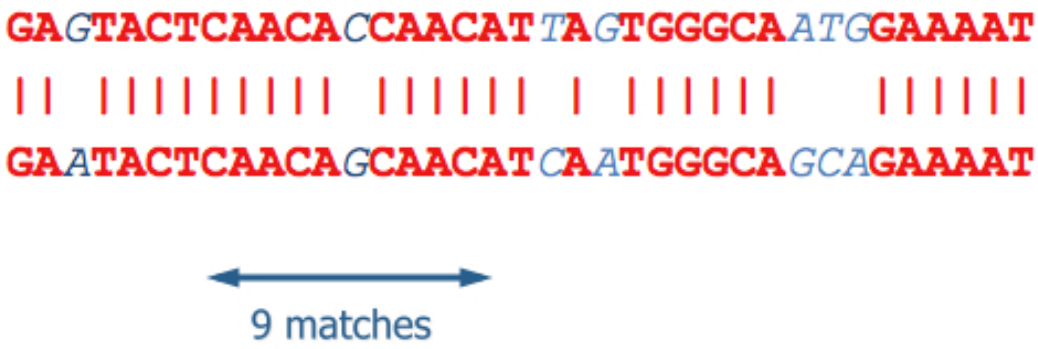
from Altschul: The expected-time computational complexity of BLAST is approximately $aW + bN + cNW/20^w$, where W is the number of words generated, N is the number of residues in the database and a, b and c are constants. The W term accounts for compiling the word list, the N term covers the database scan, and the NW term is for extending the hits. Although the number of words generated, W, increases exponentially with decreasing the threshold, it increases only linearly with the length of the query, so that doubling the query

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

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



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

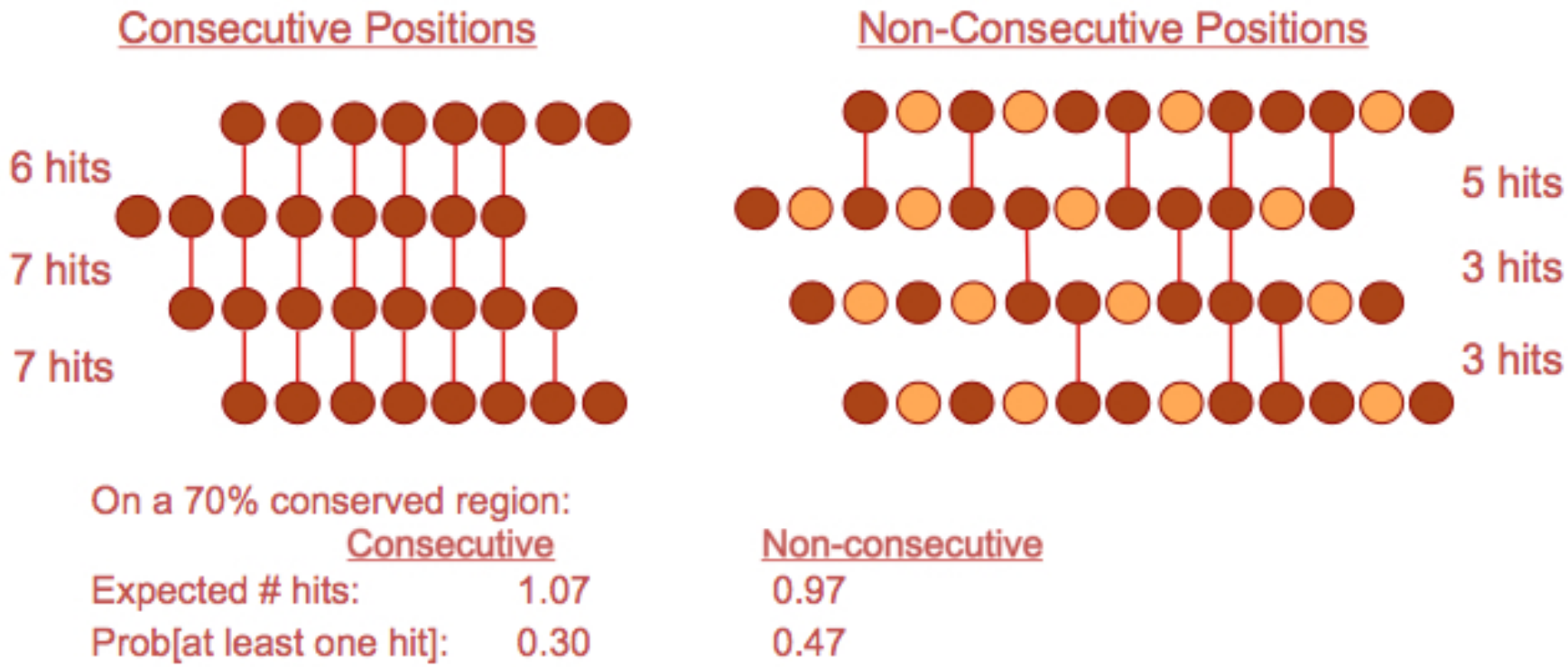
Patternhunter

The big 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 contains 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, you have to use a longer seed. However, we now face a dilemma: increasing seed size speeds up, but looses sensitivity; decreasing seed size gains sensitivity, but looses 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 dont care position. This simple change makes a huge difference: significantly increases hit number 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



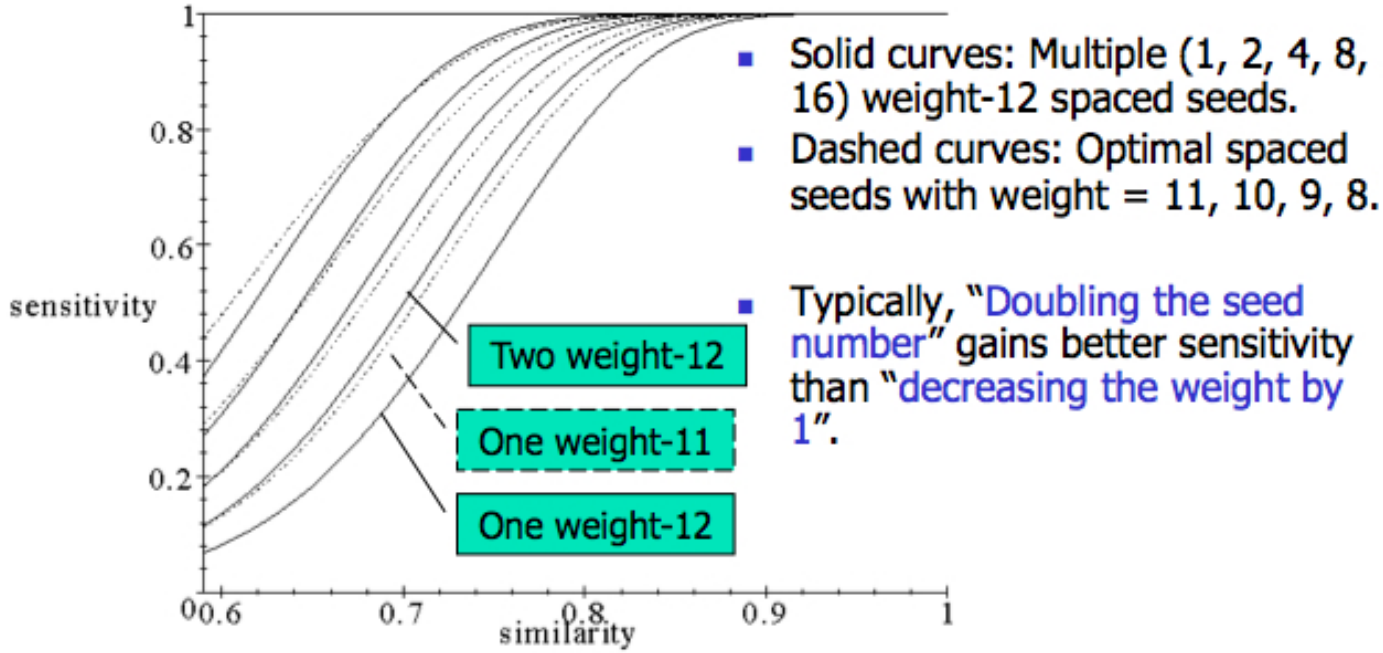
- 111010010100110111 (called a model)
  - Eleven required matches (weight=11)
  - Seven "don't care" positions

```
GAGTACTCAACACCAACATTAGTGGCAATGGAAAAT...
||  |||||||||  |||||  ||  |||||    ||||||
GAATACTCAACAGCAACACTAATGGCAGCAGAAAAT...
        1110100101001101111
```

- Hit = all the required matches are satisfied.
- BLAST seed model = 11111111111

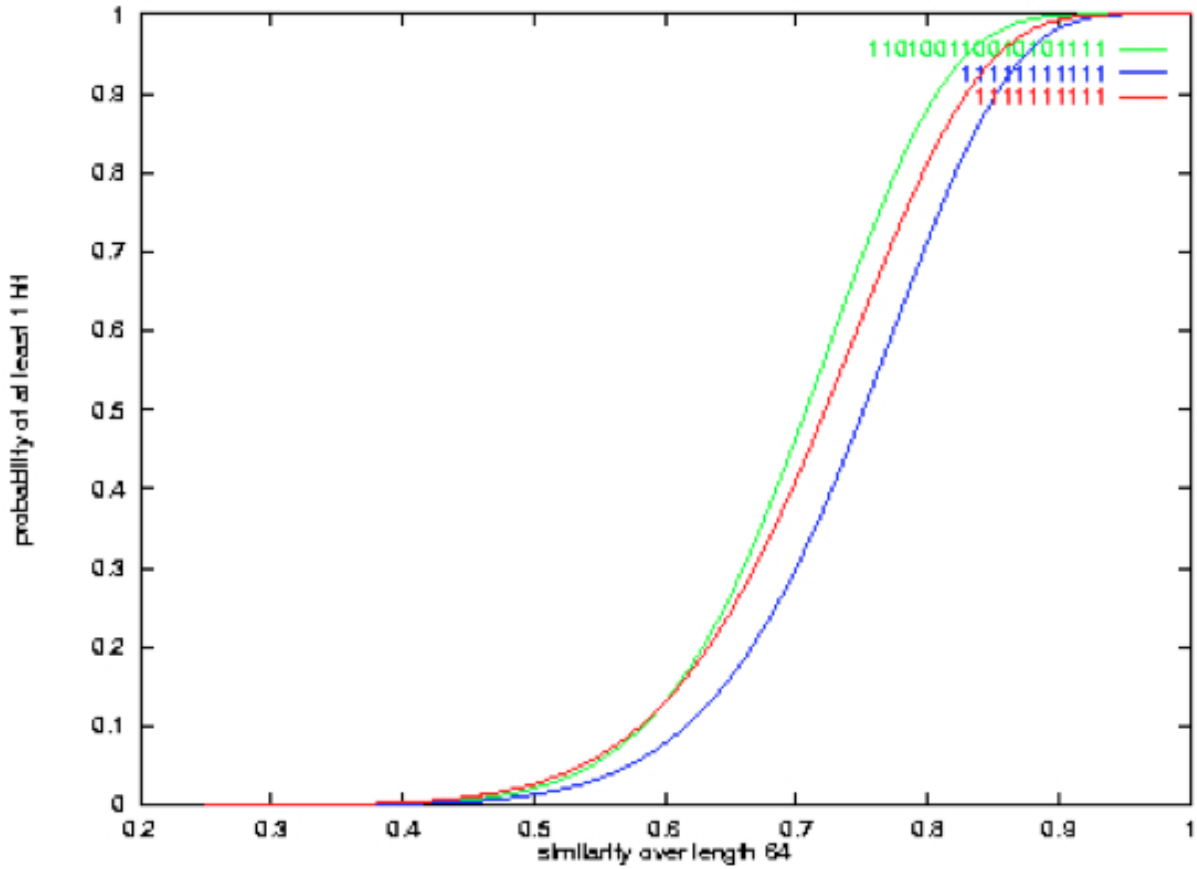
```
111010010100110111
 111010010100110111
  111010010100110111
   111010010100110111
    111010010100110111
     111010010100110111
      111010010100110111
        ......
```

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



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

Sensitivity: PH weight 11 seed vs BLAST 11 & 10



Comparing different seeds number

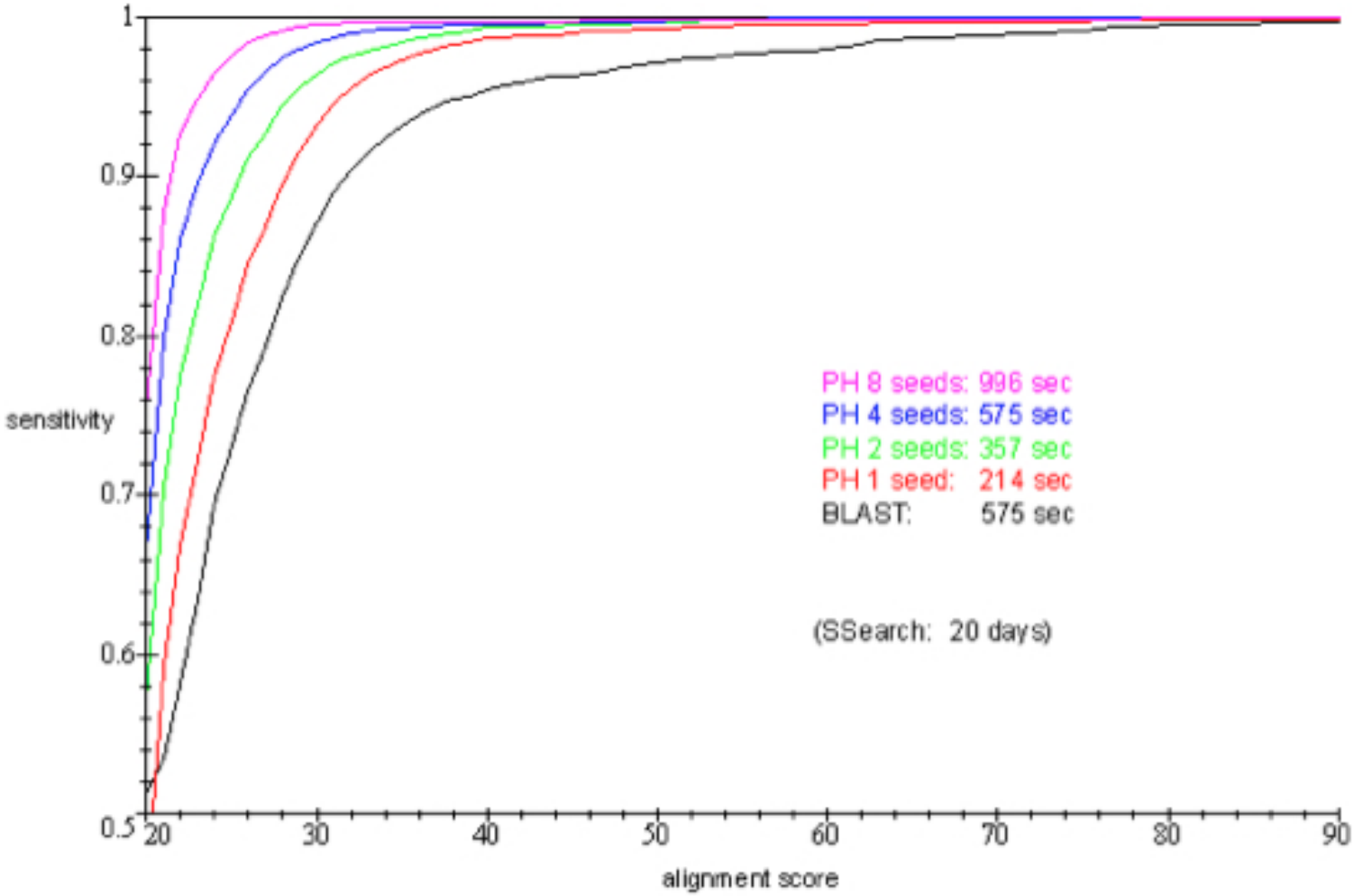
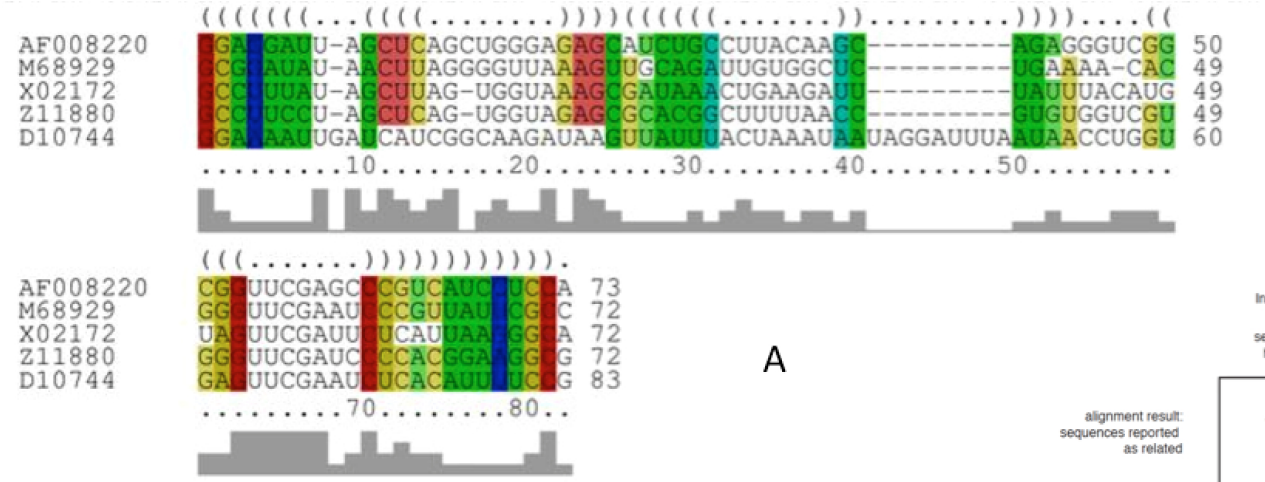
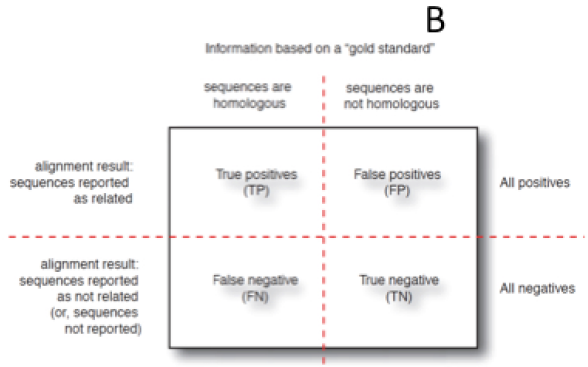


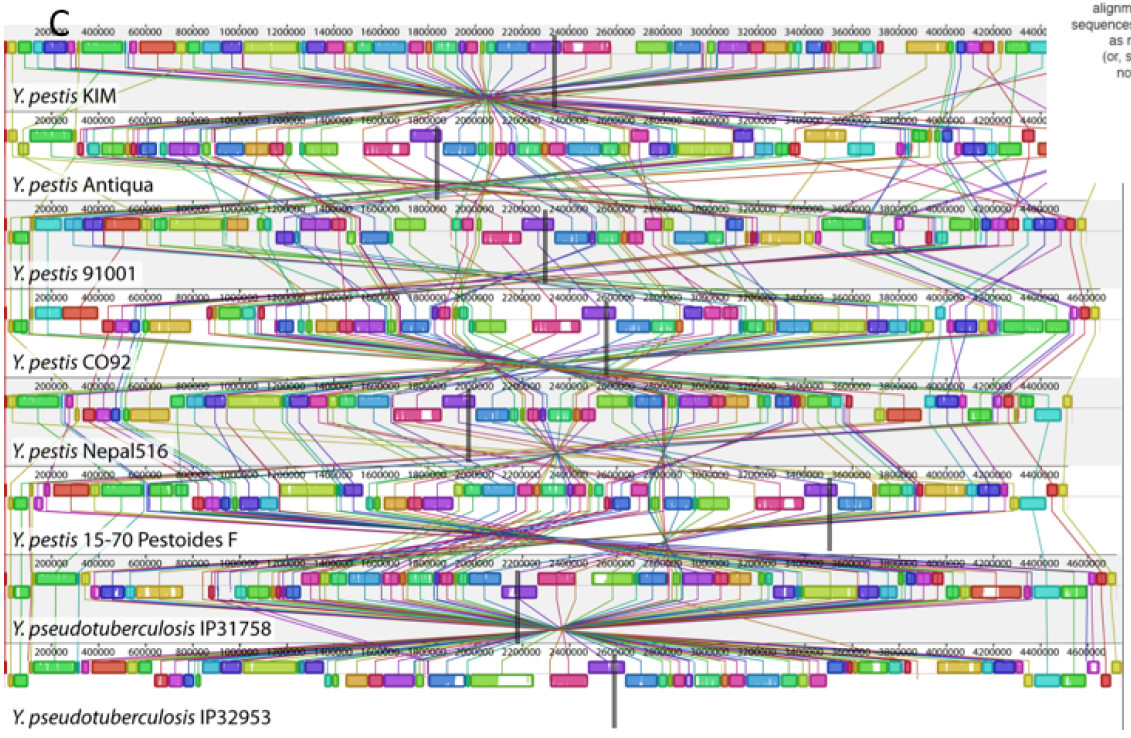
Figure : sensitivity versus alignment score

4 Multi sequence alignment: examples of results



A





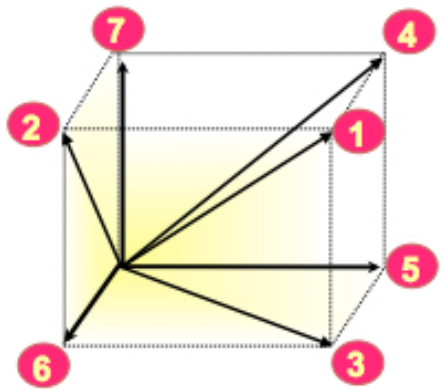
A) Multi alignment of RNA Sequences (note conservation score at the bottom and secondary structure prediction on top)

B) Positive and Negative assessment of alignment.

C) Multi alignment of genomic sequences

Challenges of extending dynamic programming to n sequences

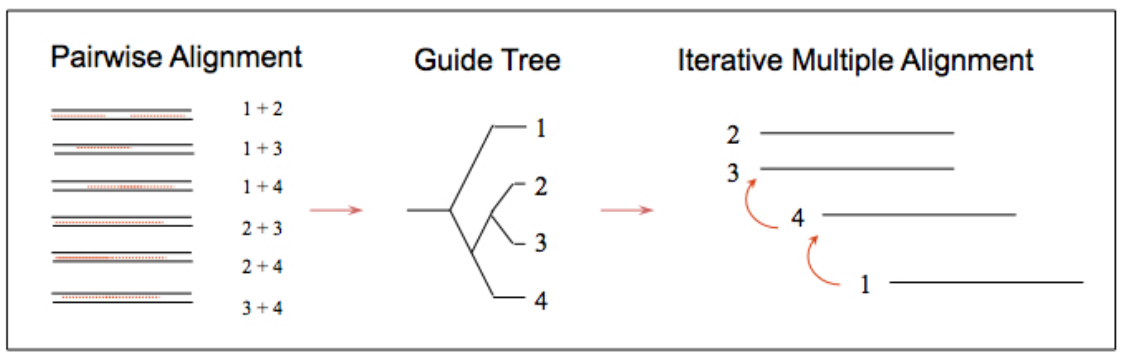
- For two sequences, there are three ways to extend an alignment
- for n sequences, a n-dimensional dynamic programming hypercube has to be computed and for each entry we have to evaluate $(2^n - 1)$ predecessors.
- Given 3 sequences, the figure below shows a three-dimensional alignment path matrix: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



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

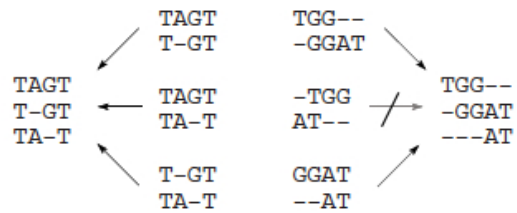
Progressive alignment

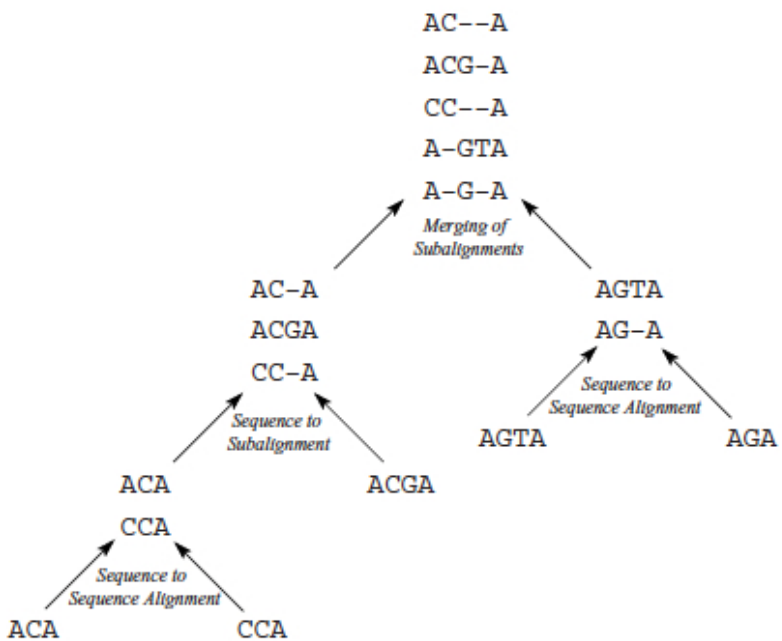
- Progressive alignment methods are heuristic in nature. They produce multiple alignments from a number of pairwise alignments.
- Perhaps the most widely used algorithm of this type is CLUSTALW.
- Given N sequences, align each sequence against each other and obtain a similarity matrix; similarity = exact 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

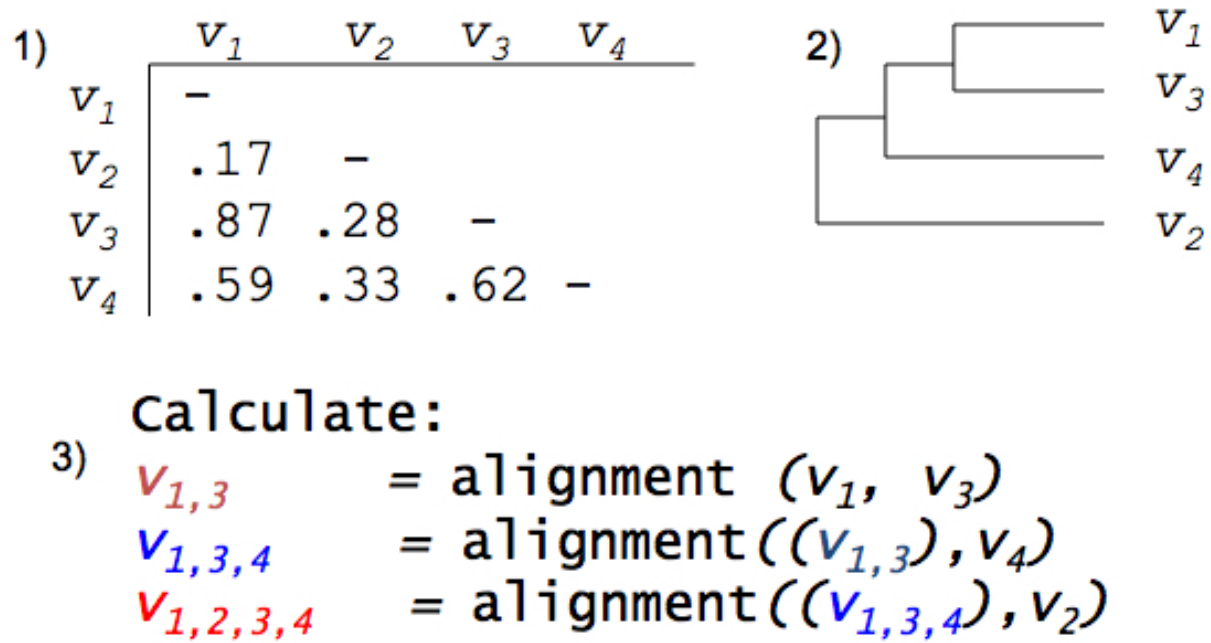


Figure : Progressive alignment of 4 sequences: 1) distance matrix from pairwise alignment; 2) pairwise alignment score analysis; tree showing the best order of progressive alignment, 3) building up the alignment.

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	-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	-2	1	3	-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	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	**5**	-2	-2	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 pA = 1, pT=pG=pC=0 (1st column);
pA = 0.75, pT = 0.25, pG=pC=0 (2nd column);
pA = 0.50, pT = 0.25, pC=0.25 pG=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

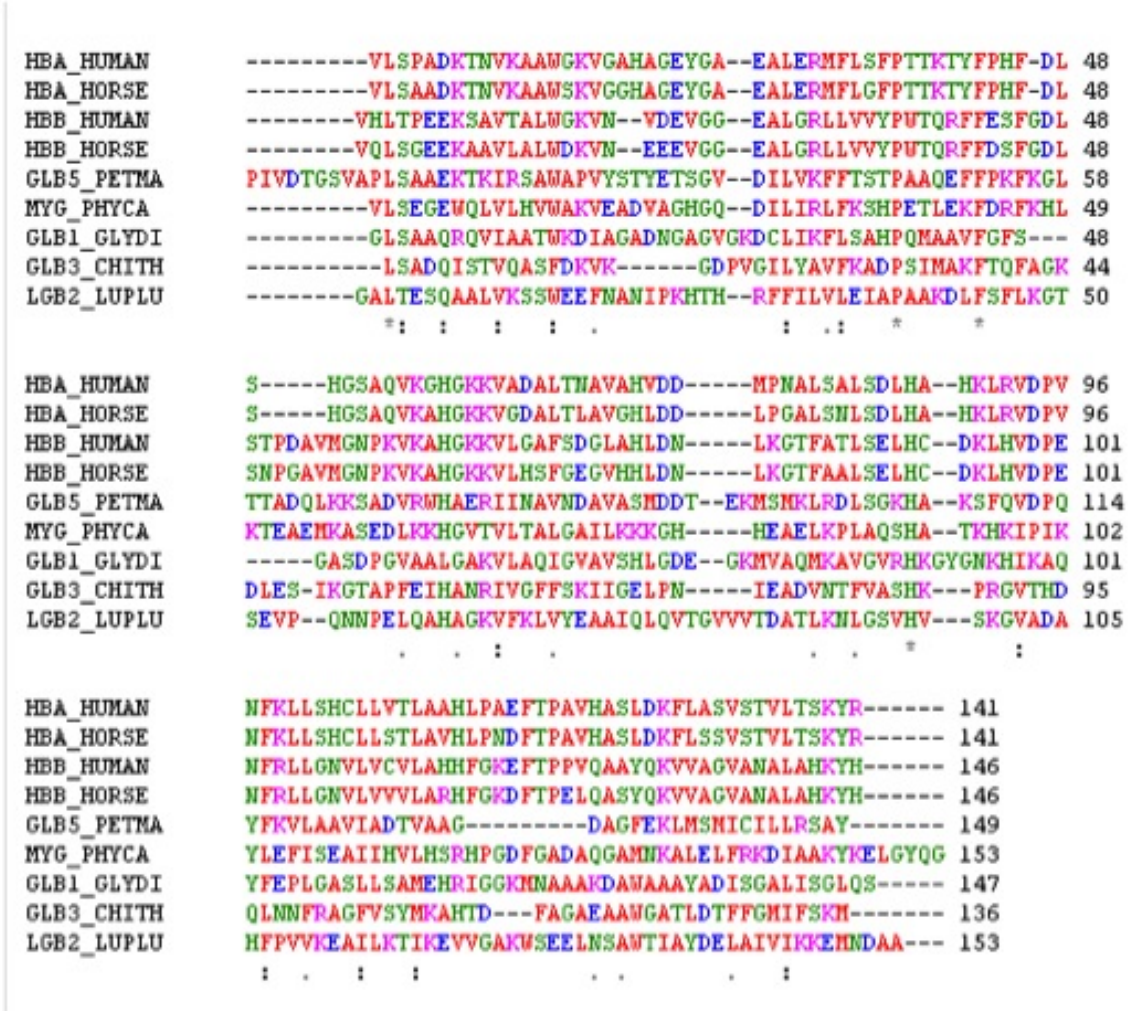


Figure : Chemical properties of amino acids are in color code. The globin proteins from different species could be easily aligned because they have many similar substrings in common.

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

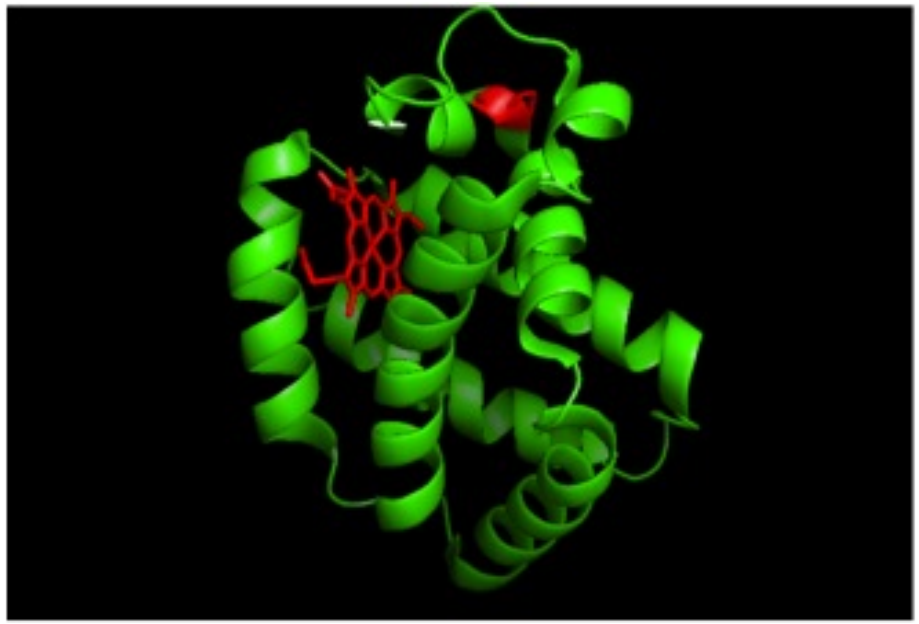


Figure : Human globin 3D structure. The small amount of changes in the globin alignment suggests that globin are likely to have very similar structure (3 D graph). Columns rich of gaps often correspond to unstructured regions (loops); conserved regions often correspond to binding sites or regions where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

5 Genome alignments

Genome scaffolding (i.e. the process of ordering and orientating contigs) of de novo assemblies usually represents the first step in most genome finishing pipelines (figure below on the right). The preferred approach to genome scaffolding is currently based on assembling the sequenced reads into contigs and then using paired end information to join them into scaffolds. The figure below show the overlapping reads used to cover assemble the genome and the problem with repeats. The algorithm presented here is the Burrows-Wheeler transform.

Burrows- Wheeler transform: saving memory in NGS alignments

The current sequencing procedures are characterized by highly parallel operations, much lower cost per base, but 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 (1994) is commonly used. The Burrows and Wheeler transform (BWT) is a block sorting lossless and reversible data transform. The BWT can permute a text into a new sequence which is usually more compressible The transformed text can be better compressed with fast locally-adaptive algorithms, such as run-length-encoding (or move-to-front coding) in combination with Huffman coding (or arithmetic coding). Burrows obtained the Ph.D at the Computer Laboratory.

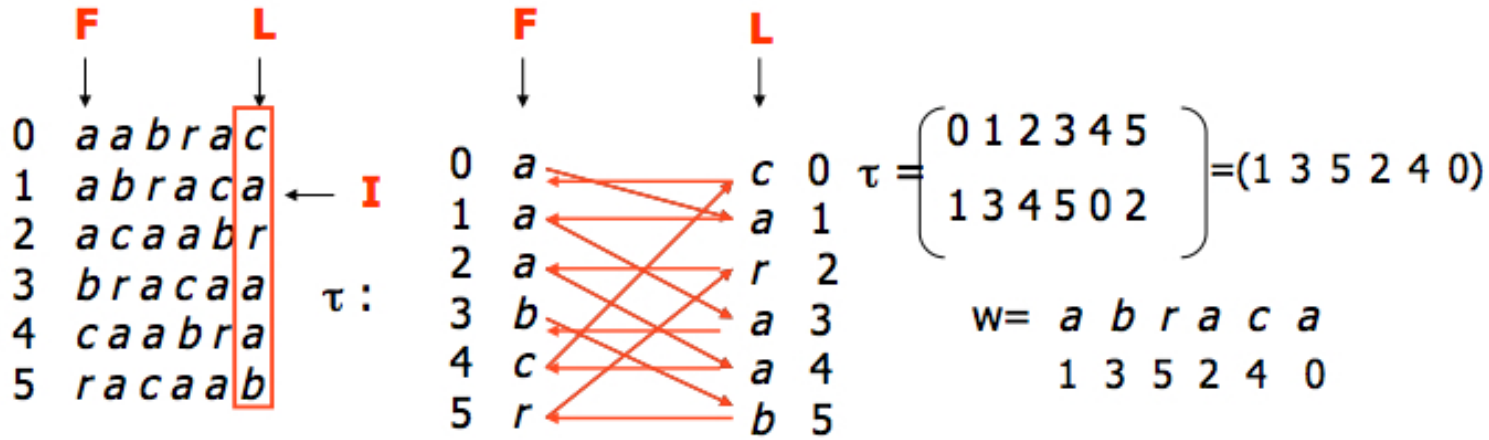
Burrows-Wheeler Transform

INPUT (example): T = “abraca”; then we sort lexicographically all the cyclic shifts of T

For all i $\neq$ I, 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.

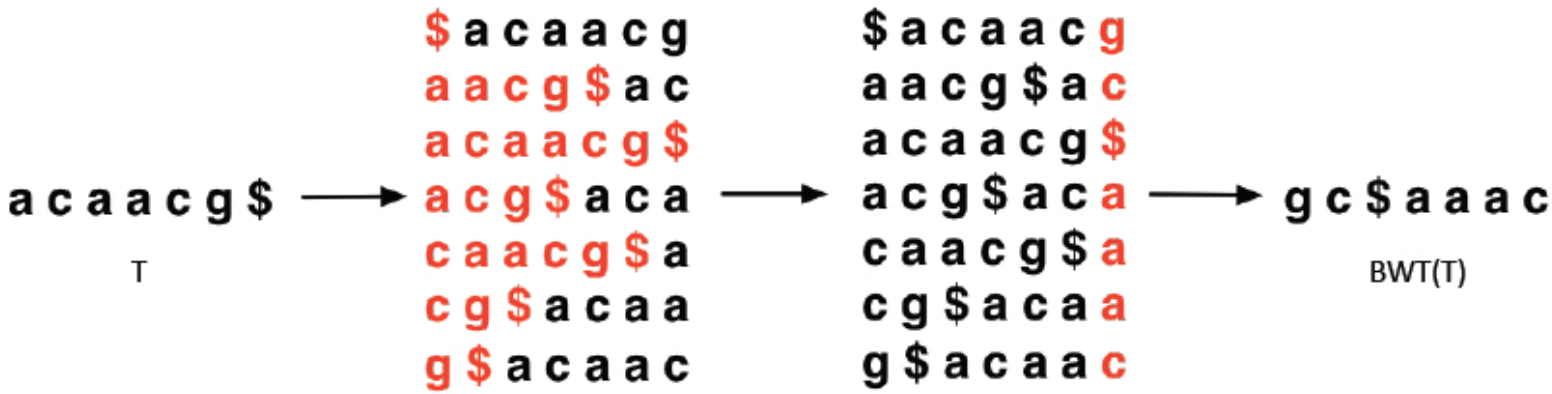
OUTPUT: BWT(T)=caraab and the index I, that denotes the position of the original word T after the lexicographical sorting.

The Burrows-Wheeler Transform is reversible, in the sense that, given BWT(T) and an index I, it is possible to recover the original word T.



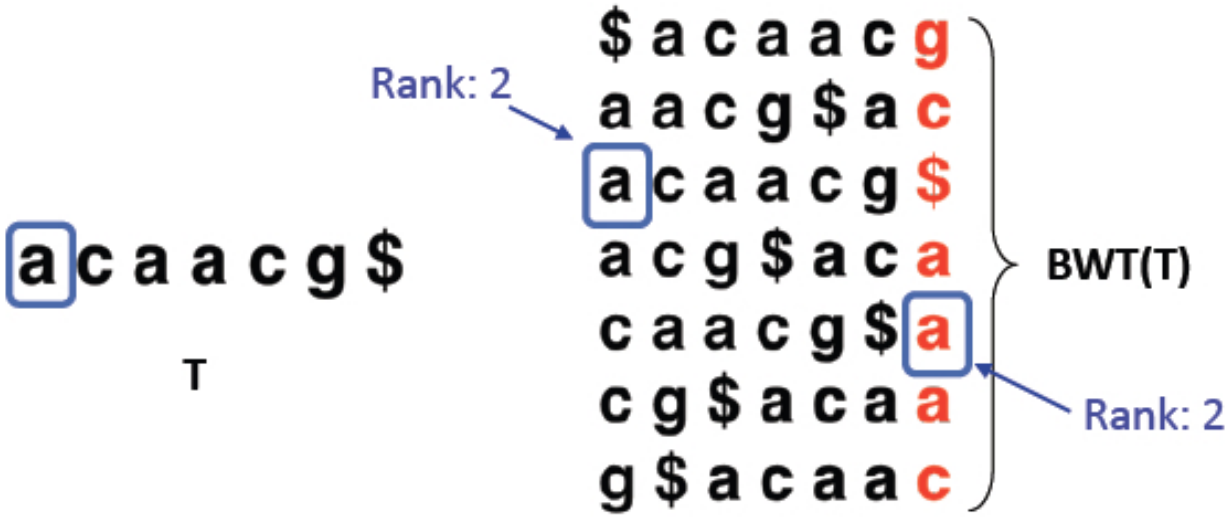
Burrows-Wheeler Transform example

Once BWT(T) is built, all else shown here (i.e. the matrix) is discarded. Three steps: 1) Form a N*N matrix by cyclically rotating (left) the given text to form the rows of the matrix. Here we use '$' as a sentinel (lexicographically greatest character in the alphabet and occurs exactly once in the text but it is not a must). 2) Sort the matrix according to the alphabetic order. Note that the cycle and the sort procedures of the Burrows-Wheeler induces a partial clustering of similar characters providing the means for compression. 3) The last column of the matrix is BWT(T) (we need also the row number where the original string ends up).



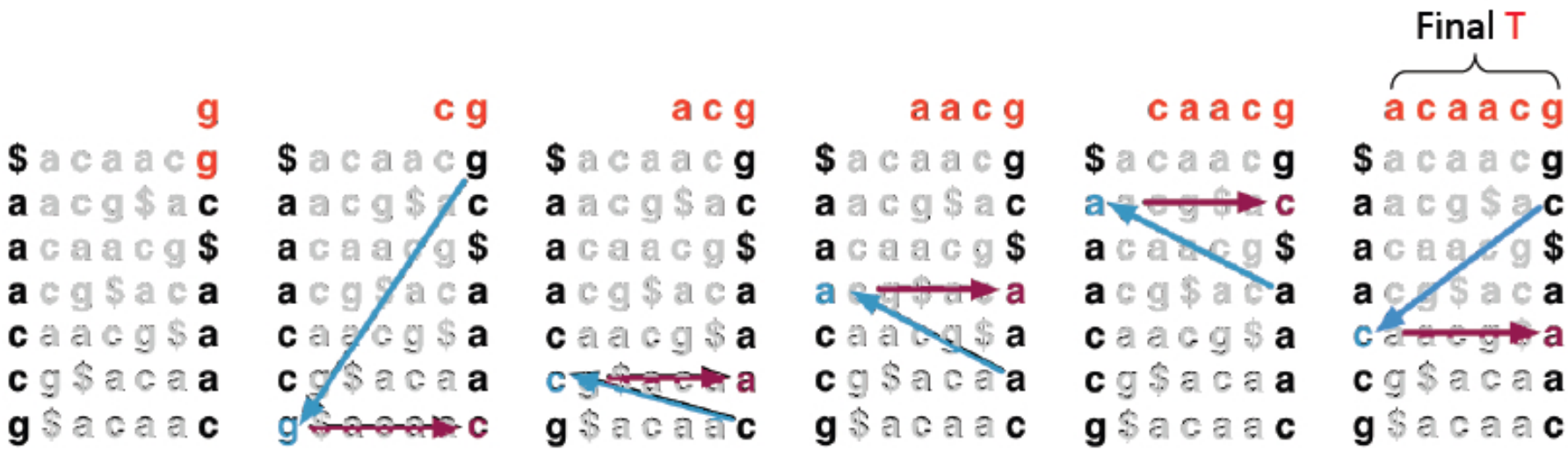
Burrows-Wheeler Transform in alignment: example

Property that makes BWT(T) reversible is LF Mapping: the ith occurrence of a character in **L**ast column is same text occurrence as the ith occurrence in the **F**irst column (i.e. the sorting strategy preserves the relative order in both last column and first column).



Burrows-Wheeler Transform in alignment: example

To recreate T from BWT(T), repeatedly apply rule: T = BWT[LF(i)] + T; i = LF(i) where LF(i) maps row i to row whose first character corresponds to i"s last per LF Mapping. First step: $S = 2$; $T = \$$. Second step: s = LF[2] =6; $T = g\$$. Third step: s = LF[6] = 5; $T = cg\$$.



The BWT(T) is more amenable to subsequent compression algorithms

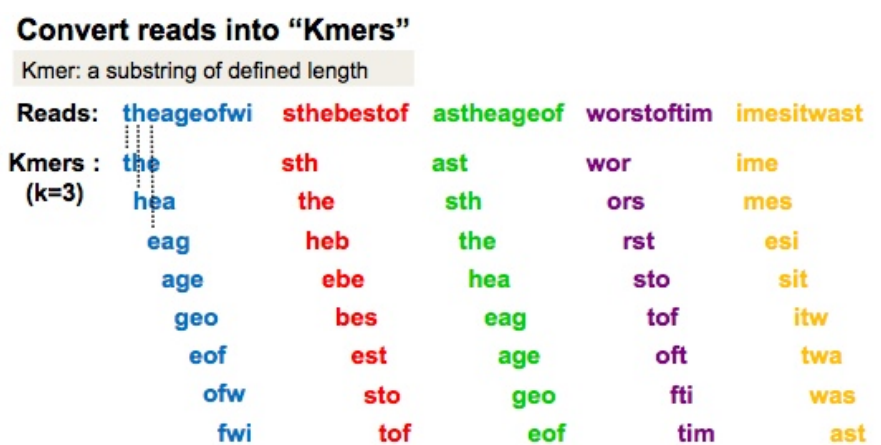
Figure : in the left,the word ”tatatatata$” undergoes cyclic shift and it is sorted in the right. Note that the BWT(tatatatata$) is a word (atttttaaaa$) with good clustering of T’s and A’s and so it can be written in a more compact way. The DNA is from an alphabet of 4 symbols so the clustering happens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

.

6 Next Generation sequencing (NGS): The Biological problem

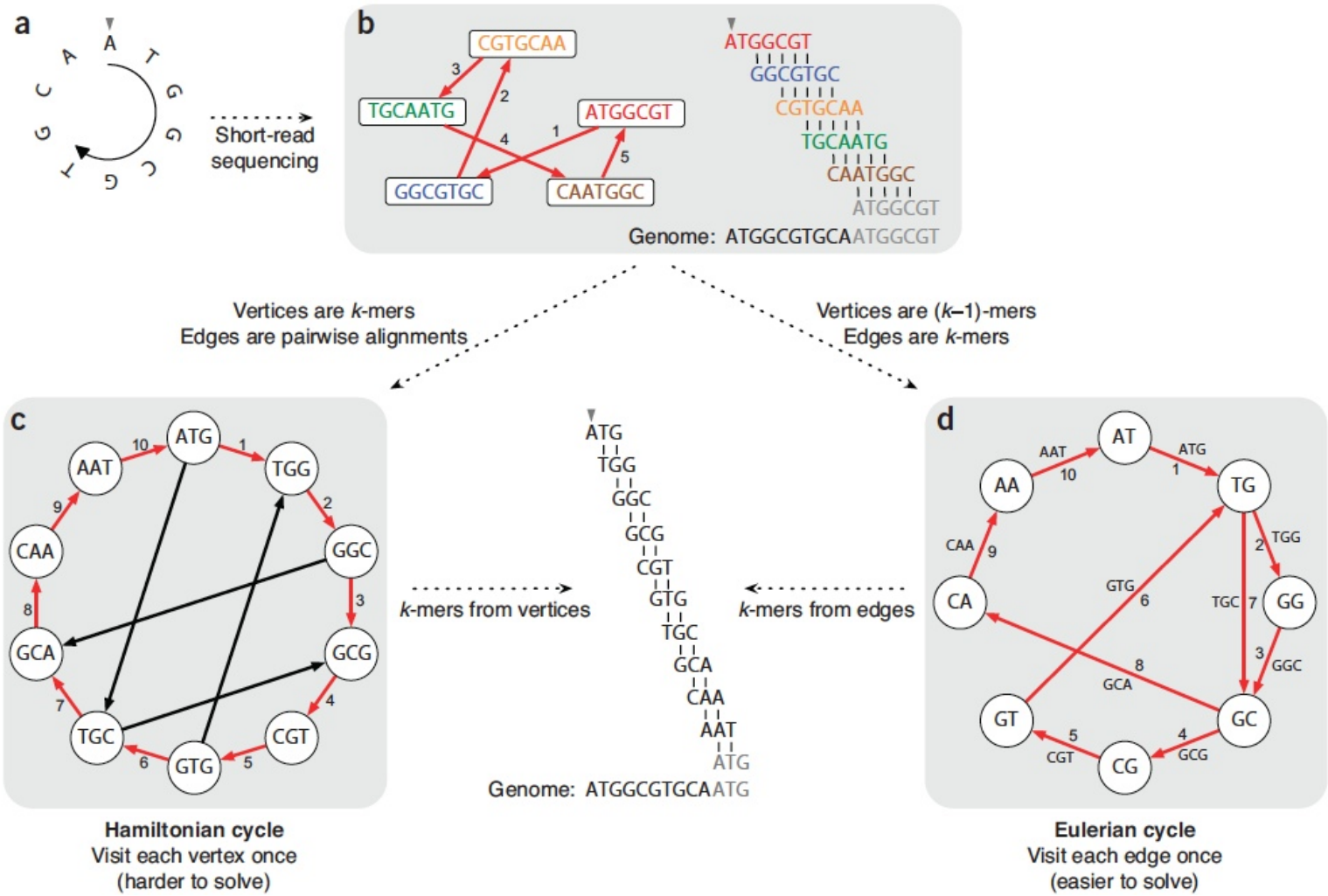
Instead of considering a DNA sequence, for sake of clarity, let's consider a sentence and we trim all spaces.

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



Three methods to reconstruct the original sequence (figure a)

one method (shown in b) uses the reads, the two other methods use k-mers derived from the reads (shown in c and d); see Compeau , Pevzner and Tesler Nature Biotechnology 29: 987 2011.



Graph approaches in alignment (previous figure)

(a) A small circular genome. In **(b)** reads are represented as nodes in a graph, and edges represent 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-based genome 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 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

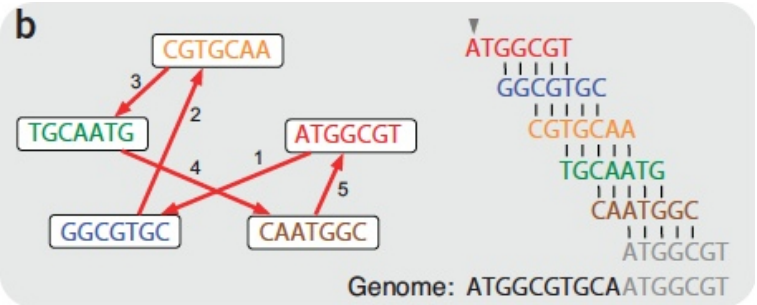


Figure b (see previous slide): The Hamiltonian graph is a graph in which each read is represented by a node and overlap between reads is represented by an arrow (called a directed edge) joining two reads. For instance, two nodes representing reads may be connected with a directed edge if the reads overlap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

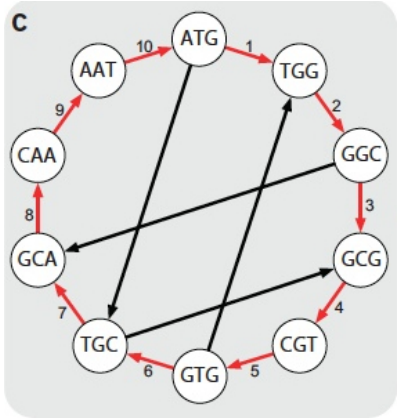


Figure c: The Hamiltonian cycle approach can be generalized to make use of k-mers by constructing a graph as follows. **First**, from a set of reads, make a node for every k-mer appearing as a consecutive substring of one of these reads. **Second**, given a k-mer, define its suffix as the string formed 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

Hamilton path is a graph that covers all vertex exactly once. When this path returns to its starting point than this path is called Hamilton cycle.

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 NP-Complete class of problems.

Next: Euler path is a graph using every edge of the graph exactly once. Euler cycle is a Euler path that returns to it starting point after covering all edges.

Eulerian graph

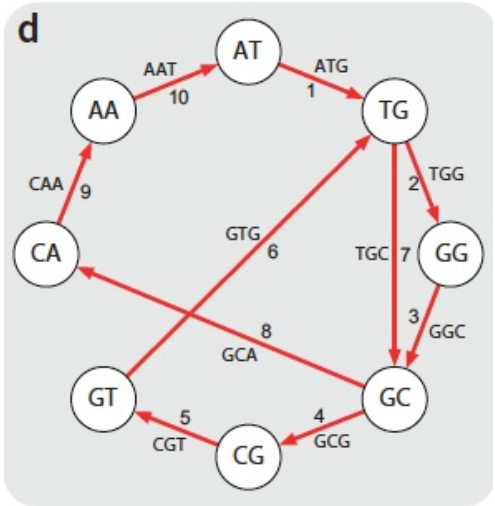


Figure d: Instead of assigning each k-mer contained in some read to a node, we will now assign each such k-mer to an edge. This allows the construction of a **de Bruijn** graph. First, form a node for every distinct prefix or suffix of a k-mer, meaning that a given sequence of length $k-1$. Then, connect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

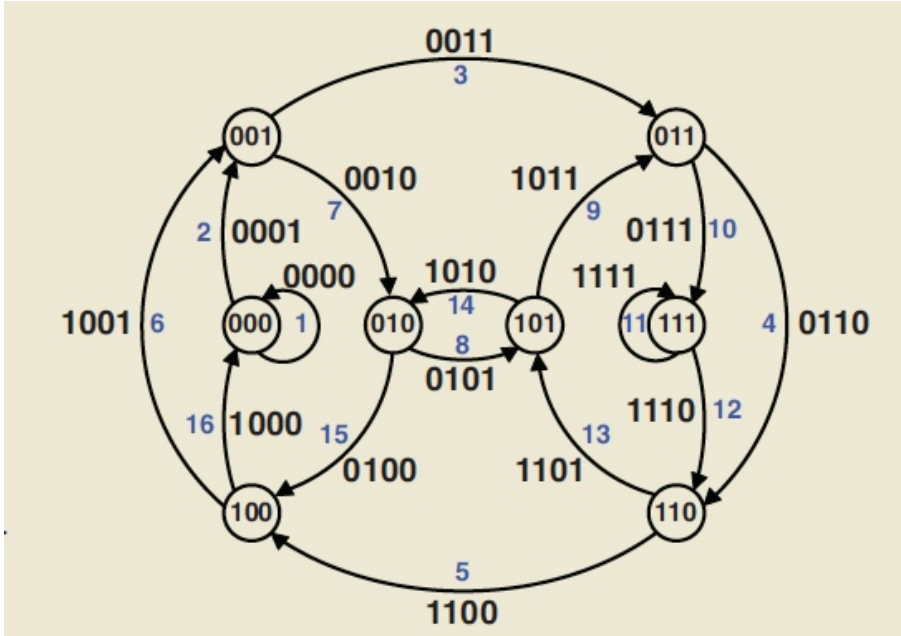
We visit all edges of the de Bruijn graph, which represents all possible k-mers; traveling will result in spelling out a candidate genome; for each edge that is traversed, one records the first nucleotide of the k-mer assigned to that edge. Euler considered graphs for which there exists a path between 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: representing the data as a graph

A 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 and software

The time required to run a computer implementation of Euler 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.

De Bruijn assembler method

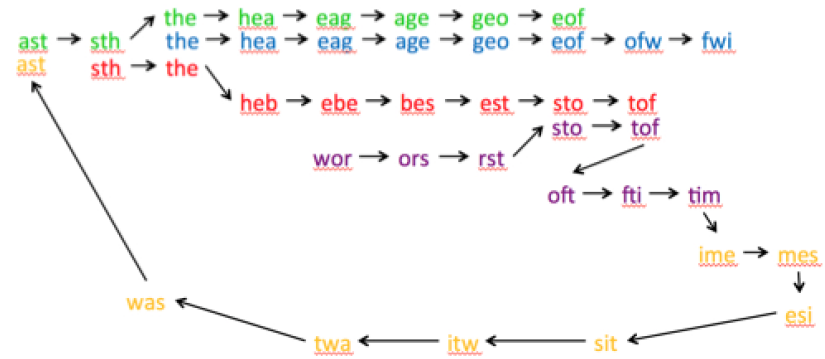
Sequencing is cheap, we generate sub-strings (reads) at random from throughout the genome. In next generation sequencing we have 10s of millions of reads. The difficult part is how we put them back together again in the right order. An intuitive way to do this may be in all versus all comparisons to 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 k-mers. As specified before a K-mer is a substring of defined length. **If we split reads in k-mers 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 k-mers rather than on the reads.

Details of the De Bruijn graph method

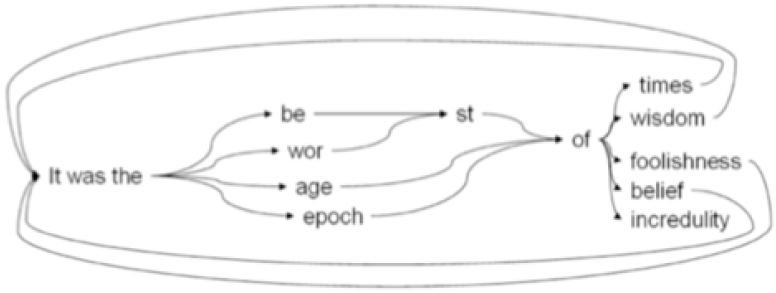
The next stage is to represent the stored k-mers in the De Bruijn graph. This is done by searching for overlaps of $k-1$. The graph has all consecutive k-mers by $k-1$ bases. Note that: 1) Adding k-mers from a second read of an overlapping region of the genome shows how the graph can be extended. It also reveals the redundancy in the data which need not be stored by the computer. This is how memory efficiency is achieved. 2) Adding a k-mers 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. **Long unbranched stretches represent unique sequence in the genome, branches and loops are the result of repeats.**

Details of the De Bruijn graph method

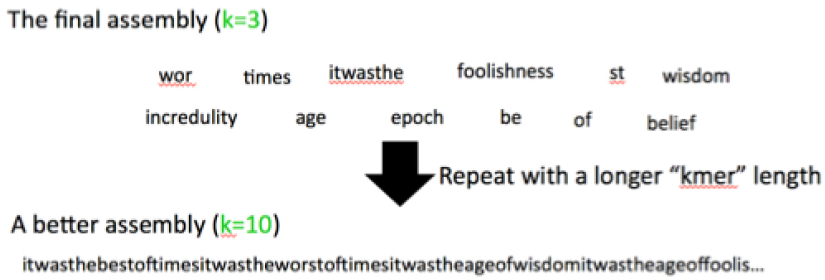
A



B



C

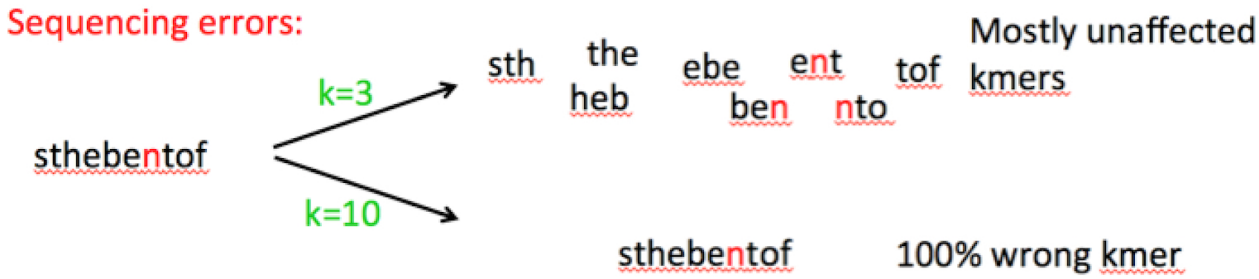


D

A, Kmerize the data; B, Build the graph; C, simplify the graph; D, get the final assembly.

Details of the De Bruijn graph method

The final step is to remove redundancy, result in the final De Bruijn Graph representation of the genome under study. Strengths and weaknesses of this approach: 1) a 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**. While this may be a satisfying solution to a computer scientist, it is not practically useful to a biologist who wants to annotate repeats (repeats are often not junk DNA).



De Bruijn method can only resolve k long repeat. Validation: look in your assembly for gene that should be there; N50: Weighted median such as 50% of your assembly is contained in contig of length $>= N50$

Software implementation:

Velvet: http://www.ebi.ac.uk/ zerbino/velvet/;

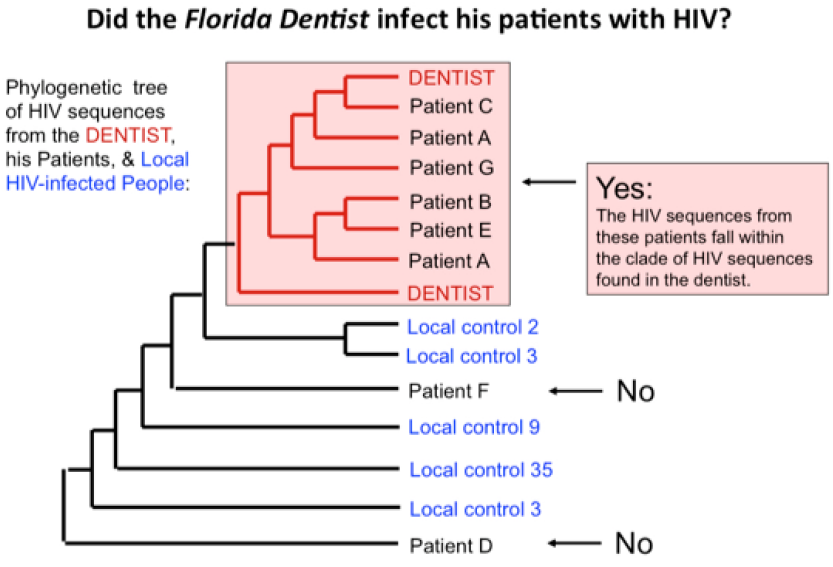
ABySS: http://www.bcgsc.ca/platform/bioinfo/software/abyss;

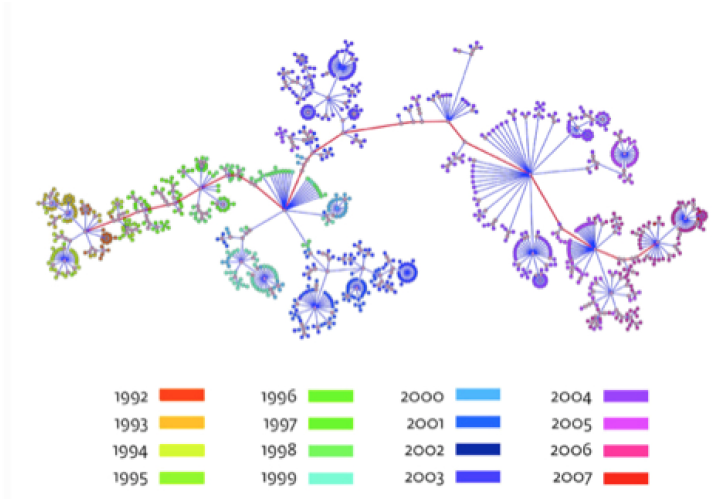
SOAP-denovo: http://soap.genomics.org.cn/soapdenovo.html;

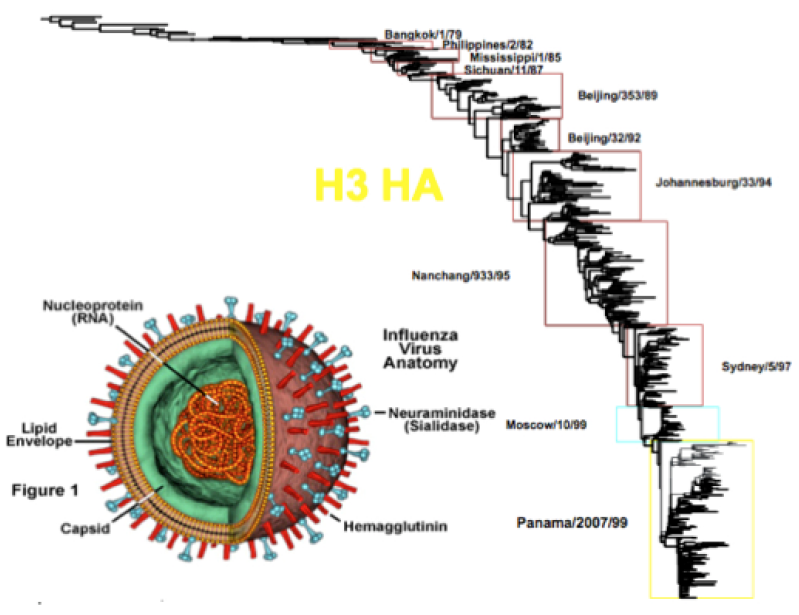
ALLPATH-LG: http://www.broadinstitute.org/software/allpaths-lg/blog/;

IDBA-UD: http://i.cs.hku.hk/ alse/hkubrg/projects/idba_ud

7 Relationships among multiple sequences, for example phylogeny: the inputs are multiple alignments, the outputs are trees







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 (except fossil). Here we use the terms species and taxa in a synonymous way. We compute the tree for each column of a multiple alignment.

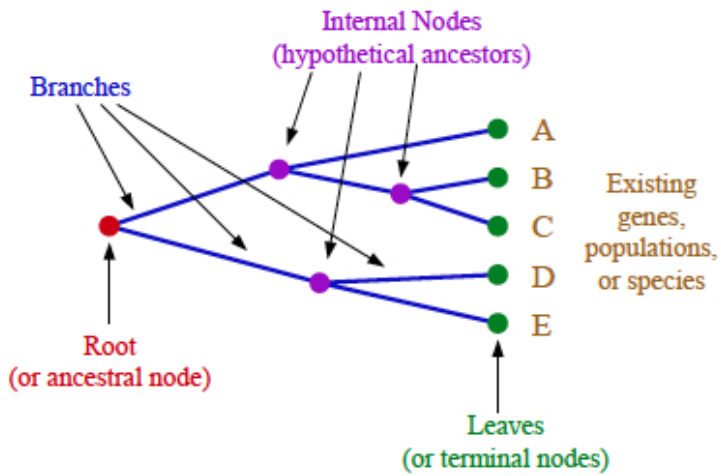
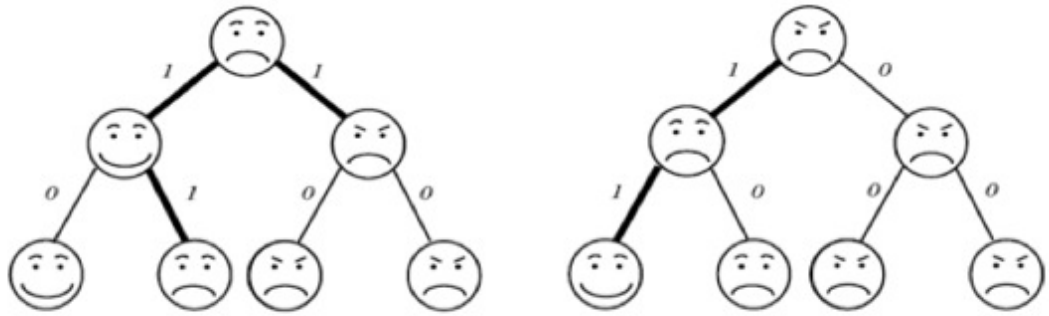


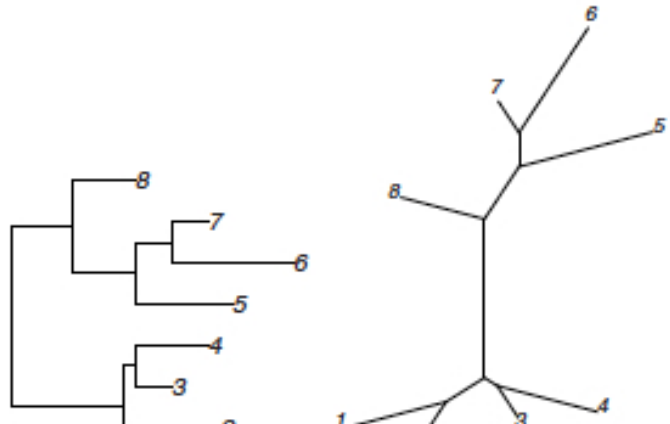
Figure : tree representation: $((a,(b,c)),(d,e))$; trees could also be unrooted

Phylogeny using parsimony (= economy of mutations)

Biological aims: from sequence alignment to phylogeny (a tree) by minimising the number of changes (mutations, see figure below from www.bioalgorithms.info). Parsimony means economy; there are two main algorithms (developed by Fitch and Sankoff); the output trees are rooted (below the difference between rooted, left, and unrooted trees, 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. $S_p \leftarrow S_q \bigcap S_r$
2. if $S_p = 0$ then
3. $S_p \leftarrow S_q \bigcup S_r$
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:

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

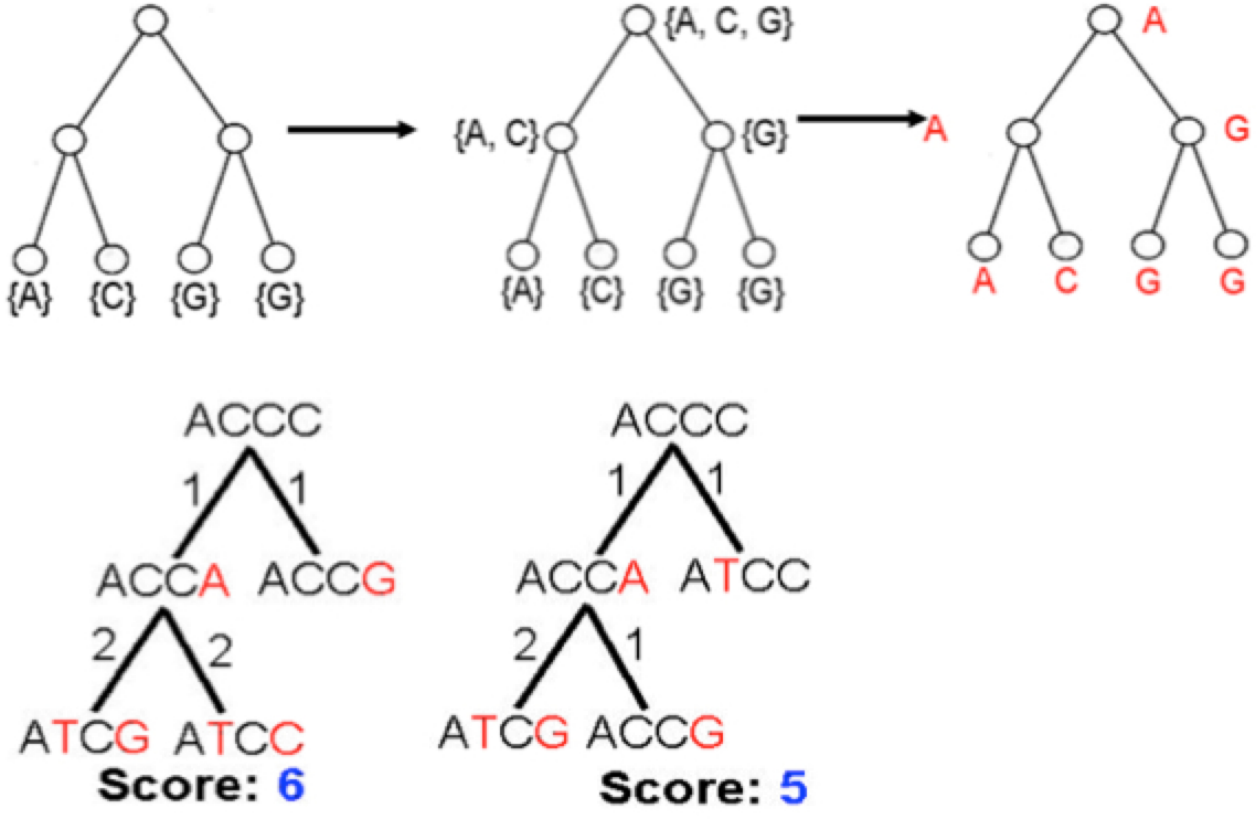
Assume that we have the final state set F_a of node a, which is the immediate ancestor of node p (S_p) that has two children q (S_q) and r (S_r).

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

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

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

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.

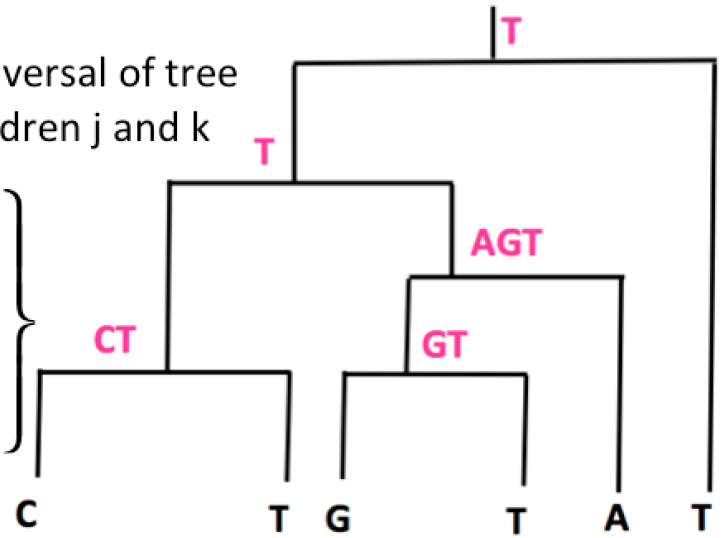


The figure in the top shows the Fitch two-step procedure. The tree is the hypothesis you are testing and you choose the tree that minimises the score. Bottom figure: you can sum the score for all the column of the alignment for each candidate tree and then you select the best tree. Choosing the candidate trees: there are algorithms for exploring the tree space.

Example of Fitch's algorithm

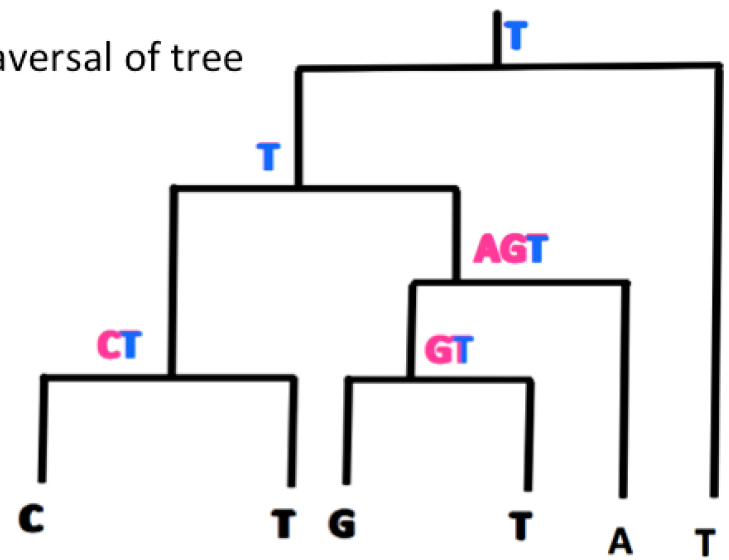
FITCH STEP1: Do a post-order (from leaves to root) traversal of tree
Find out possible states Ri of internal node i with children j and k

$$R_i = \begin{cases} R_j \cup R_k & if \quad R_j \cap R_k = \phi \\ R_j \cap R_k & otherwise \end{cases}$$



FITCH STEP2: Do a pre-order (from root to leaves) traversal of tree
Select state rj of internal node j with parent i

$$r_j = \begin{cases} r_i & if \quad r_i \in R_j \\ arbitrary\ state \in R_j & otherwise \end{cases}$$



Sankoff general parsimony: each mutation costs differently

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. $F_p \leftarrow F_p \bigcup j$
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 $O(mnk^2)$.

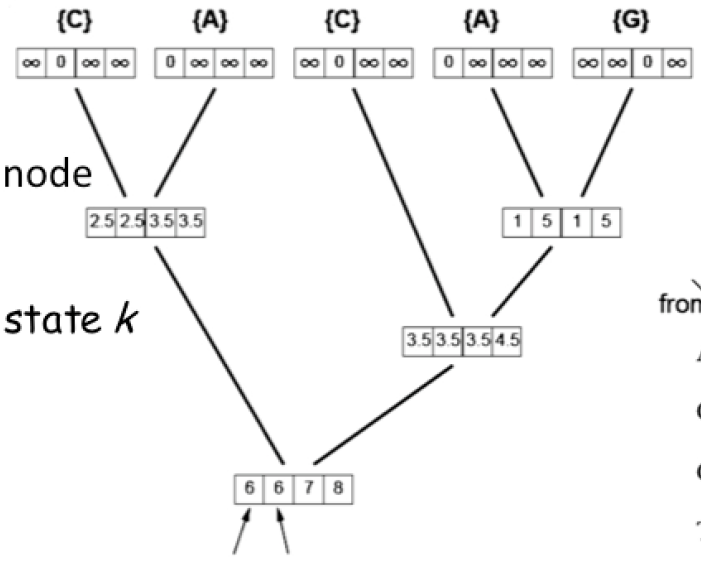
Example of Sankoff's algorithm

Let $\mathbf{c_{ij}}$ the matrix cost (on the right) and $\mathbf{S_v(k)}$ be the smallest number of steps needed to evolve the subtree at or above node v, given that node **v** is in state **k**.

$$S_v(k) = \begin{cases} 0 & \text{if node } v \text{ has (or could have) state } k \\ +\infty & \text{otherwise} \end{cases}$$

If *v* is a leaf (tip)
If *v* is a node with descendants *u* and *w*
The minimum number of (weighted) steps is

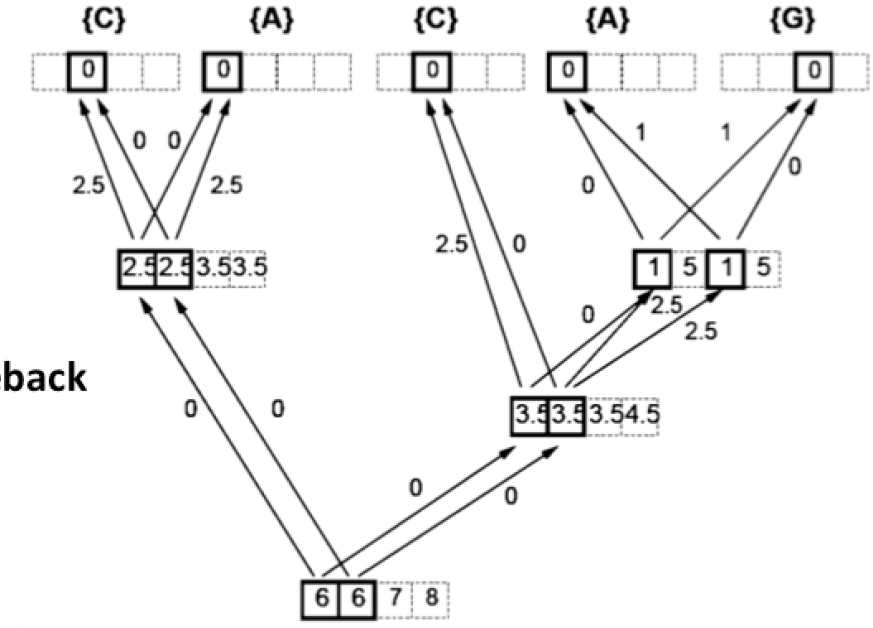
$$S^* = \min_k S_{root}(k)$$



cost matrix:

from \ to	A	C	G	T
A	0	2.5	1	2.5
C	2.5	0	2.5	1
G	1	2.5	0	2.5
T	2.5	1	2.5	0

Traceback



Sankoff: example of downpass

(A)

C_{ij}	a	g	c	t
a	0	1	3	3
g	1	0	3	3
c	3	3	0	1
t	3	3	1	0

(B)

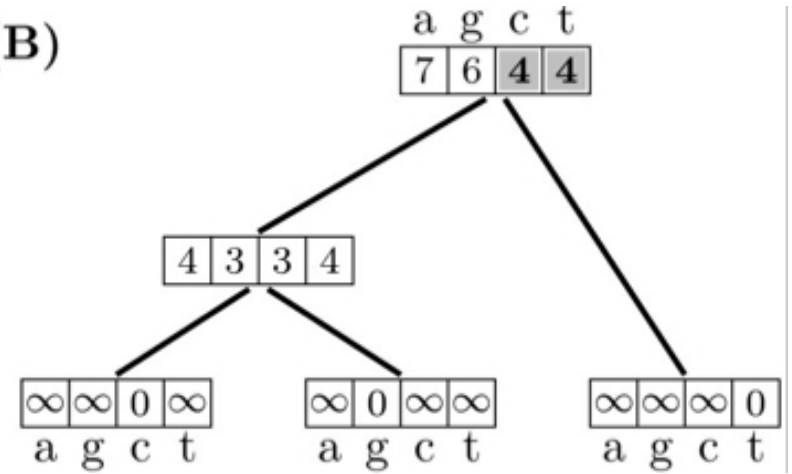


Figure : If the leaf has the character in question, the score is 0; else, score is ∞ Each mutation $a- > b$ costs the same in Fitch and differently in Sankoff parsimony algorithm (weighted matrix in A). An example of a weighted matrix for Sankoff (for proteins) is the Blosum, presented before in this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

example of uppass

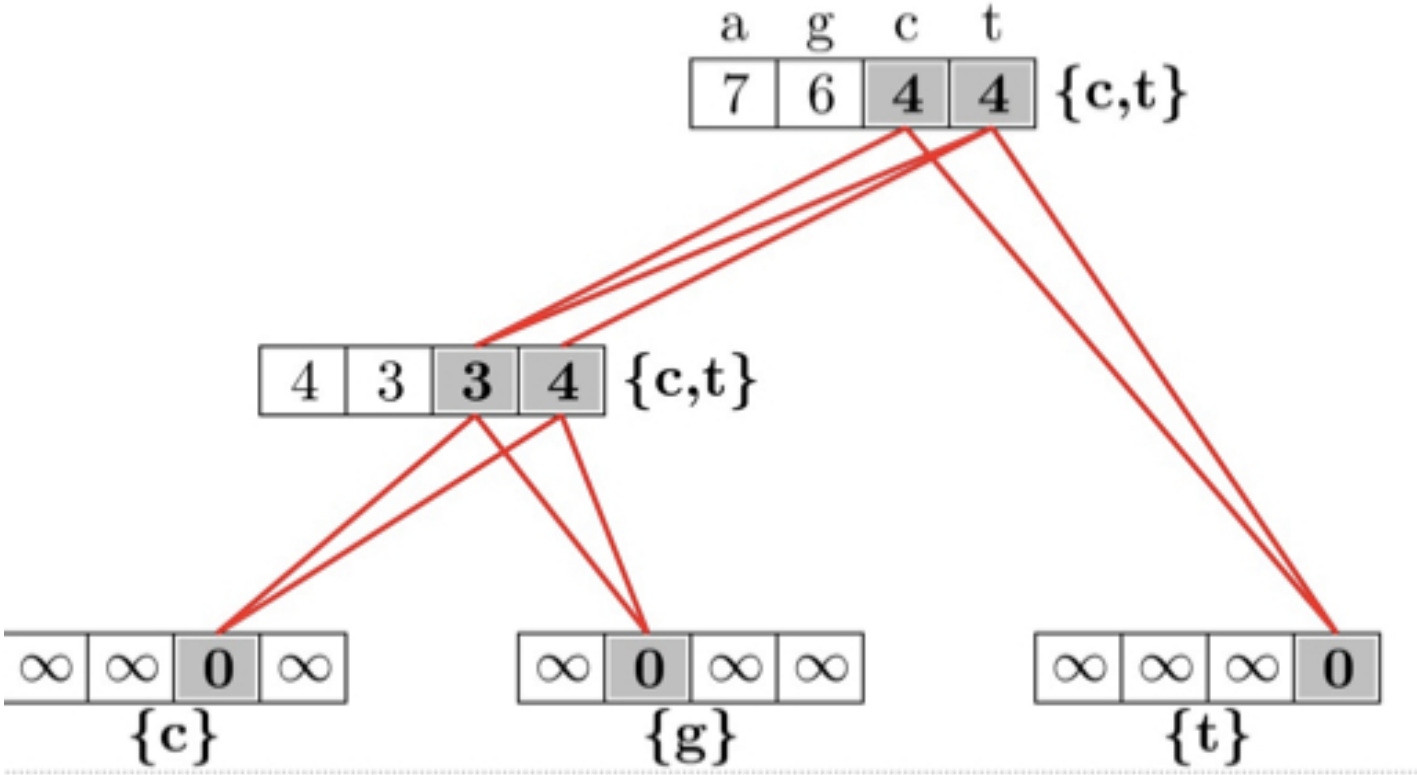
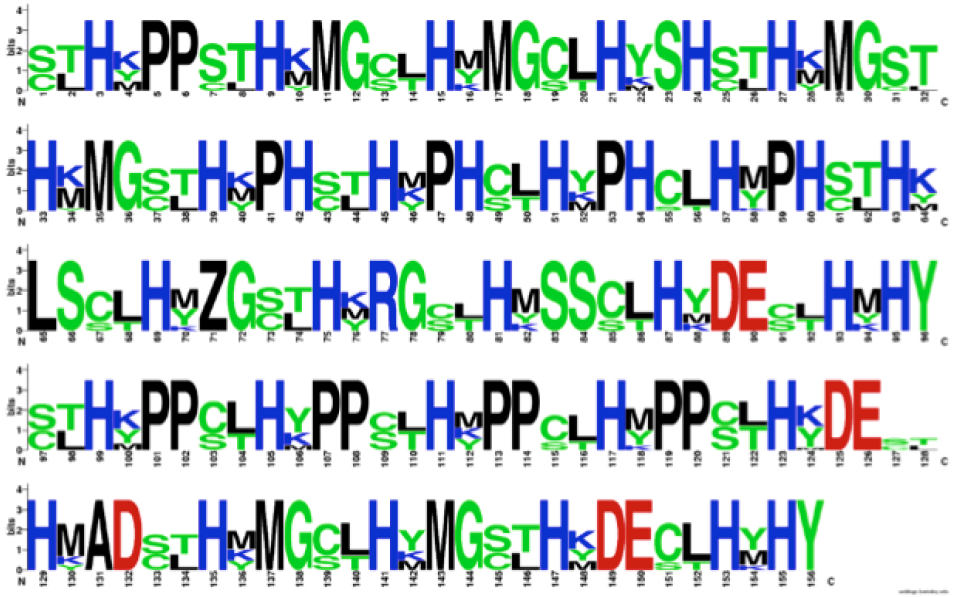
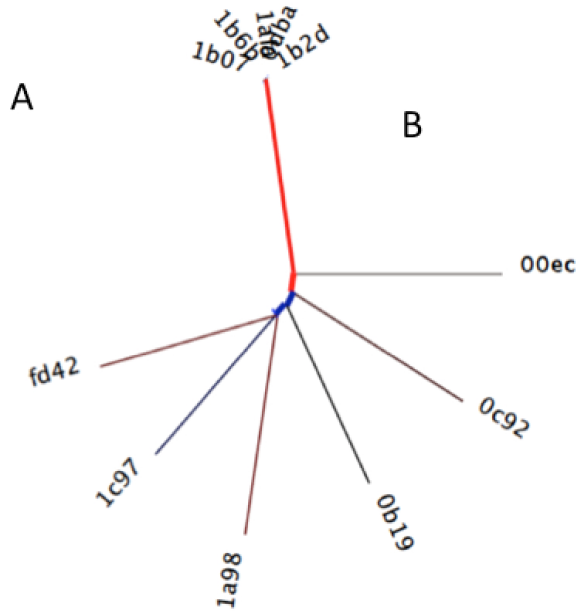


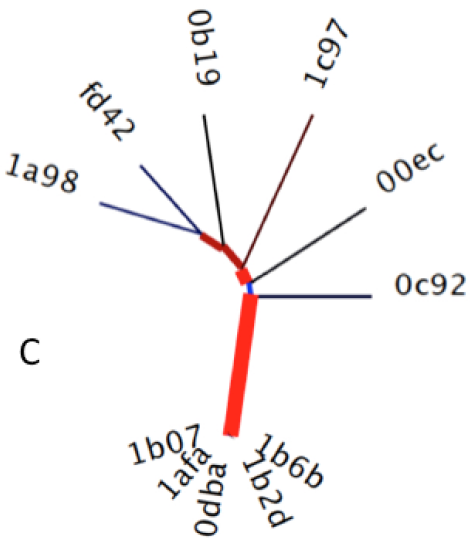
Figure : Example of Sankoff algorithm. Note that in the parsimony approaches (Fitch and Sankoff) the tree (i.e. the topology and leaves order) is the hypothesis you are testing. So you try different trees and select the one that is most parsimonious for each column of the alignment, then you select the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

Phylogeny (distance based algorithms)

Algorithms: UPGMA, Neighbor Joining







Distance algorithm in computer science

A) A sequence logo for the FakeAV-DO function " F1 ". Positions with large characters indicate invariant parts of the function; positions with small characters vary due to code metamorphism

B) A neighbour joining tree of FakeAV-DO set of procedures F1.

C) Neighbor joining tree of FakeAV-DO set of procedures F2 from the same samples of B.

(W.M. Khoo Unity in diversity: Phylogenetic-inspired techniques for reverse engineering and detection of malware families)

8 Phylogeny based on a matrix of distances

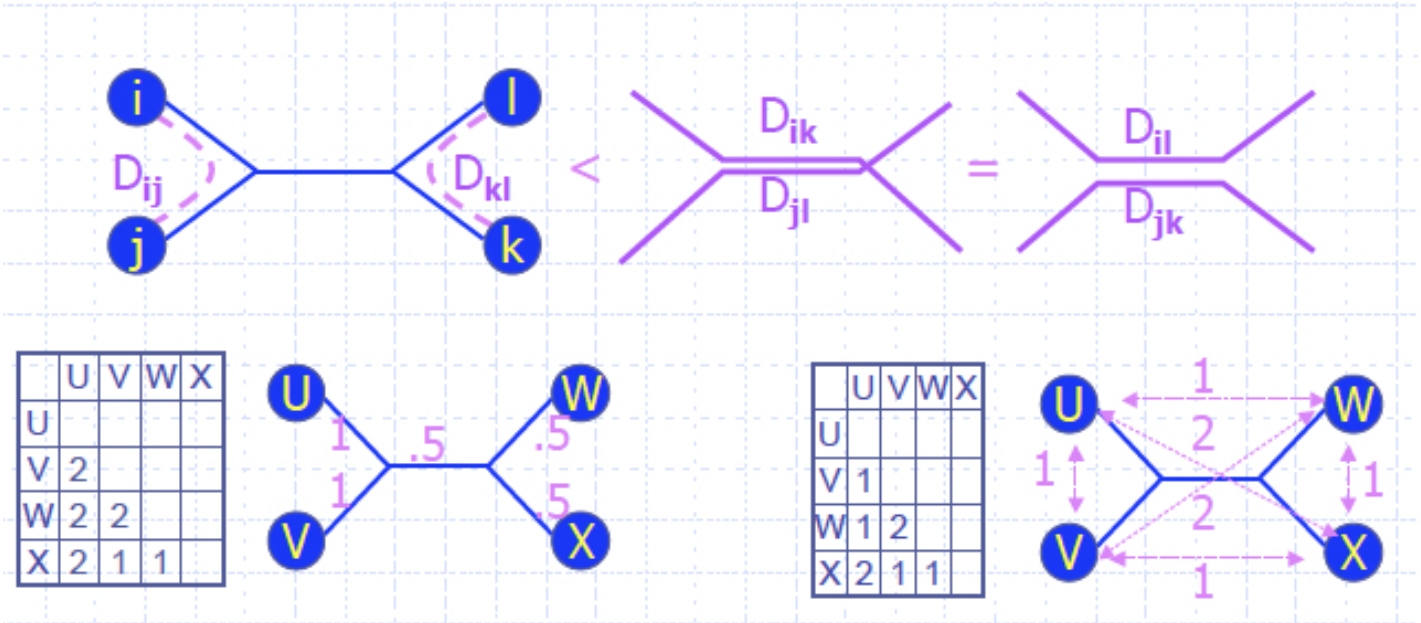
Distance methods convert the changes counted in each column of the alignment, top figure, into a single distance matrix, bottom figure (dissimilarity matrix= 1 - similarity) to construct a tree and are kin to clustering methods. We can use the same matrix we use for Blast search, for example the Blosum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

Species	Characters
A	ACTGTTCGTTCTGA
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	–

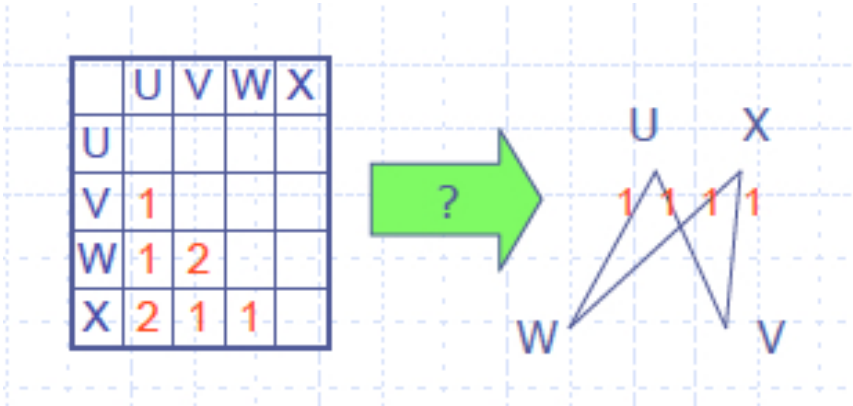
Additivity: when a distance matrix could be converted into a tree

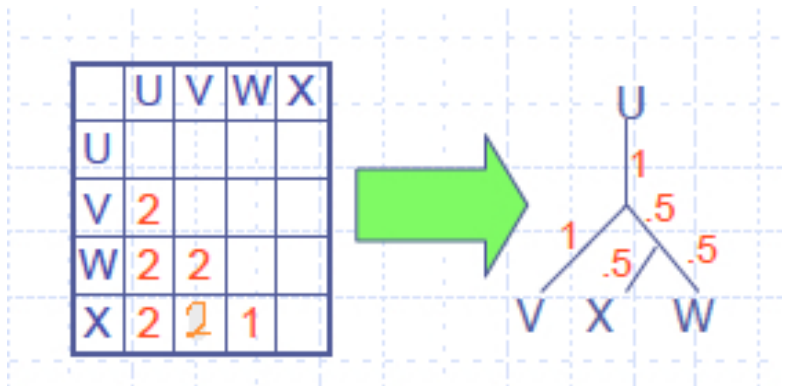
A matrix D is additive if for every four indices i,j,k,l we can write the following: $D_{ij} + D_{kl} \leq D_{ik} + D_{jl} = D_{il} + D_{jk}$. If the distance matrix is not additive we could find the tree which best fits he distance matrix.



The additivity property

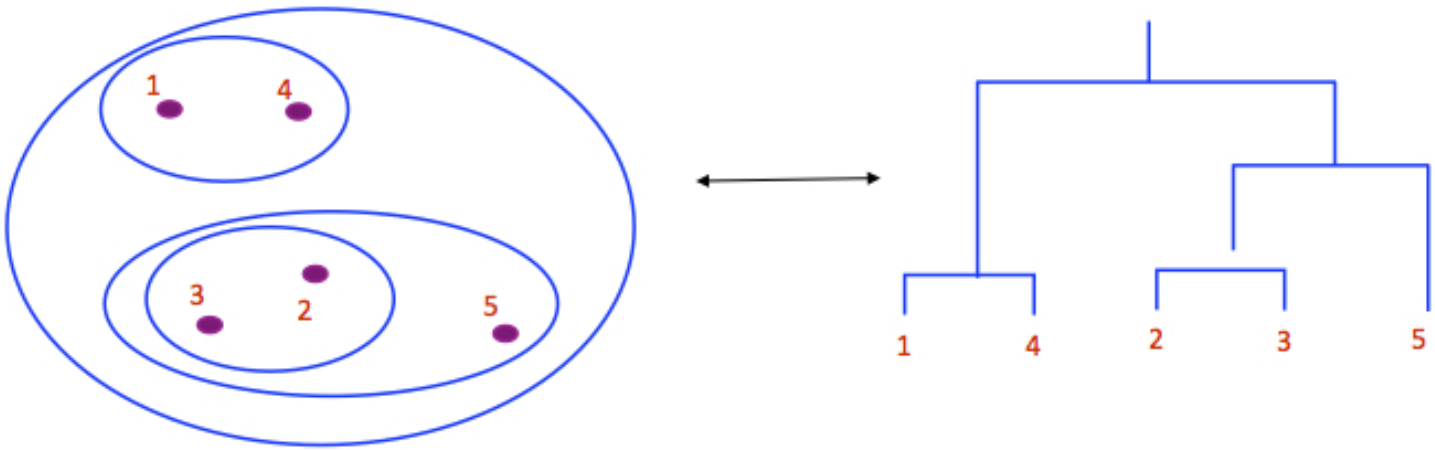
Top: distance matrix does not turn into a tree; Bottom: the distance matrix turns into a tree.





UPGMA: Unweighted Pair Group Method with Arithmetic Mean

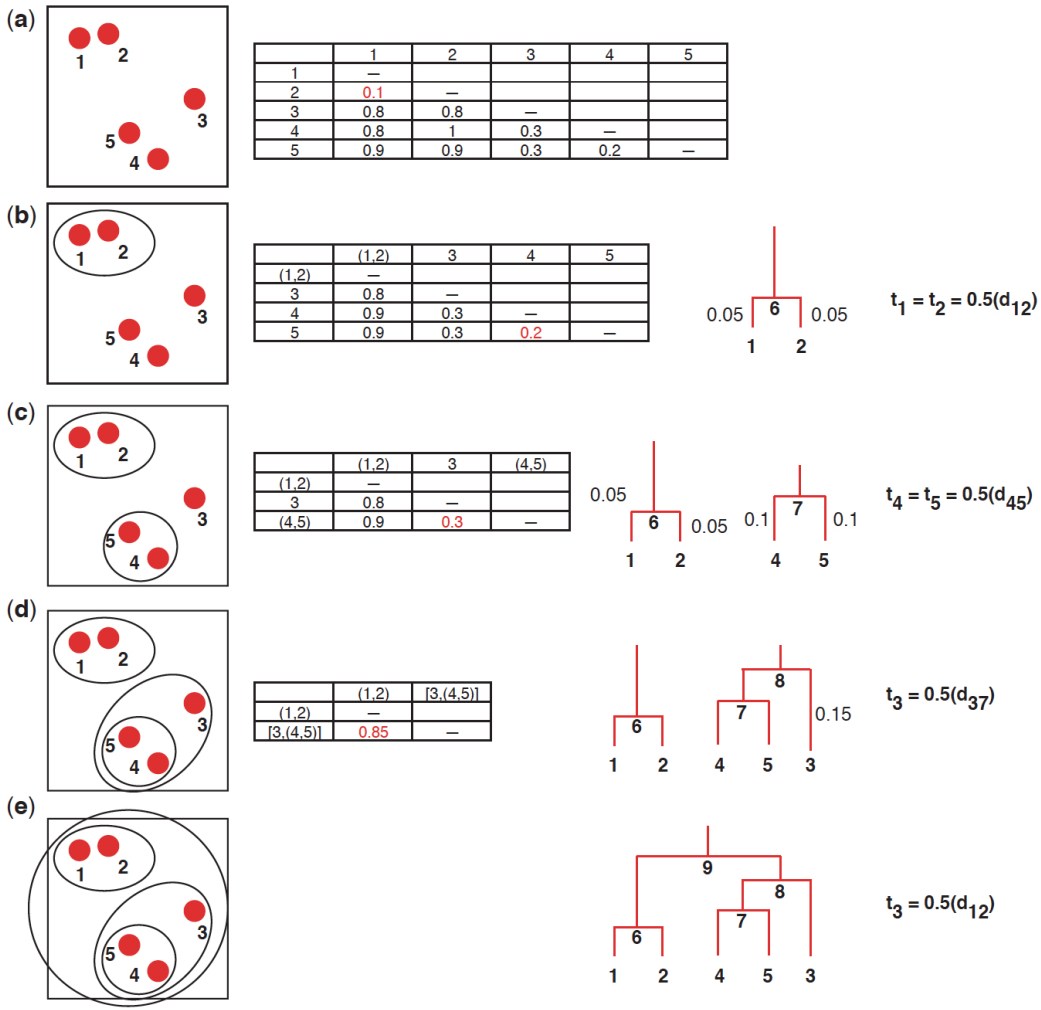
UPGMA is a sequential clustering algorithm that computes the distance between clusters using average pairwise distance and assigns a height to every vertex in the tree, effectively assuming the presence of a molecular clock and dating every vertex. The algorithm produces an ultrametric tree : the distance 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 $C_k = C_i \bigcup C_j$ 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

UPGMA



Example of UPGMA from P. Pevzner; in UPGMA when choosing the closest pair, we do not take into account the distance from all the other nodes (as we do in Neighbor Joining).

Neighbor Joining, NJ

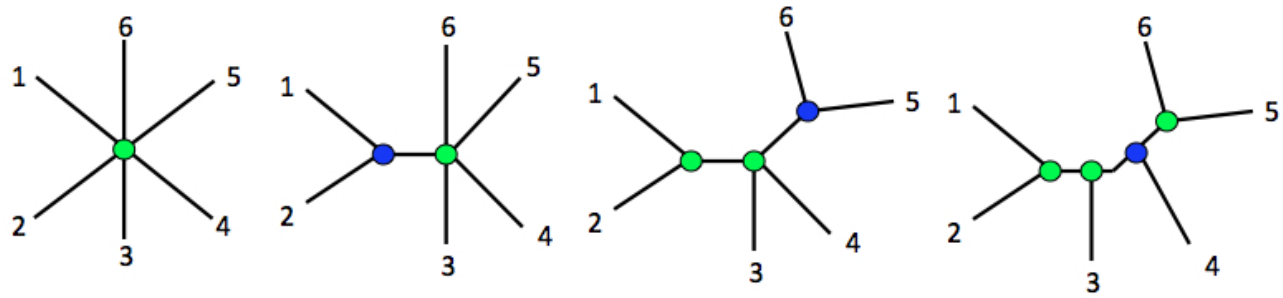
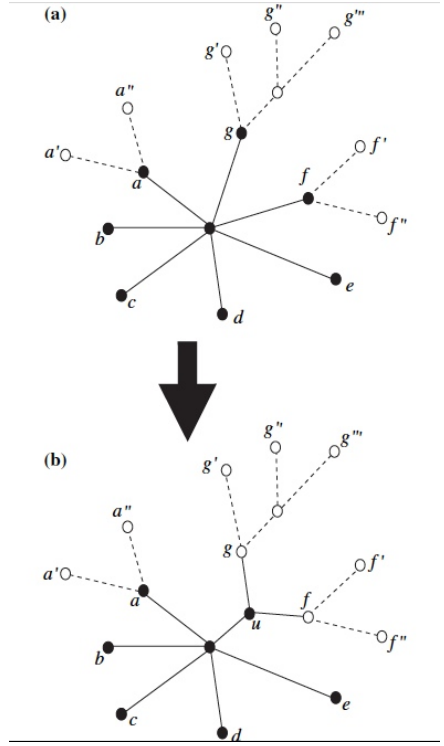


Figure : NJ starts with a star topology (i.e. no neighbors have been joined) and then uses the smallest distance in the distance matrix to find the next two pairs to move out of the multifurcation then recalculate the distance matrix that now contains a tip less.

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.



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 a time

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_1	C	D	E
C	3			
D	6	7		
E	5	6	5	
F	7	8	9	8

	U_1	C	U_2
C	3		
U_2	3	4	
F	7	8	6

	U_2	U_3
U_3	2	
F	6	6

	U_4
F	5

Step 1

S calculations

S_x = (sum all D_x)/($N-2$), where N is the # of OTUs in the set.

$S_A = (5+4+7+6+8)/4 = 7.5$
$S_B = (5+7+10+9+11)/4 = 10.5$
$S_C = (4+7+7+6+8)/4 = 8$
$S_D = (7+10+7+5+9)/4 = 9.5$
$S_E = (6+9+6+5+8)/4 = 8.5$
$S_F = (8+11+8+9+8)/4 = 11$

$S_{U_1} = (3+6+5+7)/3 = 7$
$S_C = (3+7+6=8)/3 = 8$
$S_D = (6+7+5+9)/3 = 9$
$S_E = (5+6+5+8)/3 = 8$
$S_F = (7+8+9+8)/3 = 10.6$

$S_{U_1} = (3+3+7)/2 = 6.5$
$S_C = (3+4+8)/2 = 7.5$
$S_{U_2} = (3+4+6)/2 = 6.5$
$S_F = (7+8+6)/2 = 10.5$

$S_{U_2} = (2+6)/1 = 8$
$S_{U_3} = (2+6)/1 = 8$
$S_F = (6+6)/1 = 12$

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

Step 2

Calculate pair with smallest (M), 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
$M_{CU_1} = 3 - 7 - 8 = -12$
$M_{DE} = 5 - 9 - 8 = -12$
Choose one of these (DE here).

Smallest is
$M_{CU_1} = 3 - 6.5 - 7.5 = -11$

Smallest is
$M_{U_2F} = 6 - 8 - 12 = -14$
$M_{U_3F} = 6 - 8 - 12 = -14$
$M_{U_2U_3} = 2 - 8 - 8 = -14$
Choose one of these ($M_{U_2U_3}$ here).

Step 3

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

U_1 joins A and B:
$S_{AU_1} = D_{AB}/2 + (S_A - S_B)/2 = 1$
$S_{BU_1} = D_{AB}/2 + (S_B - S_A)/2 = 4$

U_2 joins D and E:
$S_{DU_2} = D_{DE}/2 + (S_D - S_E)2 = 3$
$S_{EU_2} = D_{DE}/2 + (S_E - S_D)/2 = 2$

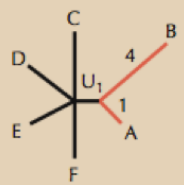
U_3 joins C and U_1:
$S_{CU_3} = D_{CU_1}/2 + (S_C - S_{U_1})/2 = 2$
$S_{U_1U_3} = D_{CU_1}/2 + (S_{U_1} - S_C)/2 = 1$

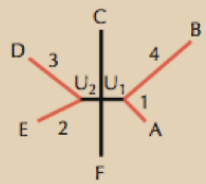
U_4 joins U_2 and U_3:
$S_{U_2U_4} = D_{U_2U_3}/2 + (S_{U_2} - S_{U_3})/2 = 1$
$S_{U_3U_4} = D_{U_2U_3}/2 + (S_{U_3} - S_{U_2})/2 = 1$.

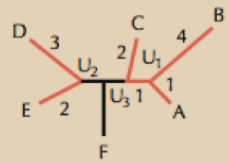
For last pair, connect U_4 and F with branch length = 5.

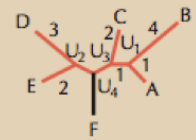
Step 4

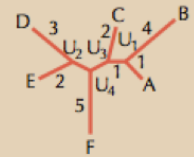
Join *i* and *j* according to *S* above and make all other taxa in form of a star. Branches in black are of unknown length. Branches in red are of known length.











Step 5

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

Comments

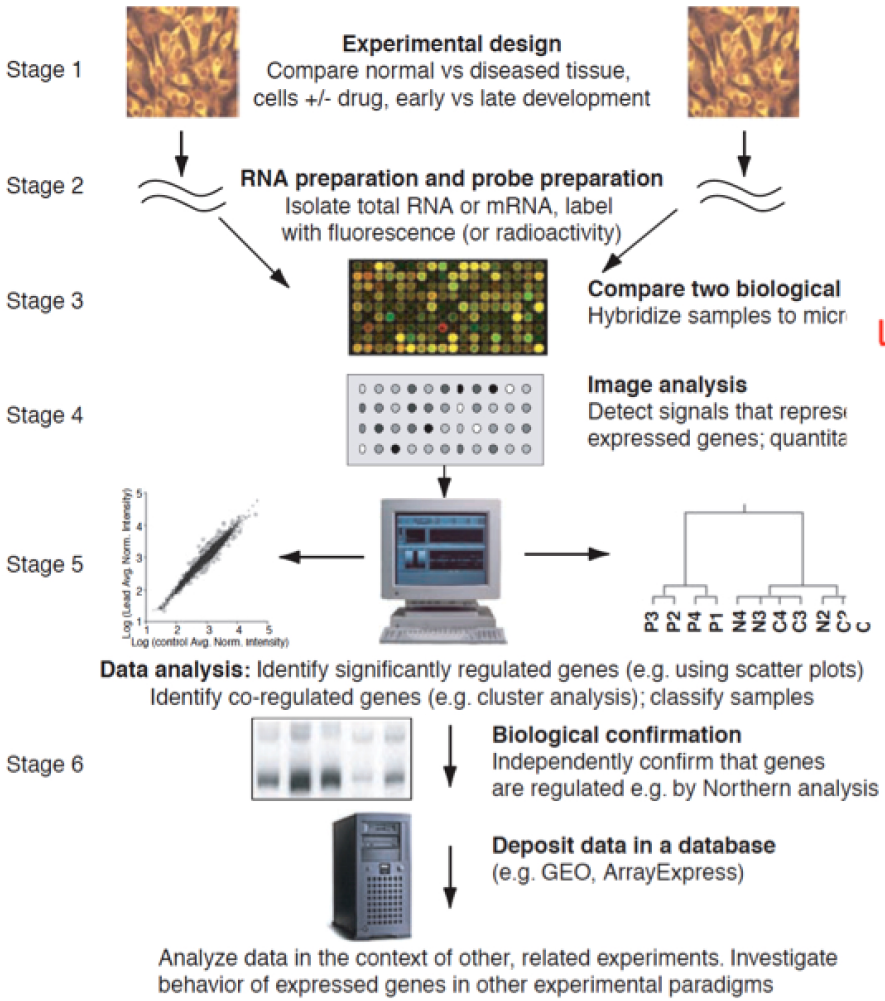
Note this is the same tree we started with (drawn in unrooted form here).

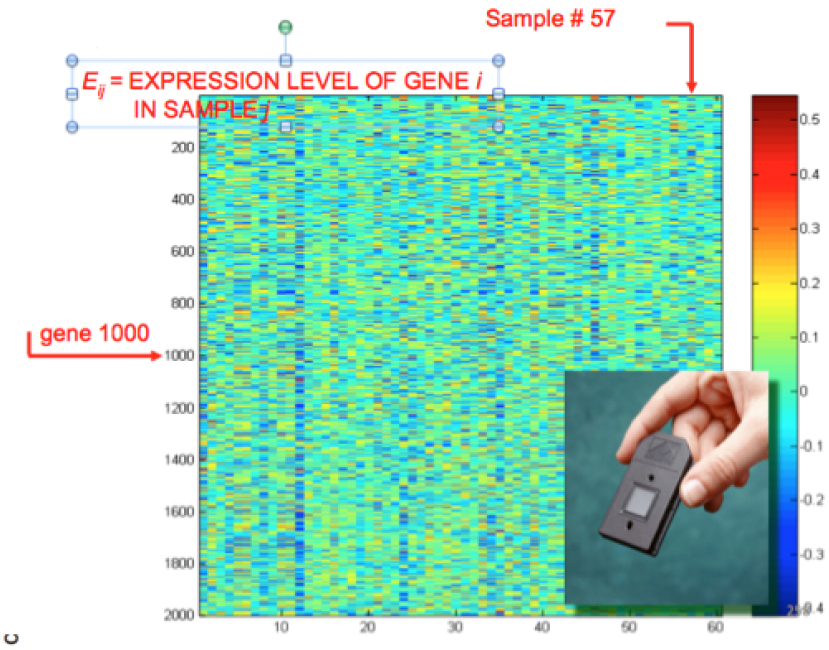
figure taken from www.evolution-textbook.org/content/free/tables/Ch_27/T11_EVOW_Ch27.jpg

9 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.

Overview of generating high throughput gene expression data





From P. Pevzner

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

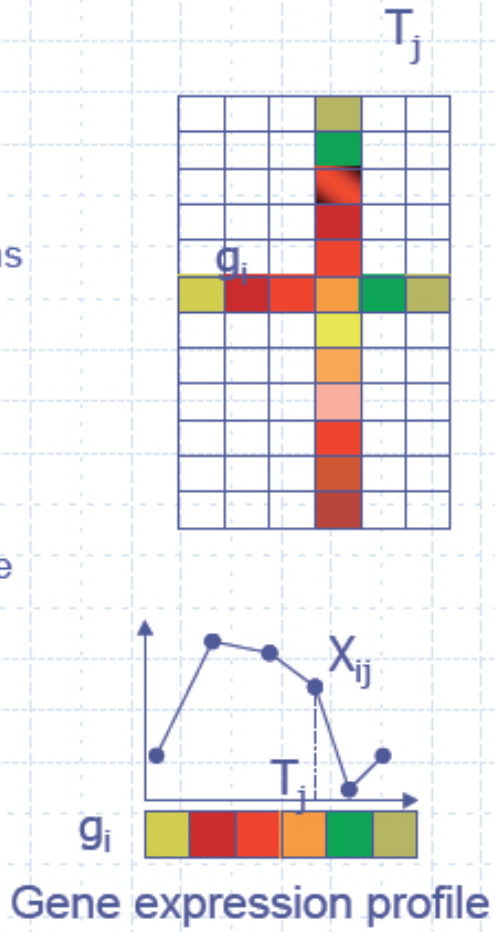


Figure : The color of the spot indicates activation with respect to control (red) or repression with respect to the control (green) or absence of regulation (yellow) of a gene, or error in the technological process (black). The sample can be all the genes of an organism (example the 6000 genes of yeast),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

Gene expression data: example of output

Type 2 Diabetes		Type 1 Diabetes		Renal disorder		Osteosarcoma		Breast cancer		Multiple myeloma	
Gene Symbol	P.Value	Gene Symbol	P.Value	Gene Symbol	P.Value	Gene Symbol	P.Value	Gene Symbol	P.Value	Gene Symbol	P.Value
AKT1	0.029425624	AKT3	0.018507548	CCR5	0.048470818	AKT2	0.000126941	AKT3	0.002023147	AKT1	0.00033578
AKT2	0.000173708	BMPR2	0.035076531	CSF1R	0.005000858	BMP2	0.007828969	BMP2	0.000340725	AKT2	0.042736344
AKT3	0.000222778	CCR5	0.00089995	IFI27	0.017479806	BMPR2	0.004351855	CCR5	0.005465932	AKT3	0.006651145
CSF1	0.013497857	CSF1	0.03300604	IGF1R	0.026672719	CCR5	0.015328671	IGF1	0.04354415	CCR5	3.95E-06
IFI44L	0.047309419	IFI44L	0.047610377	LRP6	0.01990424	CSF1R	0.00023377	IGF1R	0.000117596	CSF1	4.89E-05
IL6	0.036438795	LRP6	0.037451428	MAPK10	0.013598593	IFI27	0.003857486	IL6	0.009510583	CSF1R	4.23E-05
MAP3K7	0.00084375	MAPK13	0.007592876	NFKB2	0.019986021	IFI44L	6.39E-06	MAPK1	0.007741045	CSF2	0.000960386
MAPK1	0.015620066	MAPK4	0.030868863	PDGFB	0.03952656	IGF1R	0.014488834	MAPK10	0.02637407	LRP5	0.025696961
MAPK11	0.031190453	NFKB2	0.009642252	PDGFD	0.039240707	LRP5	0.018277828	MAPK7	0.045413004	LRP6	0.005487107
MAPK12	0.02749298	NFKBIA	1.37E-05	PDGFRA	0.031118696	LRP6	0.018570379	NFKBIA	0.007283022	MAP3K7	0.000670421
MAPK14	0.003496727	NUP62	0.010822798	TGFA	0.049134423	MAP3K7	0.00319979	NUP62CL	0.012101499	MAPK1	0.000980307
MAPK3	0.028945413	PDGFA	0.042711089	TGFB1	0.049865661	MAPK1	0.049557587	PDGFC	0.001994966	MAPK10	0.001378709
NFKB1	0.003953564	PDGFC	0.004640007	TGFBI	0.038639089	MAPK10	0.007942657	PDGFC	0.003910123	MAPK11	0.01278082
NFKB2	0.003812117	SQSTM1	0.037820278	TNFAIP3	0.005211327	MAPK11	9.03E-05	PDGFD	0.010410547	MAPK14	0.024418313
NFKB2	0.019309942	TGFA	0.046157749	TNFAIP6	0.018484587	MAPK12	0.020524185	REL	0.028959611	MAPK4	0.000213052
NFKBIA	0.000791186	TGFB3	0.041756839	TNFRSF1B	0.038860169	MAPK13	0.021601197	TGFB2	0.023637028	MAPK6	0.011024085
PDGFRA	0.022393241	TGFBI	0.018586844	TNFRSF21	0.03935251	MAPK14	0.004123891	TGFB3	0.004777696	MAPK7	0.001493782
REL	0.000888489	TGFBR2	0.001134511	TRIM2	0.001054913	MAPK4	0.000532563	TGFBR1	0.021841086	MAPK8	5.79E-07
RELA	0.000299556	TNFAIP1	0.009489591			MAPK8IP3	0.000360873	TGFBR2	0.044373469	MAPK8	6.17E-05
TGFA	0.002730714	TNFAIP3	1.77E-07			MAPK9	0.001587219	TGFBR3	8.51E-06	MAPK9	0.005178387
TGFB1	0.000835585	TNFAIP6	0.033038618			NFKB1	0.000187087	TNFAIP8L2	0.028594915	NFKB2	8.89E-05
TGFBR2	0.046911857	TNFRSF10C	0.016132998			NFKB2	1.39E-07	TNFRSF12A	0.005712663	NUP62	0.003021797
TNFAIP1	0.00063572	TNFRSF13B	0.013286132			NUP62	0.000634764	TNFRSF19	0.042594685	PDGFA	0.003534272
TNFAIP2	0.010169047	TNFRSF17	0.000850768			NUP62	0.004907568	TNFSF13	0.035351729	PDGFB	0.003508453
TNFAIP3	0.002634127	TNFRSF1A	0.004647921			NUP62CL	5.62E-08	TNFSF13B	0.015401105	PDGFRA	0.000164907
TNFRSF10B	0.032735653	TNFSF14	0.000356257			PDGFB	0.002911584	TNFSF15	0.00560085	REL	0.001828589
TNFRSF13B	0.020690122	TNFSF18	0.015538582			PDGFC	0.039650276	TNFSF4	0.002910131	RELA	6.76E-05
TNFRSF14	0.001387678	TRAF6	0.034594707			PDGFD	0.000267756			RELB	0.001237689
TNFRSF21	0.038752772					PDGFRA	7.47E-05			SQSTM1	0.001590045

Figure : Data downloaded from geo omnibus and analised with the software Limma; the genes are ranked with respect to their p-value. An observed event is significant if it is unlikely to have occurred by chance. The significance of the event is also called its p-value, a real number in the interval [0, 1]. The smaller the p-value, the more significant the occurrence of the event.

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 Ci 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 $\sum v \setminus |C|$ 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. 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 cost(P) $-$ cost($P_{i\rightarrow C}$) $>$ bestChange
8. bestChange $\leftarrow$ cost(P) $-$ cost($P_{i\rightarrow C}$)
9. $i^{'} \leftarrow$ i
10. $C^{'} \leftarrow$ C
11. if bestChange $>$ 0
12. Change partition P by moving $i^{'}$ to $C^{'}$
13. else
14. return P

Progressive greedy K-means Algorithm

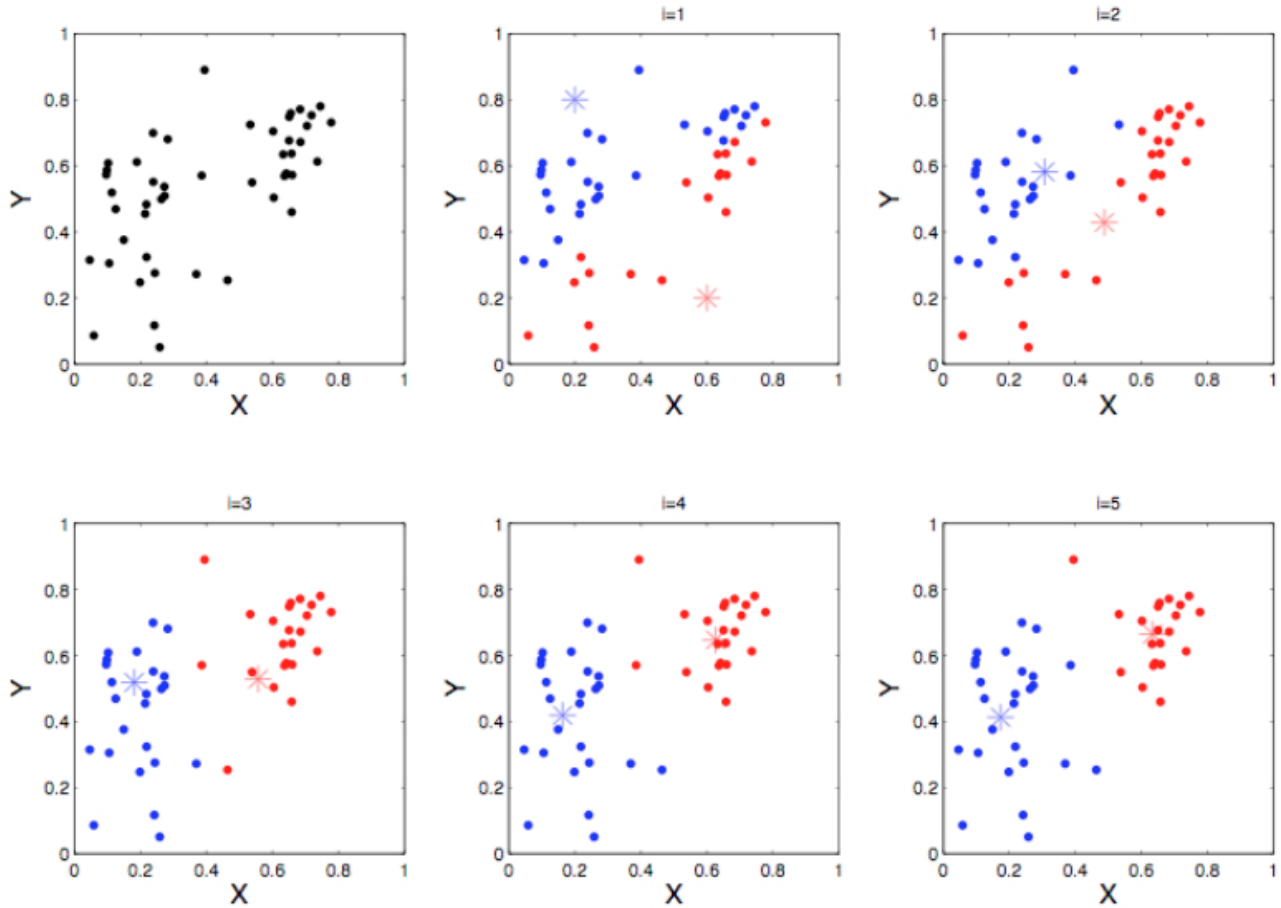
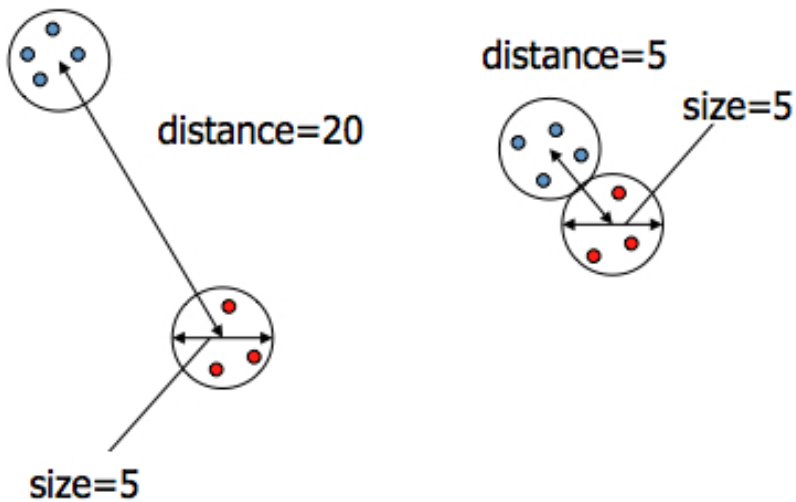


Figure : K-means progression from left to right and top to bottom; stars are center points (the centers of the cluster).

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 clustering on microarray data

The aims is clustering gene expression data: it is easy to interpret the data if they are partitioned into clusters combining similar data points.

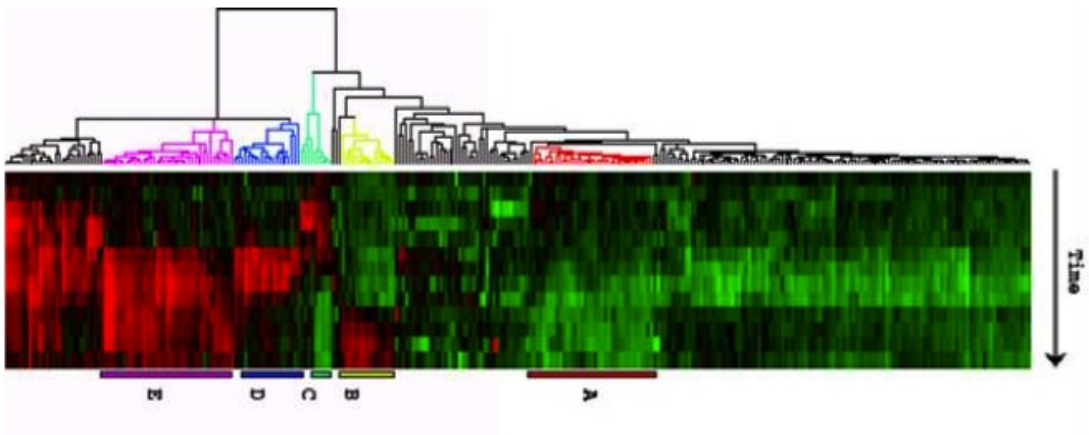


Figure : Clustering analysis obtained using Hierarchical clustering (UPGMA). The clusters are coloured differently.

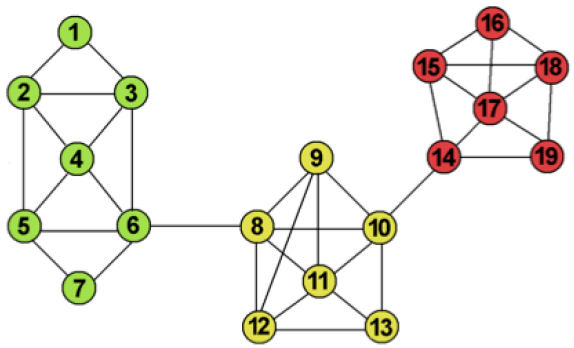
Markov Clustering algorithm, MCL, micans.org/mcl/

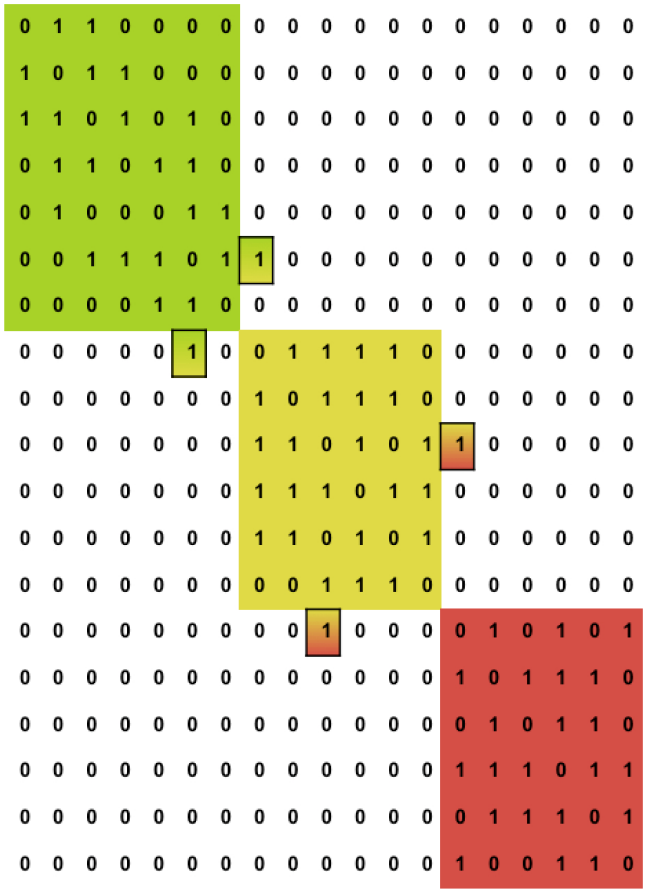
Unlike most clustering algorithms, the MCL does not require the number of expected clusters to be specified beforehand. The basic idea underlying the algorithm is that dense clusters correspond to regions with a larger number of paths.
ANALOGY: We take a random walk on the graph described by the similarity 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

The figure shows how to generate the input from a network.



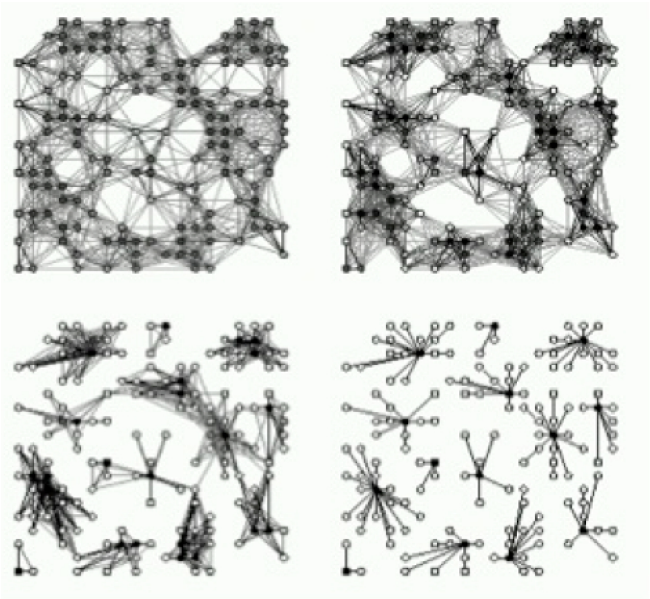


MCL Algorithm

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.



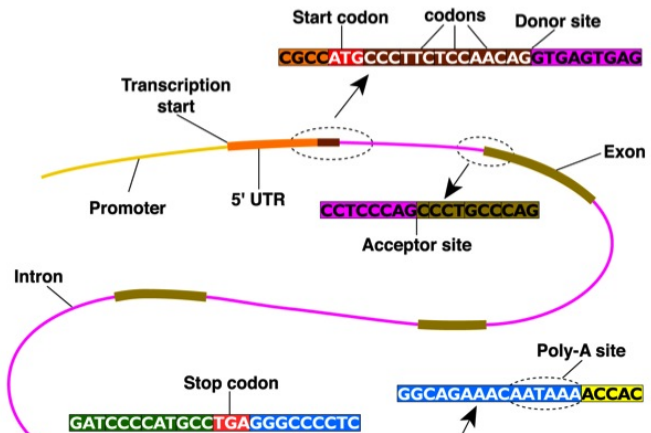
A: Progression from left to right and top to bottom

B: Biological applications: protein families analysis (tribeMCL)

Figure : Top: mcl progression; bottom: example of Tribemcl (www.ncbi.nlm.nih.gov/pubmed/11917018)

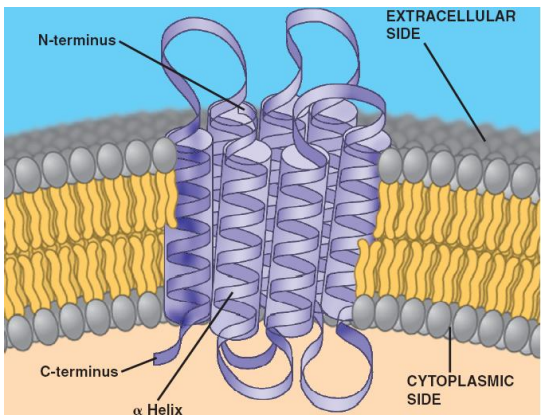
10 Applications of Hidden Markov models (HMM): recognition of a human gene

The gene information starts with the promoter, which is followed by a transcribed (i.e. RNA) but non-coding (i.e. not translated) region called 5' untranslated region (5' UTR). The initial exon contains the start codon which is usually ATG. There is an alternating series of introns and 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. The intron/exon and exon/intron boundaries are conserved short sequences and called the acceptor and donor sites. For all these different parts we need to know their probability of occurrence in a large database.



Applications of HMM: recognition of the protein structure

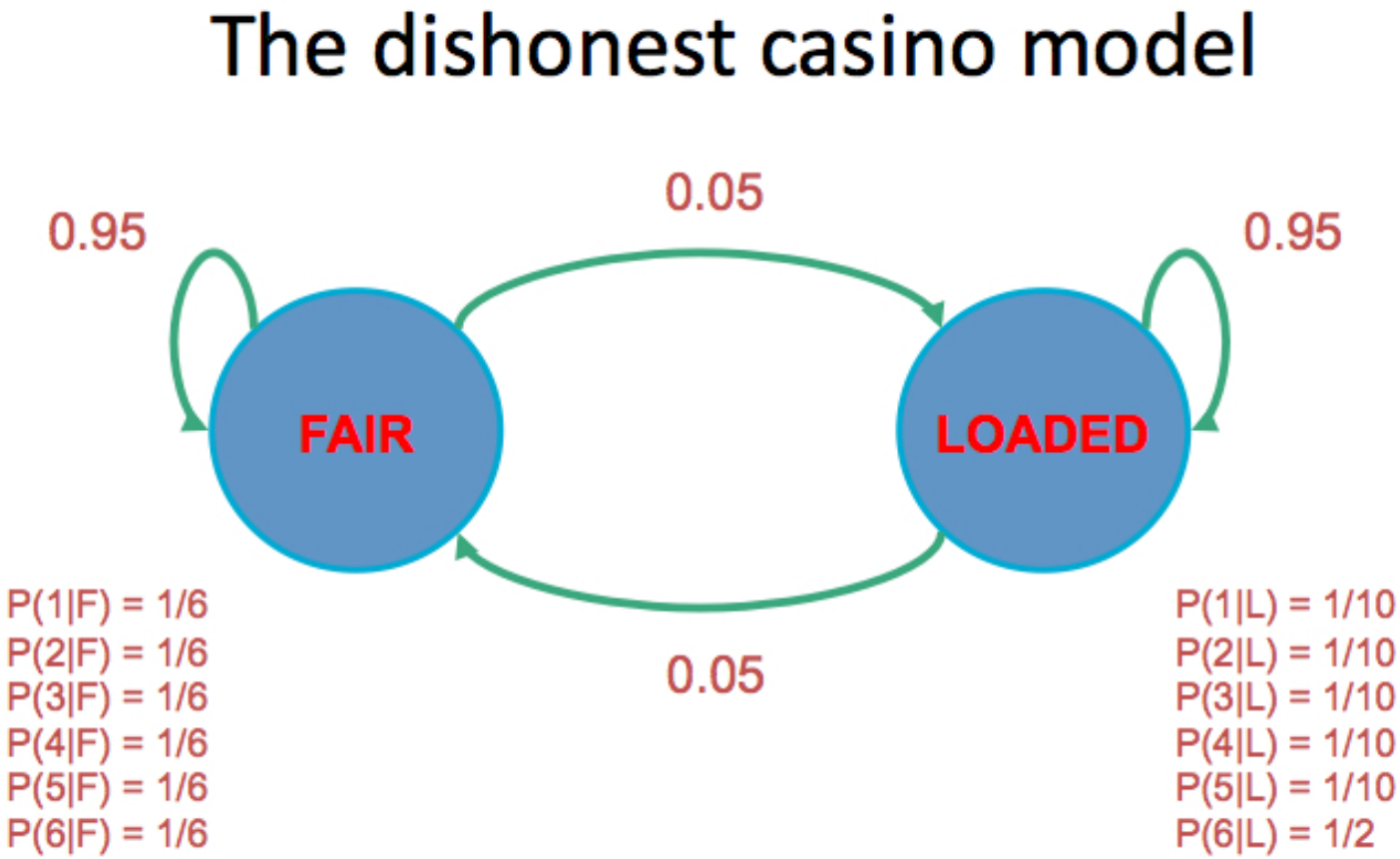
Membrane proteins that are important for cell import/export. We would like to predict the position in the amino acids with respect to the membrane. The prediction of gene parts and of the membrane protein topology (i.e. which parts are outside, inside and buried in the membrane) will require to train the model with a dataset of experimentally determined genes / transmembrane helices and to validate the model with another dataset. The figure below describes a 7 helix membrane protein forming a sort of a cylinder (porus) 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 = $b_1, b_2, , b_M$, set of states Q = 1, ..., K , and transition probabilities between any two states
a_{ij} = transition prob from state i to state j
$a_{i1} + + a_{iK} = 1$, for all states i = 1,K
Start probabilities a_{0i}
$a_{01} + + a_{0K}$ = 1
Emission probabilities within each state
$e_i(b) = P(x_i = b|\pi_i = k)$
$e_i(b_1) + + e_i(b_M)$ = 1, for all states i = 1,K
A Hidden Markov model is memoryless: $P(\pi_{t+1} = k|$ whatever happened so far) = $P(\pi_{t+1} = k|\pi_1, \pi_2, , \pi_t, x_1, x_2, , x_t)$ = $P(\pi_{t+1} = k|\pi_t)$ at each time step t, only matters the current state π_t.

Example of HMM model of using dice

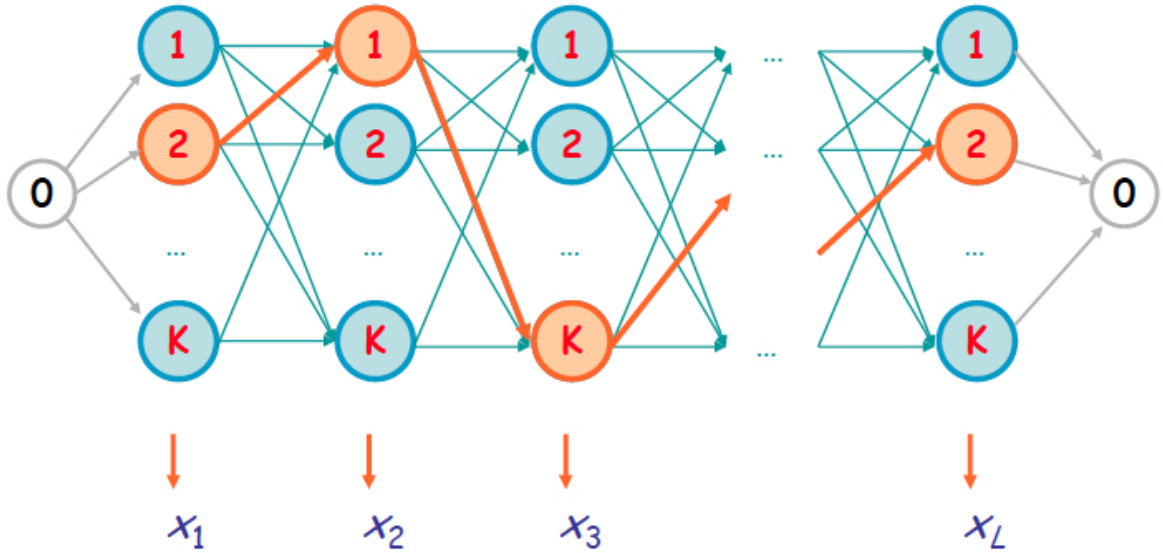


http://ai.stanford.edu/ serafim/

The dishonest casino: what is known, what we infer

- Known: The structure of the model
- The transition probabilities
- Hidden: What the casino did (ex FFFFFLLLLLLLFFFF)
- 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

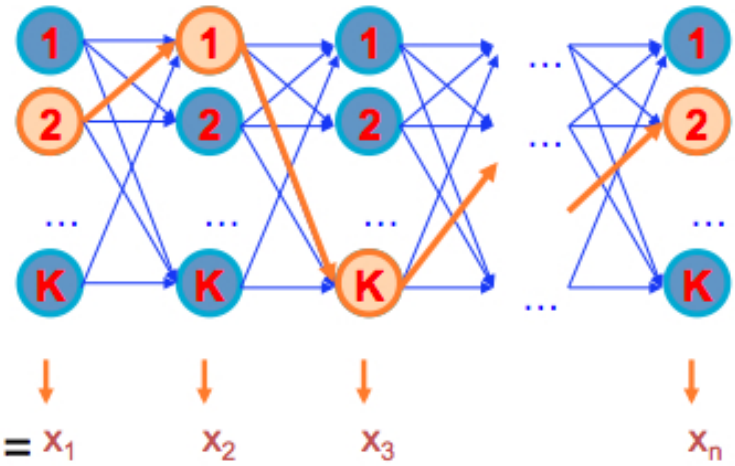


$$\Pr(x,\pi) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) \cdot a_{\pi_i \pi_{i+1}}$$

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

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

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



$P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots\ldots, \pi_N) =$

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

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

$a_{0\pi 1}\ a_{\pi 1 \pi 2} \ldots\ldots a_{\pi N-1 \pi N}\ e_{\pi 1}(x_1) \ldots\ldots e_{\pi N}(x_N)$

The three main questions on HMMs

1. Evaluation

   GIVEN a HMM M, and a sequence x,

   FIND Prob[x | M]

2. Decoding

   GIVEN a HMM M, and a sequence x,

   FIND the sequence π of states that maximizes P[x, π | M]

3. Learning

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

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

Evaluation: forward algorithm or the backwards algorithm; decoding: Viterbi; Learning: Baum Welch = forward-backward algorithm (not in this course).

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

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

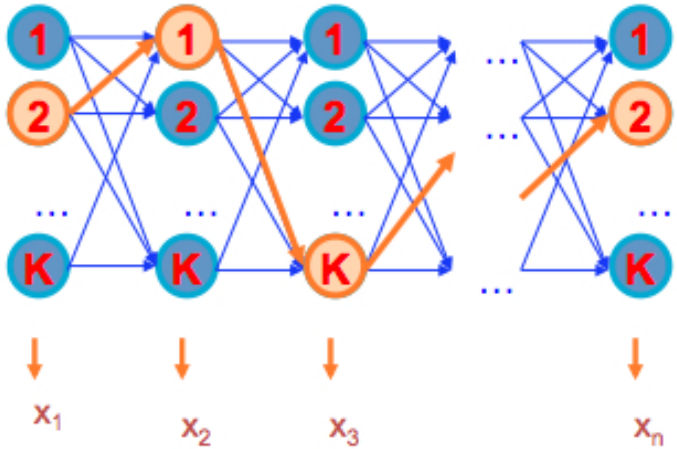
Similarly, P[x, π | M] and P[x, π] are the same

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

GIVEN $x = x_1 x_2 \ldots\ldots x_N$

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

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



We can use dynamic programming!

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

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

Decoding main idea

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

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

What is $V_k(i+1)$?

From definition,

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

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

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

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

$$e_l(x_{i+1}) \max_k a_{kl}\, V_k(i)$$

The Viterbi Algorithm

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

Initialization:

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

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

Iteration:

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

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

Termination:

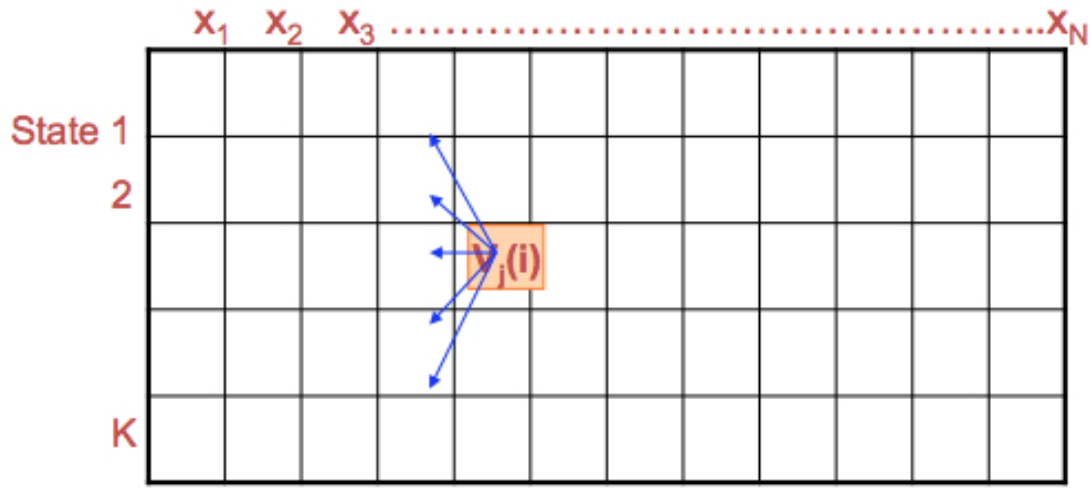
$P(x, \pi^*) = \max_k V_k(N)$

Traceback:

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

$\pi_{i-1}^* = Ptr_{\pi i}(i)$

Complexity of the Viterbi Algorithm



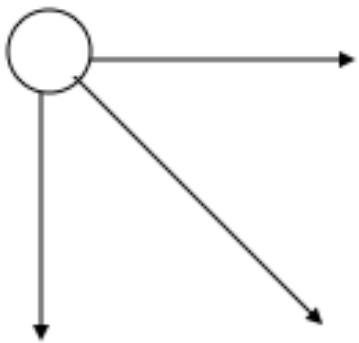
Similar to "aligning" a set of states to a sequence

Time:

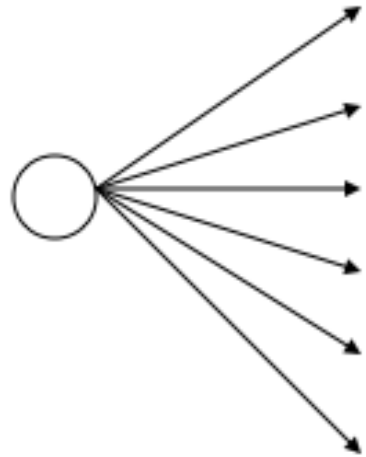
$O(K^2N)$

Space:

$O(KN)$



Valid directions in the *alignment problem.*

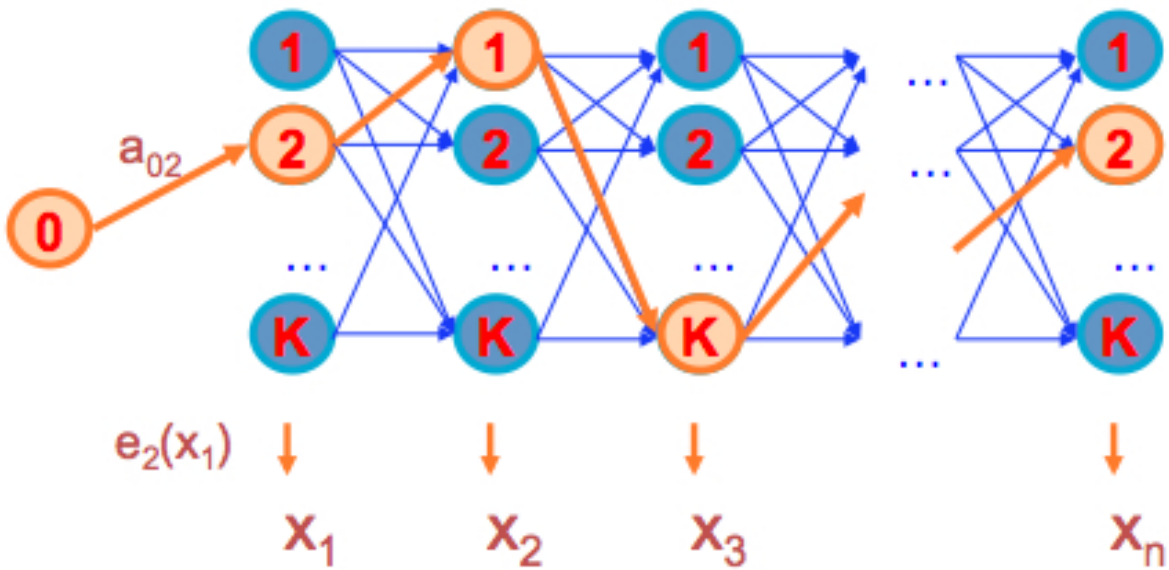


Valid directions in the *decoding problem.*

Generating a sequence by the model

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

1. Start at state π_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 π_2 according to prob $a_{\pi_1\pi_2}$
4. until emitting x_n



Evaluation

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

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

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

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

To avoid summing over an exponential number of paths π, define

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

The Forward Algorithm derivation

Define the forward probability:

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

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

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

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

The Forward Algorithm

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

Initialization:

$f_0(0) = 1$

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

Iteration:

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

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:

$V_0(0) = 1$

$V_k(0) = 0$, 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:

$f_0(0) = 1$

$f_k(0) = 0$, 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^{th} position, given x

We start by computing

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

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

The Backward Algorithm derivation

Define the backward probability:

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

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

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

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

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

The Backward Algorithm

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

Initialization:

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

Iteration:

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

Termination:

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

Complexity

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

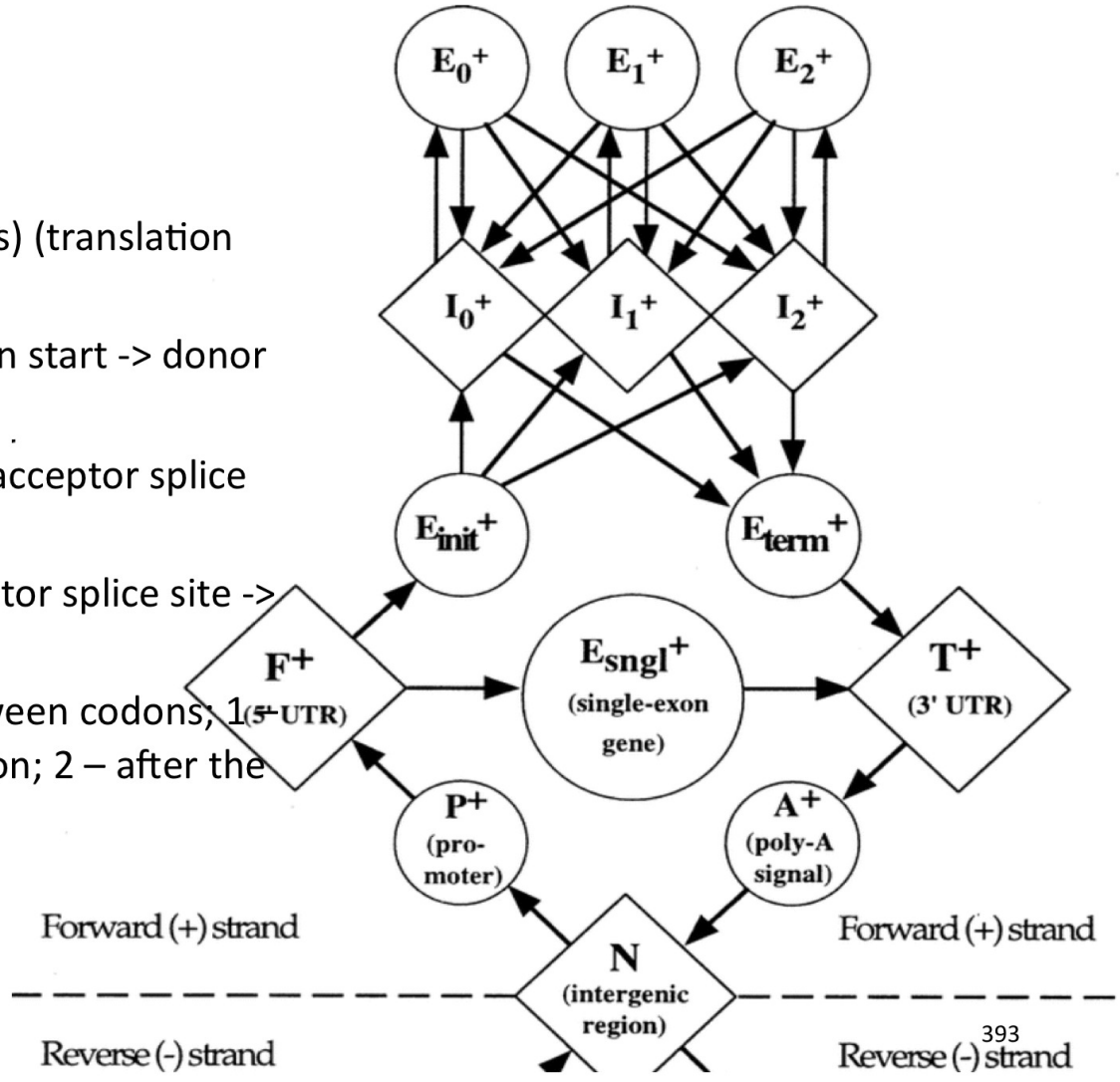
Viterbi: sum of logs

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

http://genes.mit.edu/GENSCAN.html

GenScan

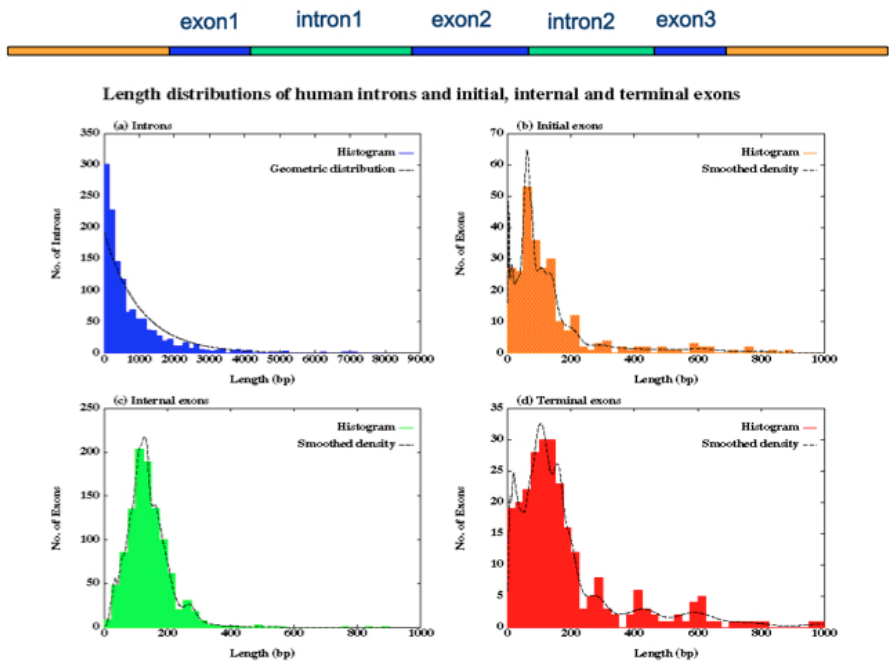
- N - intergenic region
- P - promoter
- F - 5’ untranslated region
- E_{sngl} – single exon (intronless) (translation start -> stop codon)
- E_{init} – initial exon (translation start -> donor splice site)
- E_k – phase k internal exon (acceptor splice site -> donor splice site)
- E_{term} – terminal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



393

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: ~ 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)

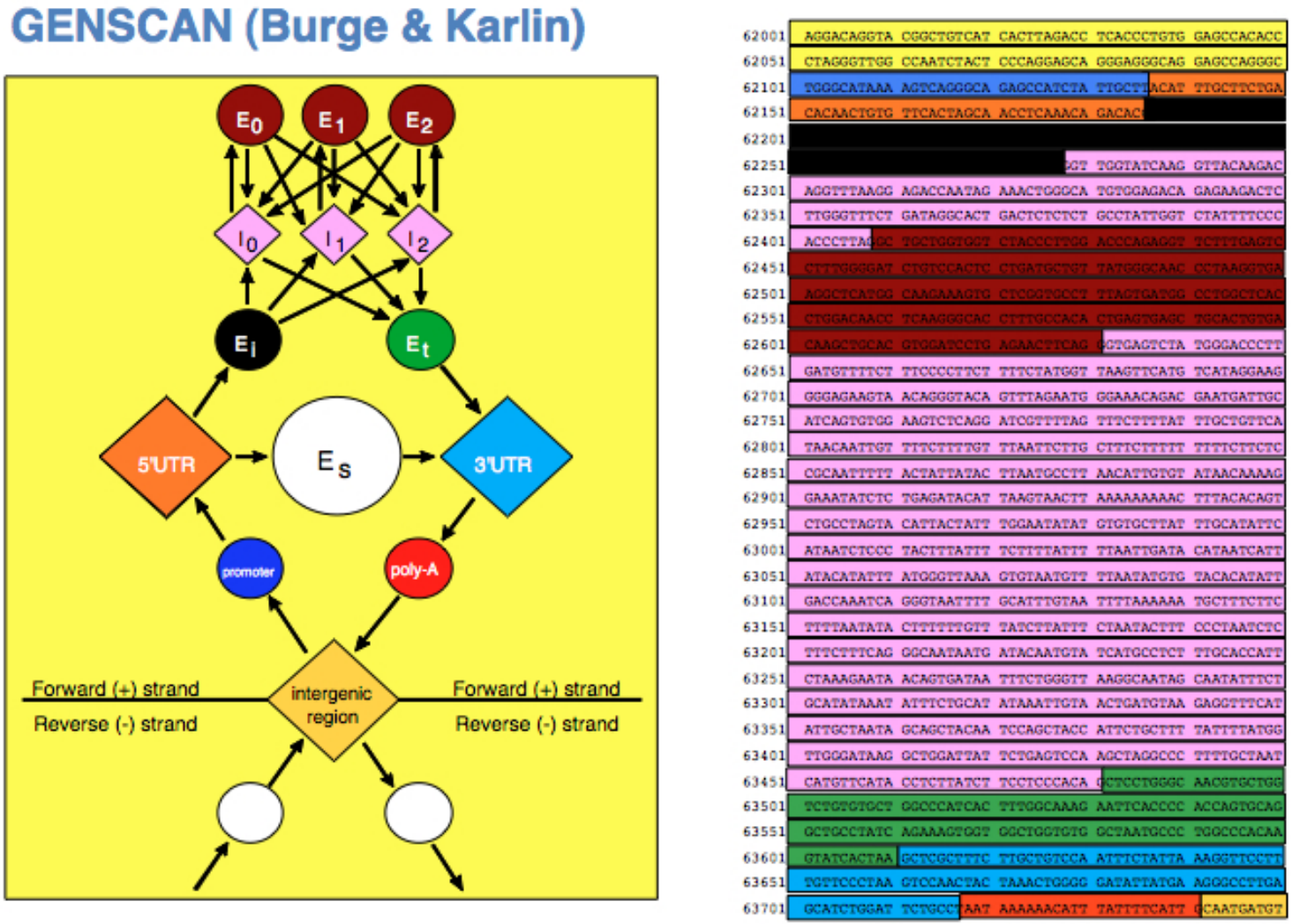


Figure : The model (left) and the output (right) of Genscan prediction of a genomic region; the result is a segmentation of a genome sequence, i.e. the colours map the HMM states with the predicted

Prediction of aminoacid segments included in membrane proteins

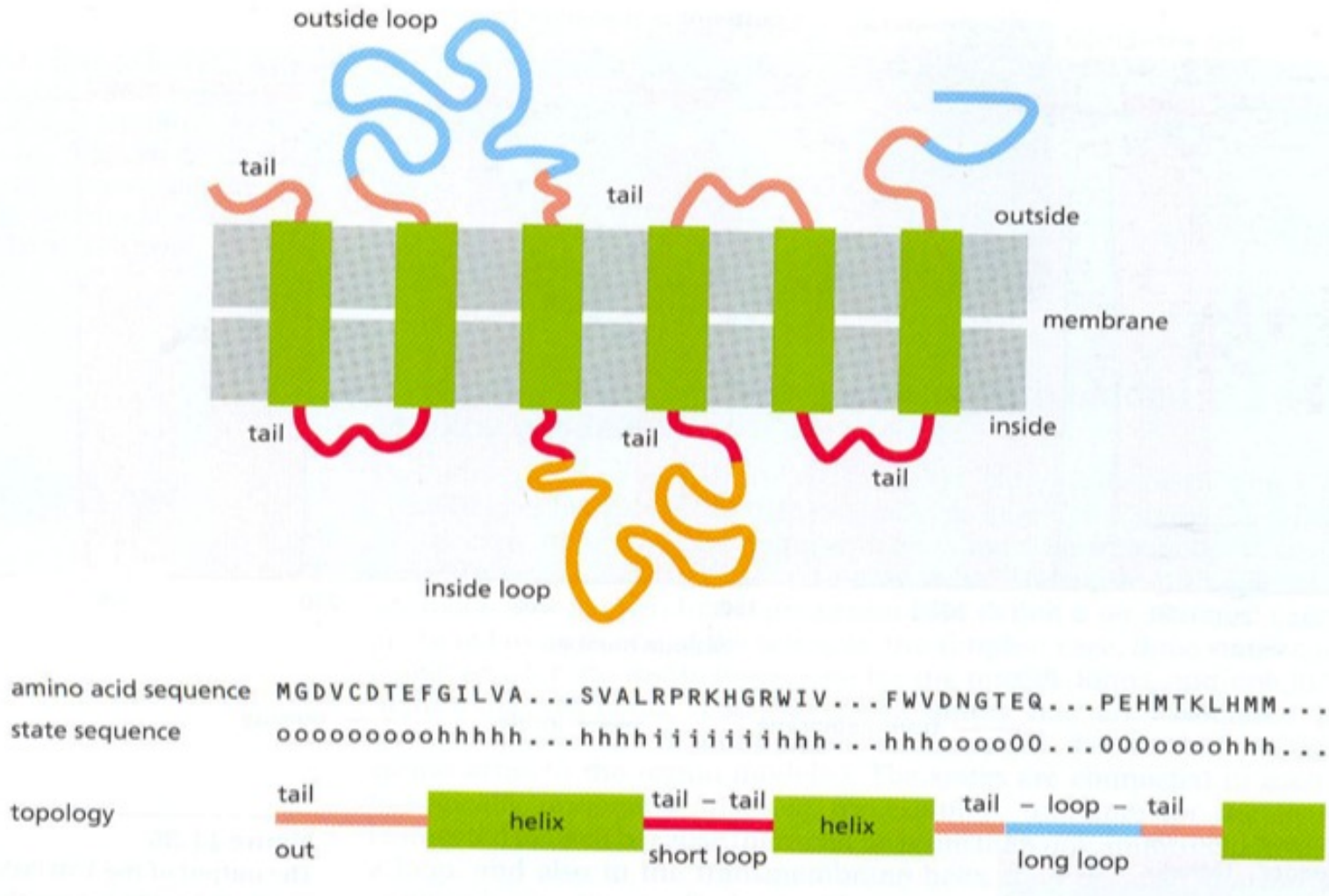


Figure : top: the 3D graph previous figure could be represented as a 2D graph; bottom, 3 state prediction: each amino acid could be in the membrane (h), outside the cell (o) or inside the cell (i)

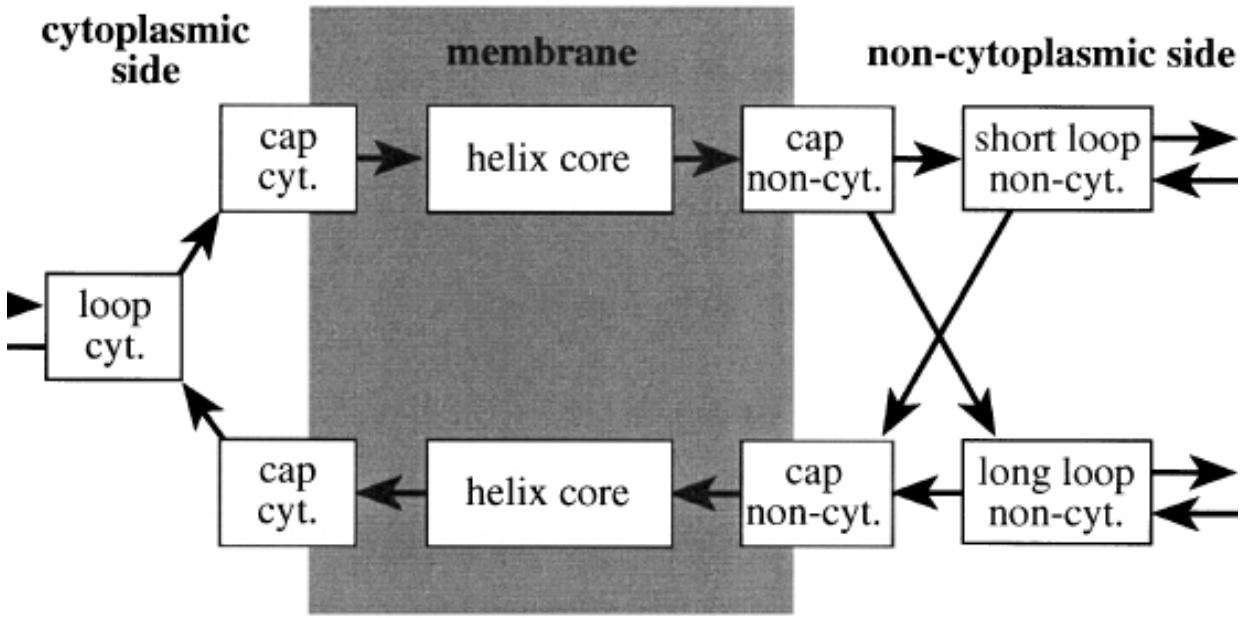
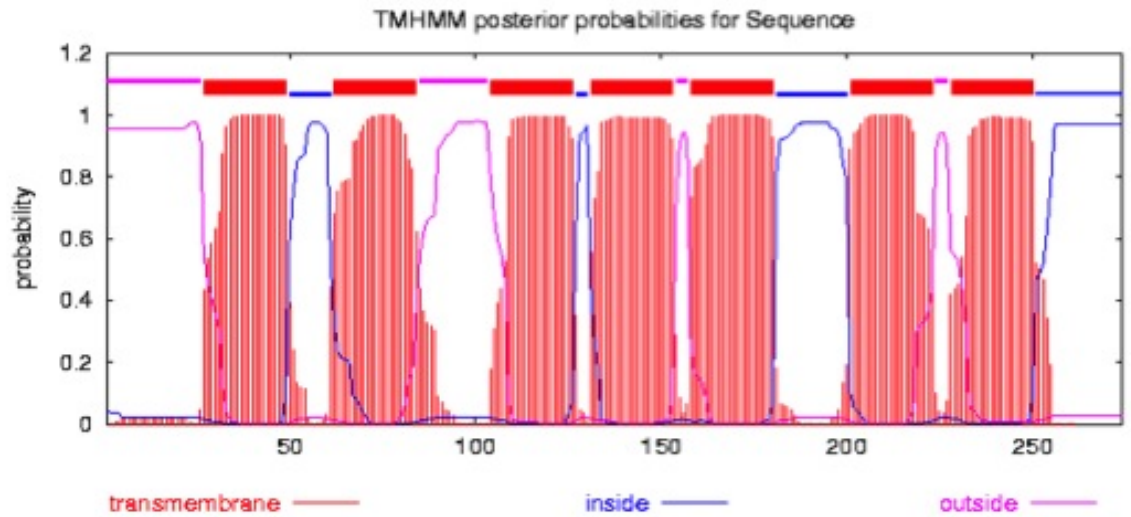


Figure : The THMM model: a three state prediction model (h,o,i) could be then refined adding more states, for example caps, i.e. the boundary between outside and membrane and inside and membrane. This refinement improves the prediction of the topology of the protein.

TMHMM http://www.cbs.dtu.dk/services/TMHMM/

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



Other important and related application: Use of HMM in sequence alignment (PFAM: http://pfam.xfam.org/)

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 $TP = PP \bigcap AP$
6. True Negative $TN = PN \bigcap AN$
7. False Negative $FN = PN \bigcap AP$
8. False Positive $FP = PP \bigcap AN$
9. Sensitivity: probability of correctly predicting a positive example Sn = TP/(TP + FN)
10. Specificity: probability of correctly predicting a negative example Sp = TN/(TN + FP) or
11. Probability that positive prediction is correct Sp = TP/(TP + FP).

11 Gibbs sampling: the string searching problem

Figure : Inserting a 15-bases motif with 4 mutations: why finding the motif is difficult? Reason to search for motifs: we know from microarray analysis that n genes are activated together so there may be a protein that binds somewhere before the start of each of them.

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 : The regulation of a gene could be very complex with several binding proteins (transcription factor) involved (left). Right: several genes are co-regulated (activated or repressed) by same protein that binds before the gene start (co-regulated genes could be identified with microarray).

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,...,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 (or you can also use all the t 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.

Non Mathematical introduction to Gibbs sampling

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

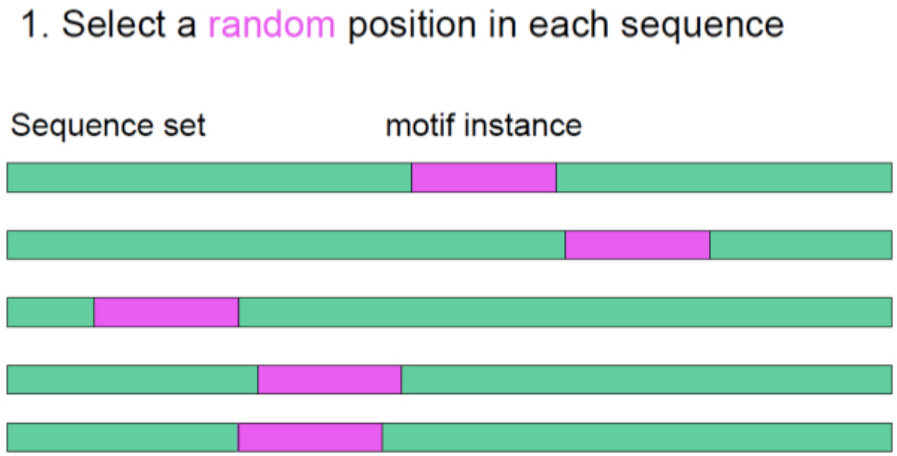
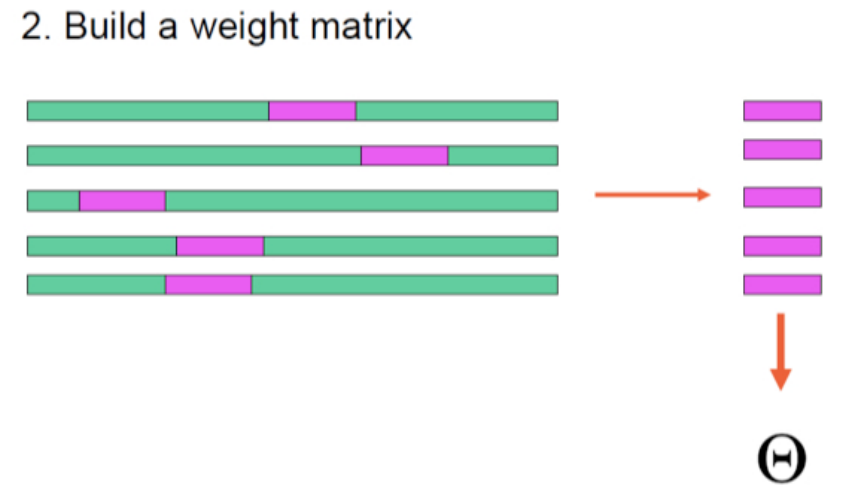
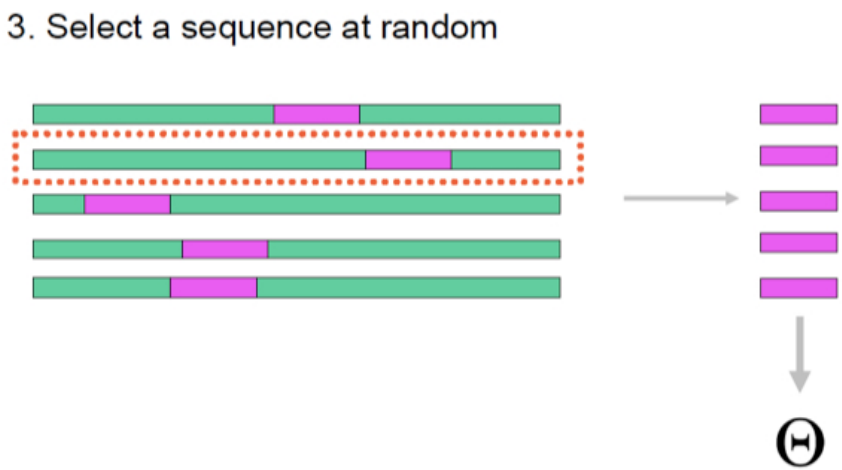


Figure : motifs in purple, the rest of the sequences in green; next figures: theta is the weight matrix i.e. the frequency of each base in the aligned set of motifs; red the best fitting motif; in y axis the likelihood of each motif with respect to the current weight matrix.

First Gibbs Sampling implementations: AlignACE (arep.med.harvard.edu/mrnadata/mrnasoft.html) and BioProspector (ai.stanford.edu/ xsliu/BioProspector/). See ccmbweb.ccv.brown.edu





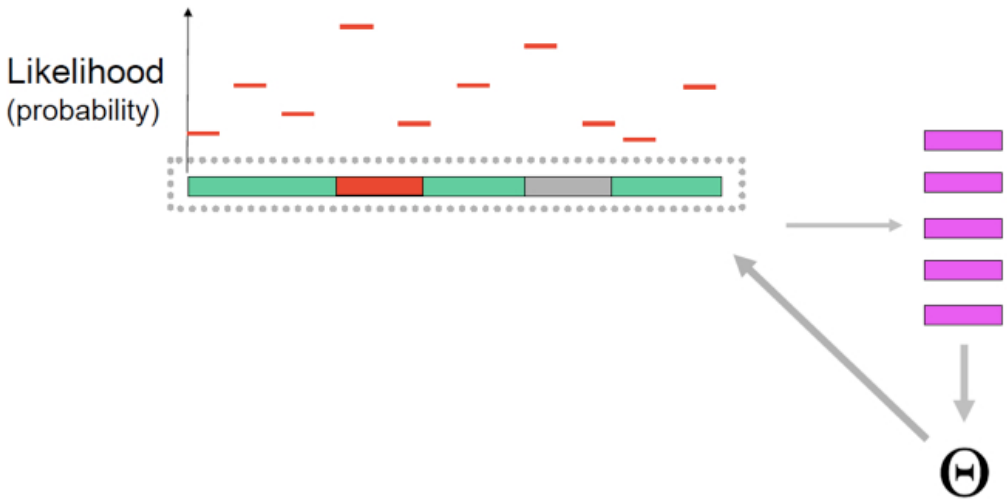
A weight matrix θ has one row for each symbol of the alphabet and one column for each position in the pattern. It is a position probability matrix computed from the frequency of each symbol in each position.

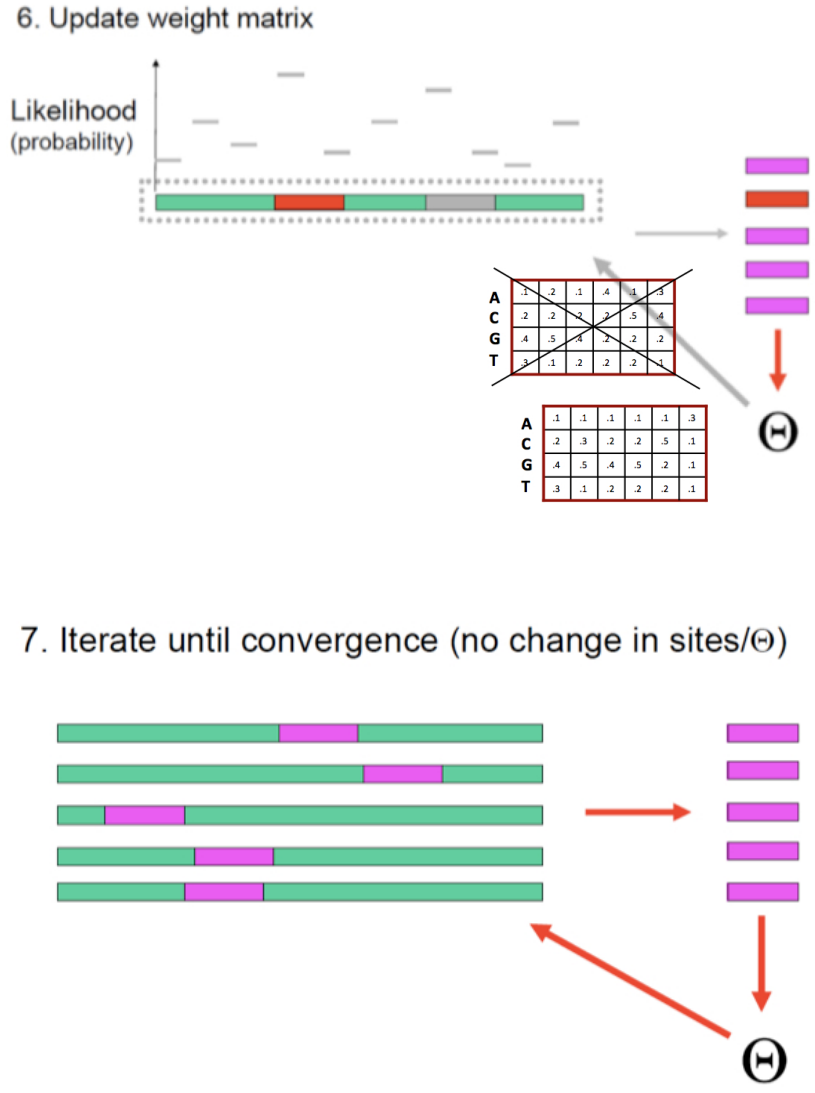
4. Score possible sites in seq using weight matrix

A	.1	.2	.1	.4	.1	.3
C	.2	.2	.2	.2	.5	.4
G	.4	.5	.4	.2	.2	.2
T	.3	.1	.2	.2	.2	.1

Score = 0.2 + 0.5 +0.2 + 0.2 + 0.2 + 0.1 = 1.4

5. Sample a new site proportional to likelihood





Doesnt do reinitializations in the middle to get out of local maxima. Doesnt optimize the width (you have to specify width explicitly).

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

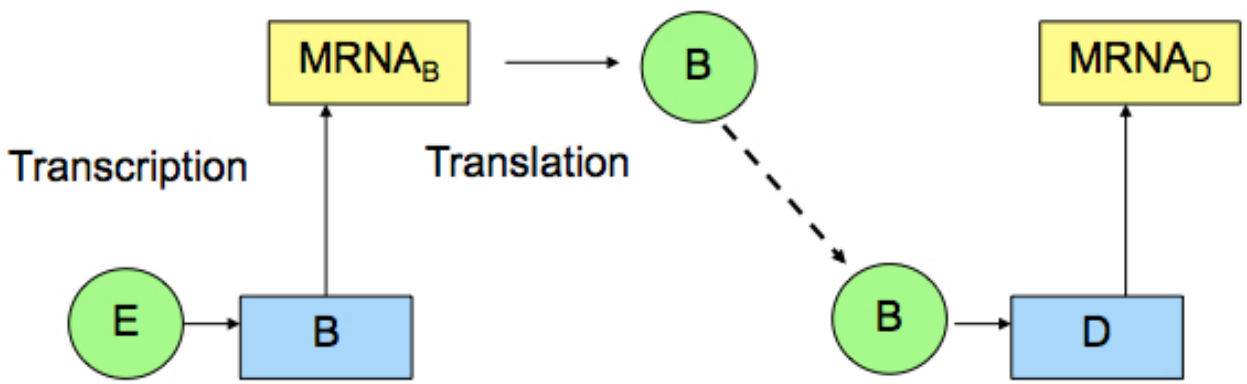


Figure : We have E only; B is a Primary Target of E; Production of $mRNA_B$ is enhanced by E; D is a Secondary Target of E; Production of $mRNA_D$ is enhanced by B; $mRNA_B$ and $mRNA_D$ quantified by microarray.

A genetic perturbation 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.

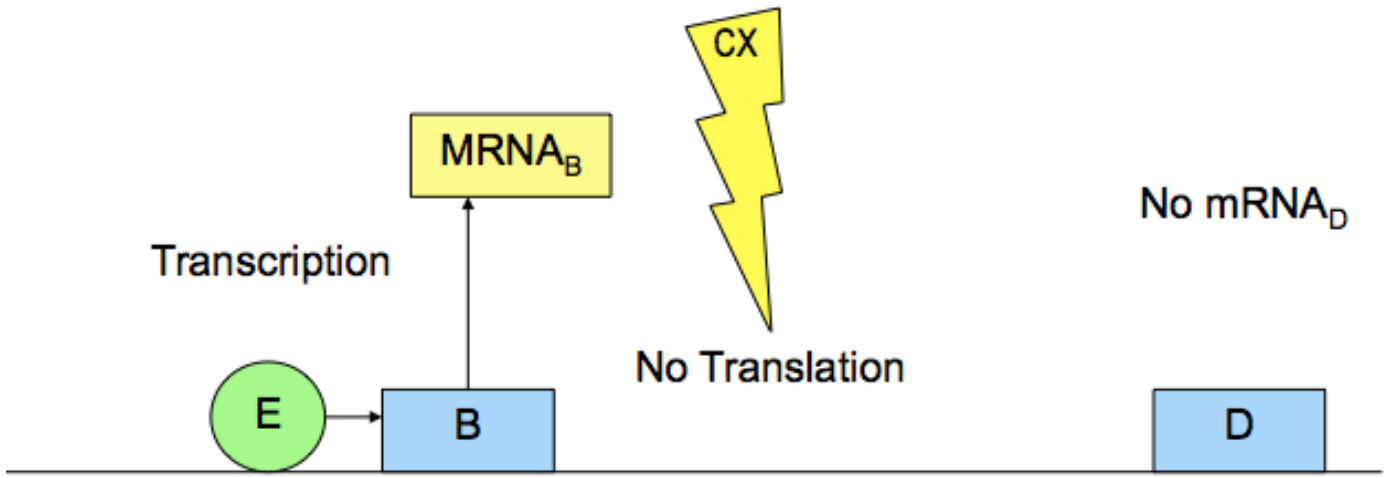


Figure : E and CX both present; B is a Primary Target; Production of RNA_B is enhanced by E; Production of RNA_D is decreased (prevented)

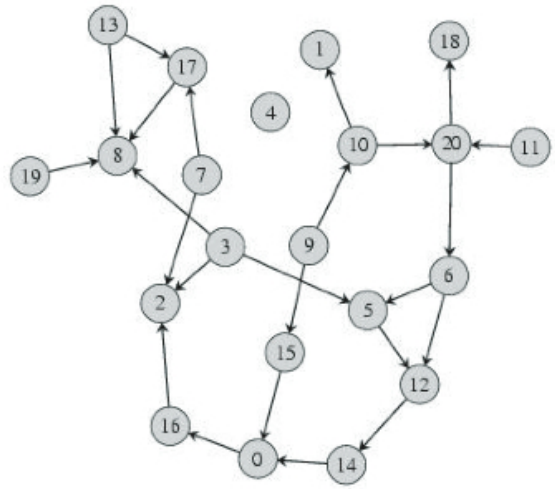
Network reconstruction: direct and indirect effects

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
18:	
19:	
20:	

(c)

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

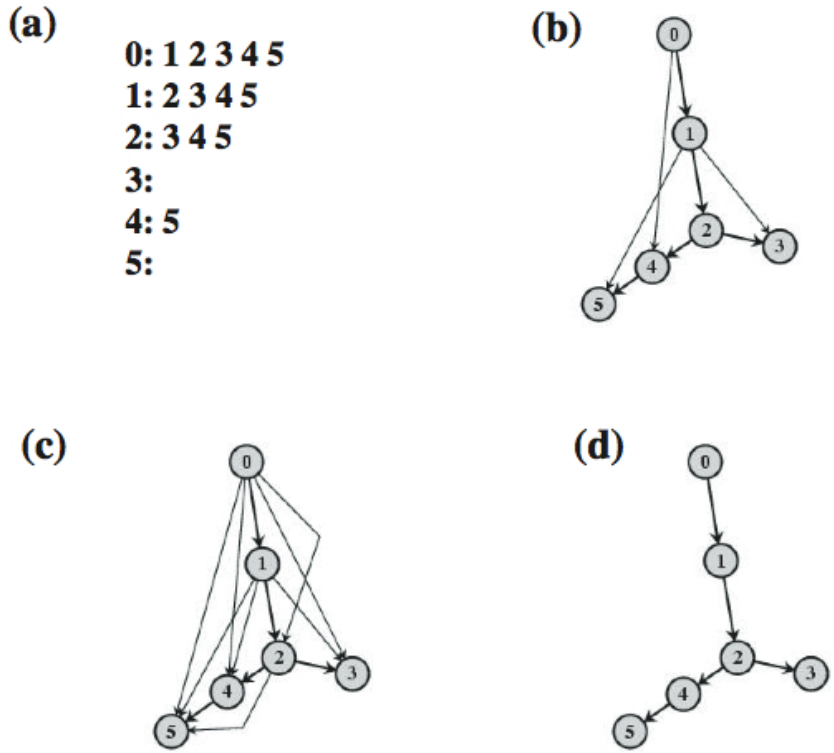


Figure : The figure illustrates three graphs (Figs. B,C,D) with the same accessibility list Acc (Fig. A). There is one graph (Fig. D) that has Acc as its accessibility list and is simpler than all other graphs, in the sense that it has fewer edges. Lets call Gpars the most parsimonious network compatible 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 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.
Acc(i) is the set of nodes that can be reached from node i by following all paths of directed edges leaving i. 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.

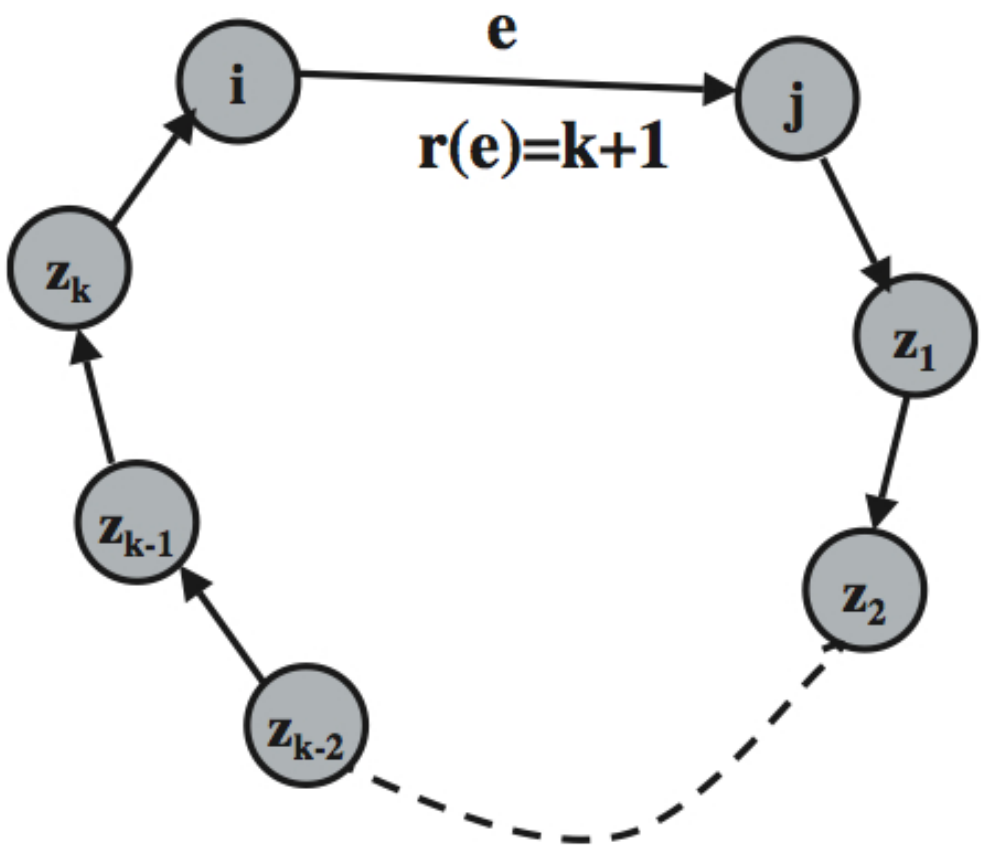
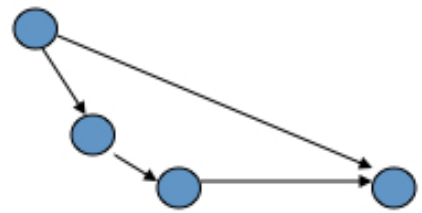


Figure : A shortcut is an edge connecting two nodes, i and j that are also connected via a longer path of edges. The shortcut e is a shortcut range k+1. That is, when eliminating e, I and j are still connected by a path of length k+1.

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 Gpars 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: = ∞.

Theorem

Let Acc(G) be the accessibility list of an acyclic directed graph, Gpars its most parsimonious graph, and V(Gpars) the set of all nodes of Gpars . We have the following equation (1):
$\forall i \in V\left(G_{pars}\right) \ldots Adj\left(i\right) = Acc\left(i\right) \setminus \cup_{j \in Acc(i)} Acc\left(j\right)$
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

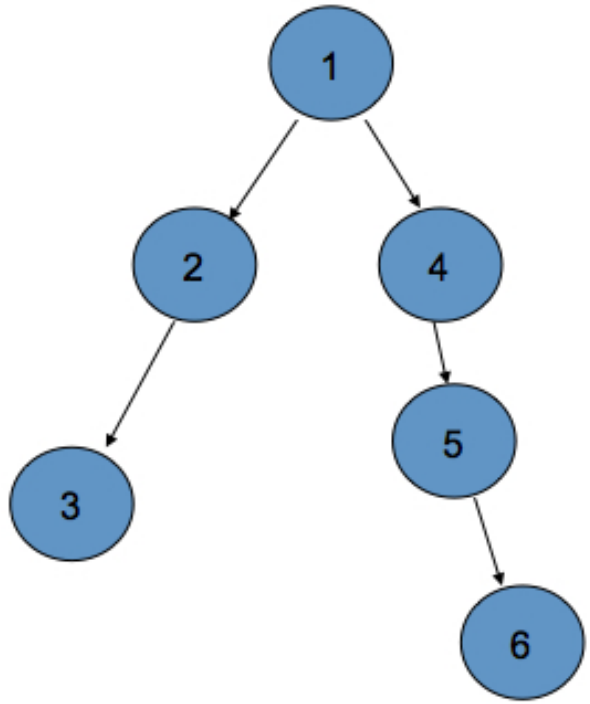


Figure : $Adj(1) = Acc(1) -$
$(Acc(2) + Acc(3) + Acc(4) + Acc(5) + Acc(6)) = (2, 3, 4, 5, 6)$ -
$(3 \cup (5, 6) \cup 6) = (2, 4)$

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

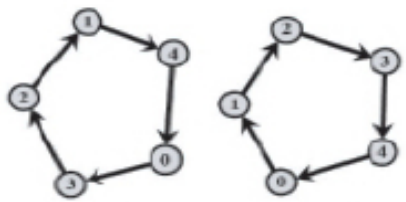
Two different cycles have the same accessibility list

Perturbations of any gene in the cycle influences the activity of all

other genes in the same cycle

Cant 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

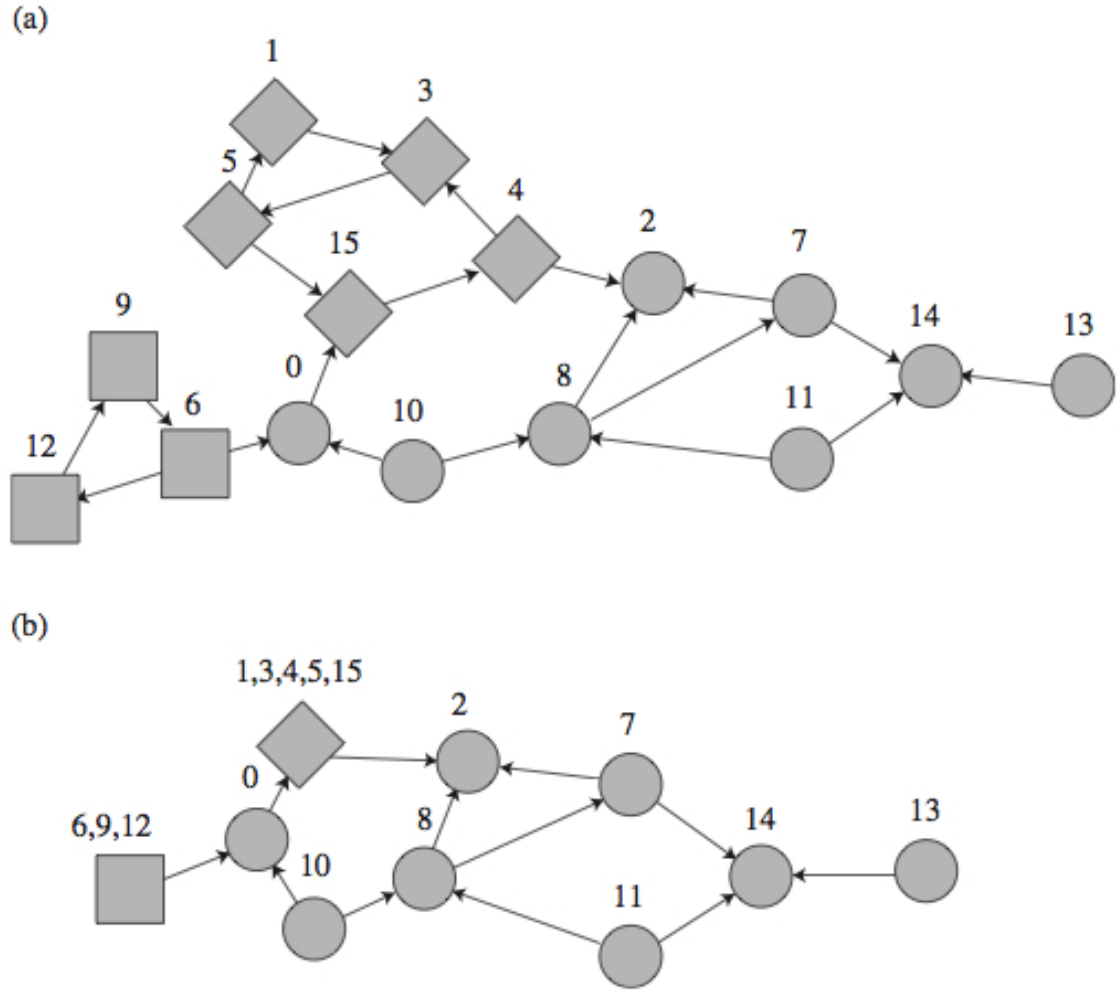


Figure : Basic idea: Shrink each cycles (strongly connected components) into one node and apply the algorithm of step 1. A graph after shrinking all the cycles into nodes is called a condensation graph

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 $O(nka)$.
- For practical matters, large scale experimental gene perturbations in the yeast Saccharomyces cerevisiae ($n = 6300$) suggests that $k < 50$, $a < 1$ and thus that $nka < n^2$ in that case.

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   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 ∈Acc(i)
9           if Acc(j)=∅
10              declare j as visited.
11          else
12              call PRUNE_ACC(j)
13          end if

14      for all nodes j ∈ Acc(i)
15          for all nodes k ∈ Adj(j)
16              if k ∈Acc(i)
17                  delete k from Adj(i)
18              end if
19  declare node i as visited
20  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 no *i* the adjacency list *Adj(i)* is initialized as equal to the accessibility list.

The algorithm will delete elemen from this *Adj(i)* until the adjacen 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 ∈Acc(i)
9           if Acc(j)=∅
10              declare j as visited.
11          else
12              call PRUNE_ACC(j)
13          end if

14      for all nodes j ∈ Acc(i)
15          for all nodes k ∈ Adj(j)
16              if k ∈Acc(i)
17                  delete k from Adj(i)
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 ∈Acc(i)
9             if Acc(j)=∅
10                declare j as visited.
11            else
12                call PRUNE_ACC(j)
13            end if

14        for all nodes j ∈ Acc(i)
15            for all nodes k ∈ Adj(j)
16                if k ∈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     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 ∈Acc(i)
9             if Acc(j)=∅
10                declare j as visited.
11            else
12                call PRUNE_ACC(j)
13            end if

14        for all nodes j ∈ Acc(i)
15            for all nodes k ∈ Adj(j)
16                if k ∈Acc(i)
17                    delete k from Adj(i)
18                end if
19    declare node i as visited
20    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)* =∅, then *j* is declared as visited.

Because its accessibility list is empty, its adjacency list must be empty as well (*Adj(i)* ⊆*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 ∈Acc(i)
9             if Acc(j)=∅
10                declare j as visited.
11            else
12                call PRUNE_ACC(j)
13            end if

14        for all nodes j ∈ Acc(i)
15            for all nodes k ∈ Adj(j)
16                if k ∈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 ∈Acc(i)
9           if Acc(j)=∅
10              declare j as visited.
11          else
12              call PRUNE_ACC(j)
13          end if

14      for all nodes j ∈ Acc(i)
15          for all nodes k ∈ Adj(j)
16              if k ∈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 ∈Acc(i)
9                 if Acc(j)=∅
10                      declare j as visited.
11                else
12                      call PRUNE_ACC(j)
13                end if

14          for all nodes j ∈ Acc(i)
15                for all nodes k ∈ Adj(j)
16                      if k ∈Acc(i)
17                          delete k  from Adj(i)
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.

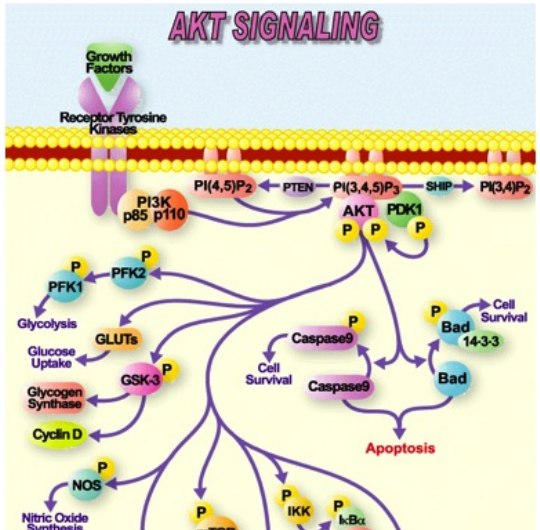
Notice also that if a node j with $Acc(j) = \in$ is encountered, the loop in line 15 is not executed.

```
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 ∈Acc(i)
9            if Acc(j)=∅
10               declare j as visited.
11           else
12               call PRUNE_ACC(j)
13           end if

14       for all nodes j ∈ Acc(i)
15           for all nodes k ∈ Adj(j)
16               if k ∈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∈Acc(i)
6                 if i∈Acc(j)
7                     component[j]=x
8                 end if
9         end if

10    for all nodes i of G*
11        Acc_G*(i)=∅
12    for all nodes i of G
13        for all nodes j ∈ Acc(i)
14            if component[i] ≠ component[j]
15                if component[j]∉Acc_G*(component[i])
16                    add component[j] to Acc_G*(component[i])
17                end if
18            end if
```

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 networks. Using deterministic methods, it is not possible to capture emergent phenomena that arise from inherent randomness. The below figure shows an example of biochemical pathway (you can find it in Kegg database) that could be simulated using Gillespie.



Gillespie algorithm

Consider a system of N molecular species $S_1, , S_N$ interacting through M elemental chemical reactions $R_1, , R_M$.
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 $X(t) = X_1(t), , X_N(t)$, where $Xi(t)$ 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 ith 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 Gillespies stochastic simulation algorithm (SSA)

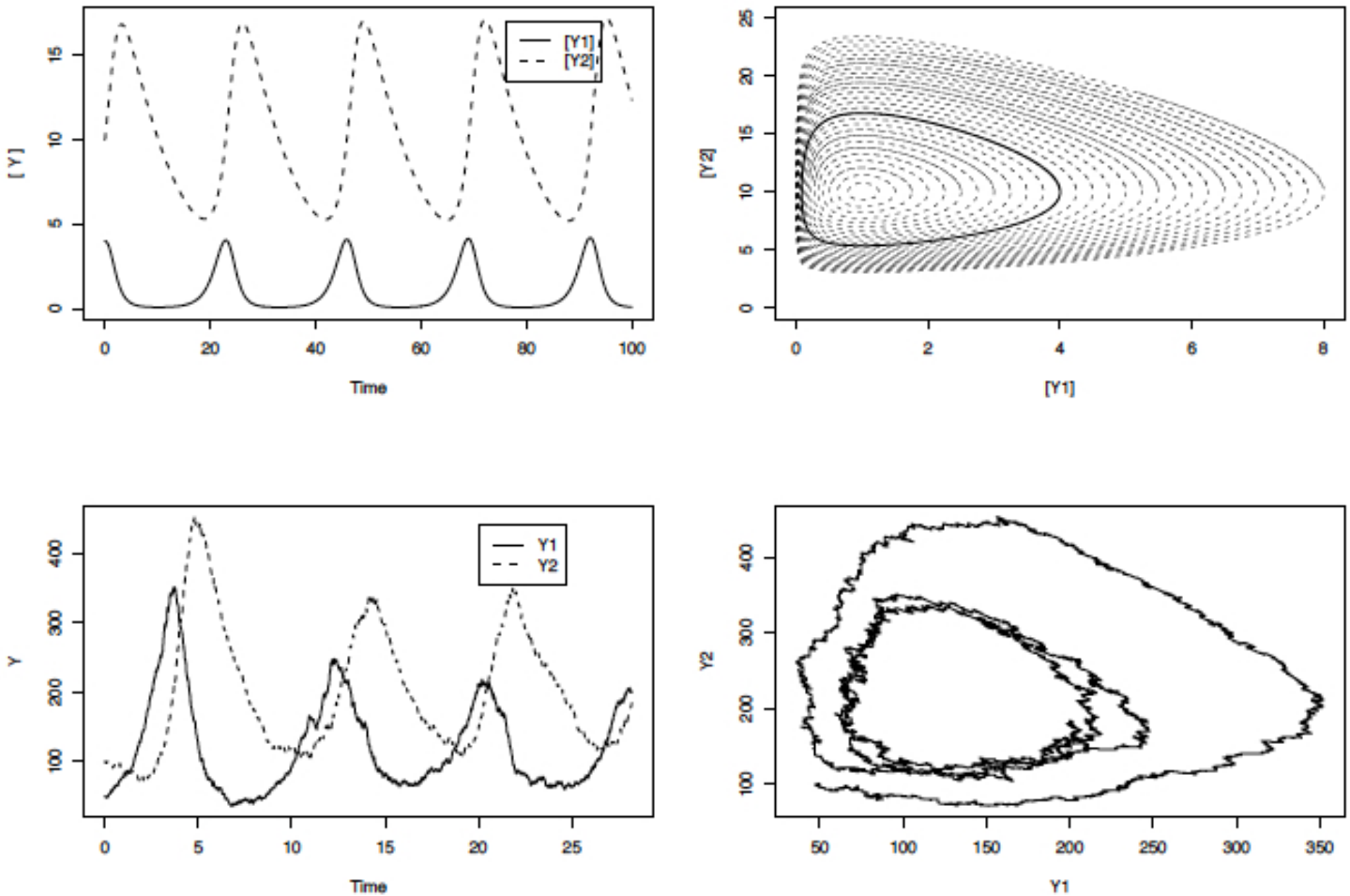
- The Gillespie algorithm is widely used to simulate the behavior of a system of chemical reactions in a well stirred container
- The key aspects of the algorithm is the drawing of two random numbers at each time step, one to determine after how much time the next reaction will take place, the second 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 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 numbers, set time $t = 0$.
2. Calculate the propensity function a_i for each reaction, and the total propensity according to equation $a_0(x) \equiv \sum_{j=1}^{M} a_j(x)$, j = 1,...,M.
3. Generate two uniformly distributed random numbers r_1 and r_2 from the range (0, 1).
4. Compute the time τ to the next reaction using equation $\tau = \frac{1}{a_{0(x)}} ln\left(\frac{1}{r_1}\right)$.
5. Decide which reaction R_μ occurs at the new time using equation $r_2 > \sum_{k=1}^{\mu-1} a_k \ldots and \ldots r_2 < \frac{1}{a_0} \sum_{k=1}^{\mu} a_k$.
6. Update the state vector v (molecules quantity) 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}.

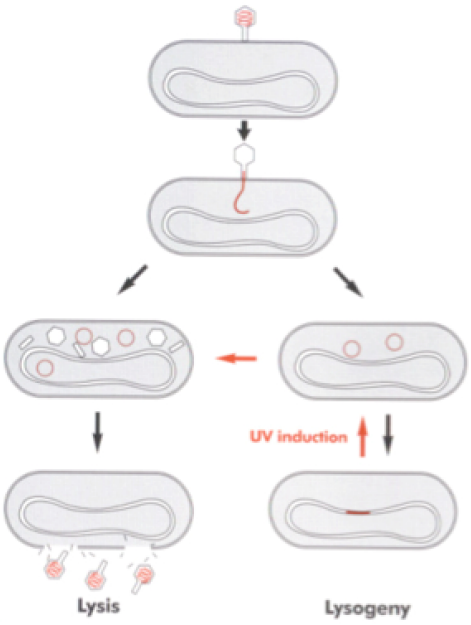
In each step, the SSA starts from a current state $x(t) = x$ and asks two questions: When will the next reaction occur? We denote this time interval by t . When the next reaction occurs, which reaction will it be? We denote the chosen reaction by the index j. To answer the above questions, one needs to study the joint probability density function $p(\tau, j \mid x, t)$ 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
$p(\tau, j \mid x, t) = a_j(x)\, exp(-a_0(x)\tau)$,
where $a_0(x) \equiv \sum_{j=1}^{M} a_j(x)$. It implies that the time t 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 τ 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.

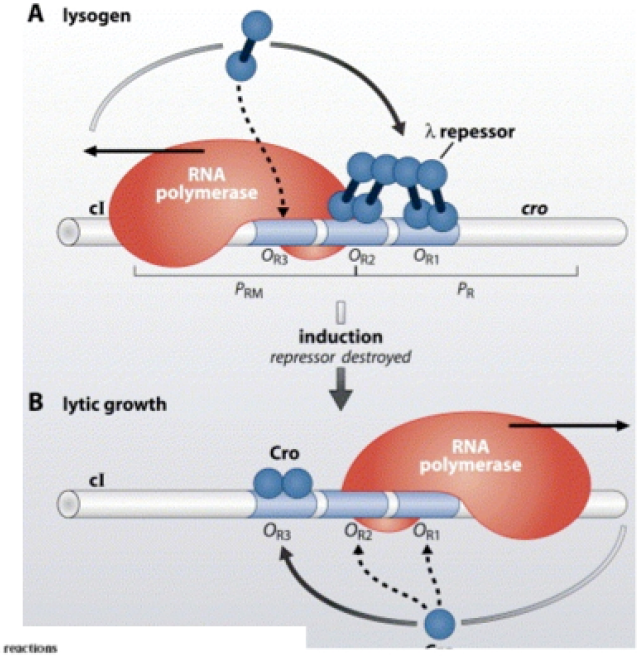
Example: ODE (top) versus Gillespie (bottom)

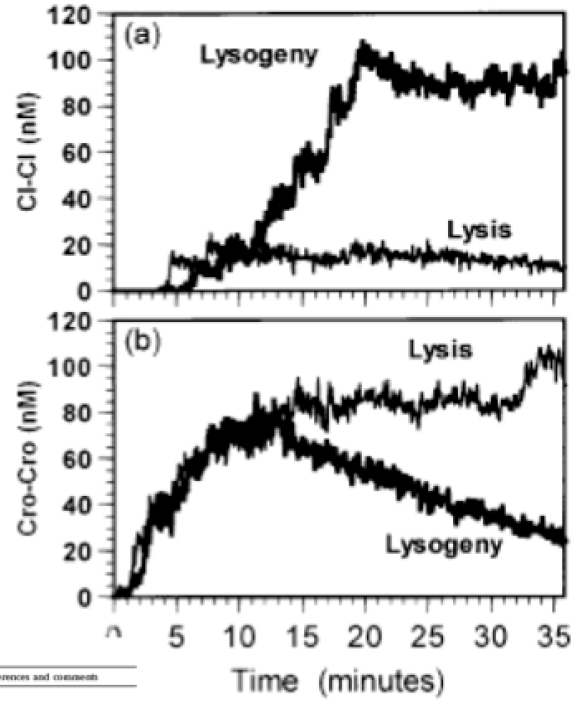


Examples of software: Copasi, www.copasi.org; stochkit, stochkit.sourceforge.net

Example of Gillespie algorithm







Parameters for transcription and translation reactions

Reaction/event	Parameter	References and comments
Transcription reactions		
RNAP·$DNA_o \xrightarrow{k_{2i}}$ RNAP·DNA_{o+1}	k_{2i} = 30 nt sec^{-1}	Selected as an average rate. Measured elo rates vary widely, depending on DNA s and cell state (Gotta *et al.* 1991; Kenn Riezman 1977; Kornberg and Bake Vogel and Jensen 1994)
RNAP·$DNA_{tad1,0} \xrightarrow{k_{2i}}$ RNAP·$DNA_{tad1,0+1}$	k_{2i} = 5 nt sec^{-1}	
RNAP·$DNA_{tad1,0}$ + N $\underset{k_{2i}}{\overset{k_{2i}}{\rightleftharpoons}}$ RNAP·N·$DNA_{tad1,0+1}$	k_{2i} = 0.145 (nM sec)$^{-1}$ k_{2i} = 0.1 sec^{-1}	Selected to produce termination and anti tion consistent with Li *et al.* (1992) and *et al.* (1988)
RNAP·N·$DNA_{tad1,0} \xrightarrow{k_{2i}}$ RNAP·N·$DNA_{tad1,0+1}$	k_{2i} = 30 nt sec^{-1}	
RNAP·$DNA_{T1} \xrightarrow{k_{2i}}$ RNAP·DNA_{T1+1}	k_{2i} = 15 nt sec^{-1}	Selected to yield 50% termination at *N* (Dambly-Chaudiere *et al.* 1983; Friedr Gottesman 1983)
RNAP·$DNA_{T1} \xrightarrow{k_{2i}}$ RNAP + DNA_{T1}	k_{2i} = 15 sec^{-1}	
RNAP·N·$DNA_{T1} \xrightarrow{k_{2i}}$ RNAP·N·DNA_{T1+1}	k_{2i} = 30 nt sec^{-1}	Assumption that antiterminated RNAP pass nator freely
RNAP·$DNA_{T1i} \xrightarrow{k_{2i}}$ RNAP·DNA_{T1i+1}	k_{2i} = 5 nt sec^{-1}	Selected to yield 80% termination at *N* =
RNAP·$DNA_{T1i} \xrightarrow{k_{2i}}$ RNAP + DNA_{T1i}	k_{2i} = 25 sec^{-1}	Selected to yield 80% termination at *N* =
RNAP·N·$DNA_{T1i} \xrightarrow{k_{2i}}$ RNAP·N·DNA_{T1i+1}	k_{2i} = 30 nt sec^{-1}	Assumption: antiterminated RNAP passes tor freely
Translation reactions		
Ribosome + $RNA_{RBS} \xrightarrow{k_{2i}}$ Ribosome·RNA_{RBS}	k_{2i} = 0.002 (nM sec)$^{-1}$	(Kennell and Riezman 1977; Sorensen ersen 1991)
Ribosome + $RNA_o \xrightarrow{k_{2i}}$ Ribosome·RNA_{o+1}	k_{2i} = 100 nt sec^{-1}	(Adhya and Gottesman 1982; Kennell a man 1977; Sorensen and Pedersen 19
RNase + $RNA_{RBS} \xrightarrow{k_{2i}}$ RNase	k_{2i}·RNase = 0.2 sec^{-1}	Adjusted to get an average of 10 proteins script
Average number of proteins per transcript		

Parameters for housekeeping and nongenetic reactions

Reaction/event	Parameter	References and comments
Housekeeping reactions		
Available RNAP	RNAP = 30 nM	
Available ribosomes	Ribosomes = 500 nM	
Cell volume (δ = (1 + k_v *t*) × 10^{-15} liters	k_v = 4.76 × 10^{-16} liters sec^{-1}	McClure (1980, 1983) To double initial cell volume of 10^{-15} liters in 35 min
Nongenetic reactions*		
CI $\xrightarrow{k_i}$ ( )	k_i = 0.0007 sec^{-1}	Selected to yield a CI/CI_2 life time of approxi mately 40 min (Reinitz and Vaisnys 1990) in the concentration range between 20 and 100 nM
2·CI $\underset{k_i}{\overset{k_i}{\rightleftharpoons}}$ CI_2	k_i = 0.05 nM^{-1} sec^{-1} k_i = 0.5 sec^{-1}	Ross *et al.* (1994); Shea and Ackers (1985)
Cro $\xrightarrow{k_i}$ ( )	k_i = 0.0025 sec^{-1}	Selected to match Cro/Cro_2 lifetime of ap proximately 30 min (Reinitz and Vaisnys 1990) in the concentration range between 20 and 100 nM
2·Cro $\underset{k_i}{\overset{k_i}{\rightleftharpoons}}$ Cro_2	k_i = 0.05 nM^{-1} sec^{-1} k_i = 0.5 sec^{-1}	Reinitz and Vaisnys (1990); Sauer (1979)
N $\xrightarrow{k_i}$ ( )	k_i = 0.00231 sec^{-1}	Gottesman and Gottesman (1981)
P1 concentration*	*P1* = 35 nM	Adjusted to match the % lysogeny vs. API data (Kourilsky 1973)
CII + *P1* $\underset{k_i}{\overset{k_i}{\rightleftharpoons}}$ *P1*·CII	k_i = 0.01 nM^{-1} sec^{-1}	Selected to match CII half life in Gottesman and Gottesman (1981)
P1·CII $\xrightarrow{k_i}$ *P1*	k_i = 0.01 sec^{-1} k_i = 0.002 sec^{-1}	
CIII + *P1* $\underset{k_i}{\overset{k_i}{\rightleftharpoons}}$ *P1*·CIII	k_i = 0.01 nM^{-1} sec^{-1}	Selected to match CIII protection of CII deg radation (Hoyt *et al.* 1982; Rattray *et al.* 1984) and CII half life Kobrtzer *et al.* (1991a,b)
P1·CIII $\xrightarrow{k_i}$ *P1*	k_i = 0.001 sec^{-1} k_i = 0.0001 sec^{-1}	
P2 concentration	*P2* = 140 nM	
CII + *P2* $\underset{k_i}{\overset{k_i}{\rightleftharpoons}}$ *P2*·CII	k_i = 0.00025 nM^{-1} sec^{-1}	Selected to match CII half life in Gottesman and Gottesman (1981)
P2·CII $\xrightarrow{k_i}$ *P2*	k_i = 0.065 sec^{-1}	
	k_i = 0.6 sec^{-1}	
CIII + *P2* $\underset{k_i}{\overset{k_i}{\rightleftharpoons}}$ *P2*·CIII	k_i = 0.01 nM^{-1} sec^{-1}	Selected to match CIII protection of CII from degradation (Hoyt *et al.* 1982; Rattray *et al.* 1984) and CII half life (Kobrtzer

Gillespie simulations (above) based on rates (left); references: Stochastic kinetic analysis of a developmental pathway bifurcation in phage-Escherichia coli cell. Arkin, Ross, McAdams (1998) Genetics 149: 1633-48

References and acknowledgements

http://bionumbers.hms.harvard.edu/
- 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, 195197, 1981.
- affine gaps:
http://courses.cs.washington.edu/courses/cse527/00wi/lectures/lect05.pdf
- D.S. Hirschberg, A linear space algorithm for computing longest common subsequences, Communications of the ACM 18, 341343, 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
- Ma B., Tromp J., and Li M. (2002) PatternHunter: faster and more sensitive homology search, Bioinformatics.

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, 11651173 (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 25:25178.
- 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. 2002 30:1575-84.

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.html?ic
- Viterbi: http://www.cs.umb.edu/ srevilak/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

Softare 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.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
- Various libraries to help with Biological data : www.biojava.org; www.bioperl.org; www.biopython.org; C++ 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.

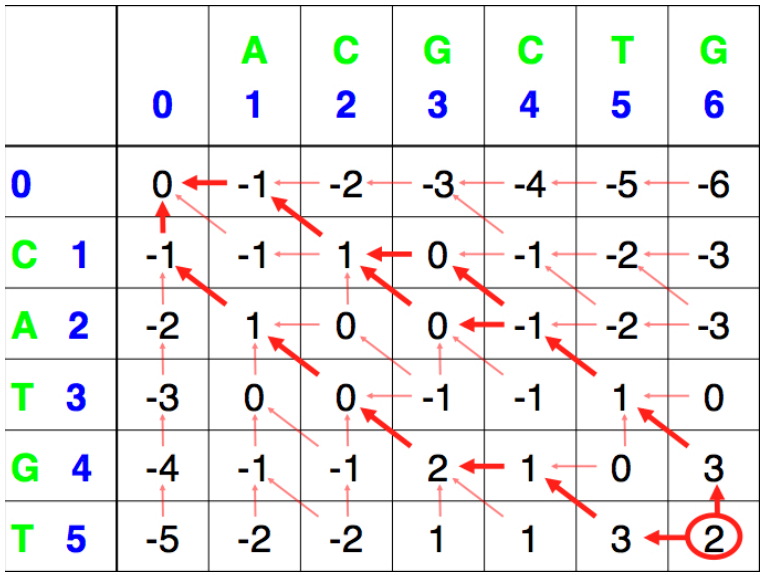
Figures acknowledgement (also orally during the lecture; it refers to the numbers in the copies distributed during the lectures, i.e. 4 slides/pages)

- 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 Koeppl'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 at 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



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 $nka << n^2$.

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

Bioinformatics and computational medicine

Understanding disease complexity is the definite scientific challenge of the twenty-first century medicine. The future foreseen is that computers will assist our health and disease conditions in a more effective way than nowadays: a medical check up will be supported by well-tuned artificial intelligence and patient-based modeling . At the clinical level, computer-aided therapies and treatments will develop into intervention strategies undertaken under acute disease conditions or due to external factors (infections) to contrast cascade effects.