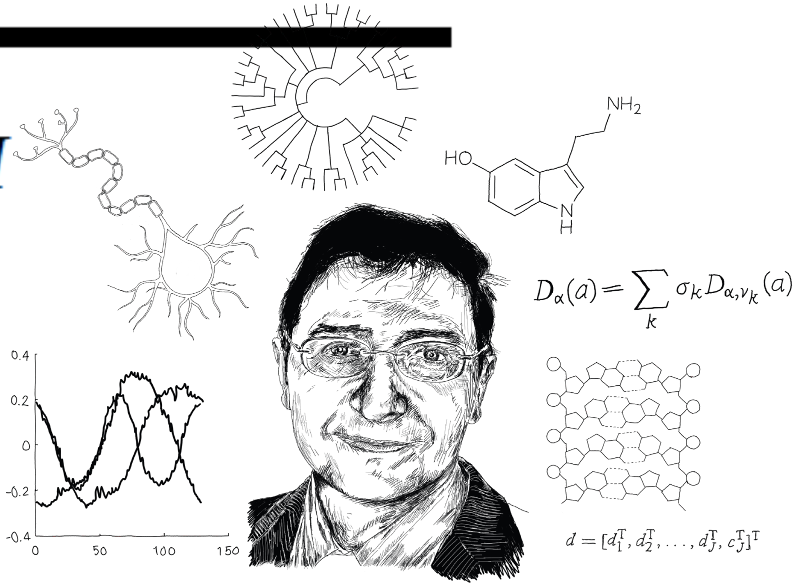
Bioinformatics

Michaelmas 2022

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Computer Science Tripos Part II


NH2
HO
N
H
$D_\alpha(a) = \sum_k \sigma_k D_{\alpha,\nu_k}(a)$
0.4
0.2
0
-0.2
-0.4
0
50
100
150
$d = [d_1^T, d_2^T, \ldots, d_J^T, c_J^T]^T$

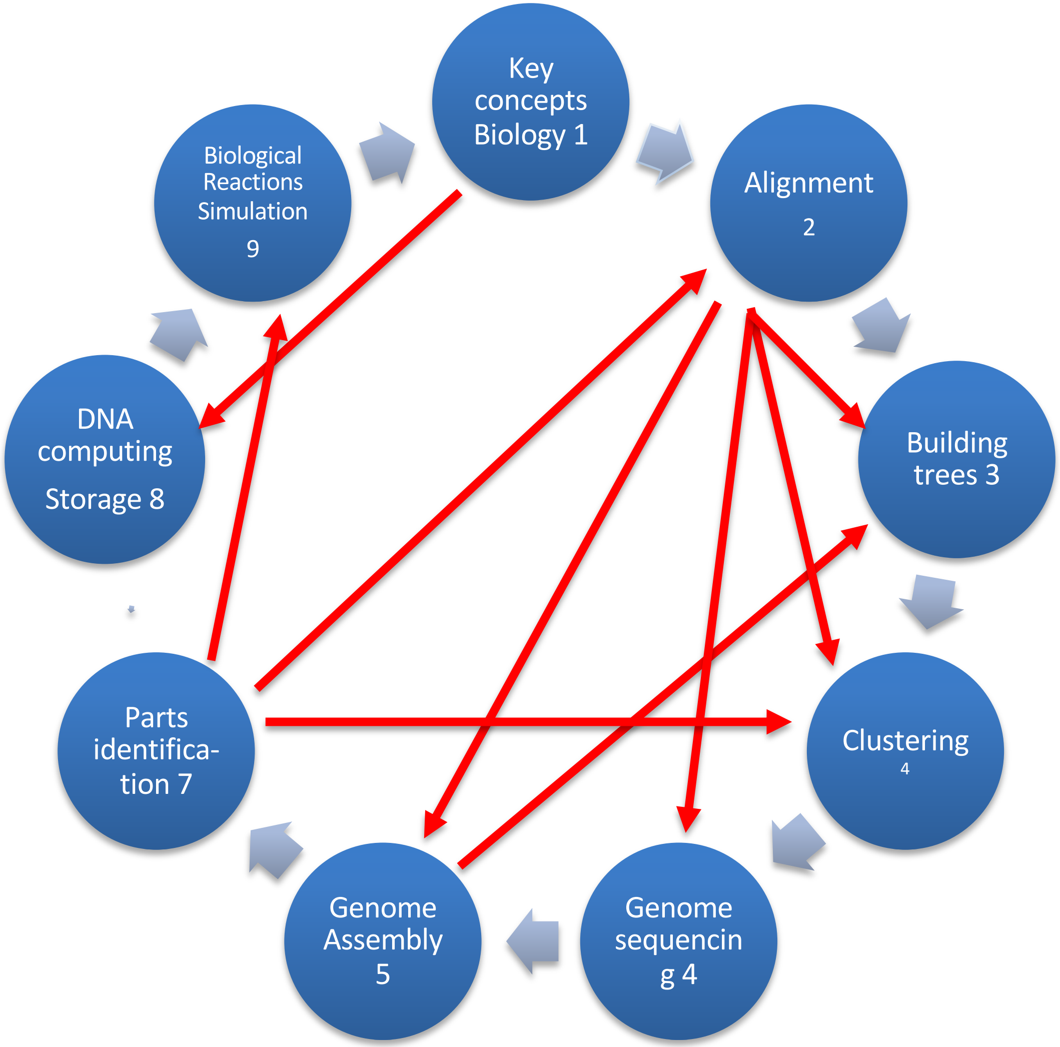
Pietro Lio`

Ph.D. Engineering,
Ph.D. Genetics

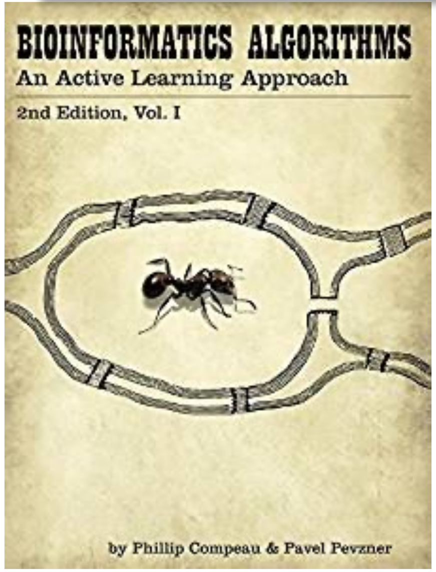
Bioinformatics: Interesting biological problems computer scientists should look at

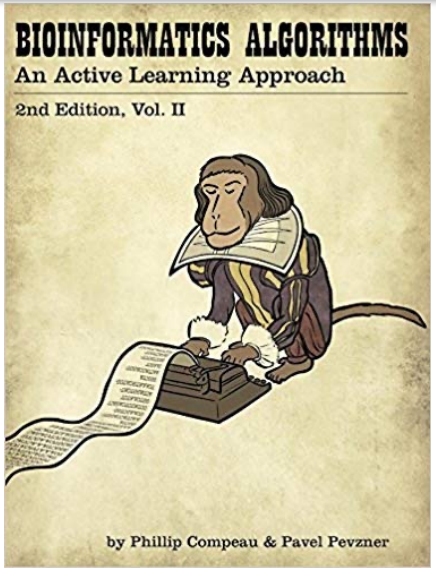
Bioinformatics offers an opportunity to help understand biology and medicine more accurately. Bioinformatics is an effective way to blend biological and medical concepts and programming tools to help understand biology and medicine better. A researcher must identify the widest variety of data that makes an organism. Second, she must know the context in which the disruption of information causes a disease.

Bioinformatics is nowadays about algorithms and machine learning methods. In the course we focus on algorithms. The course has 9 sections (figure right).



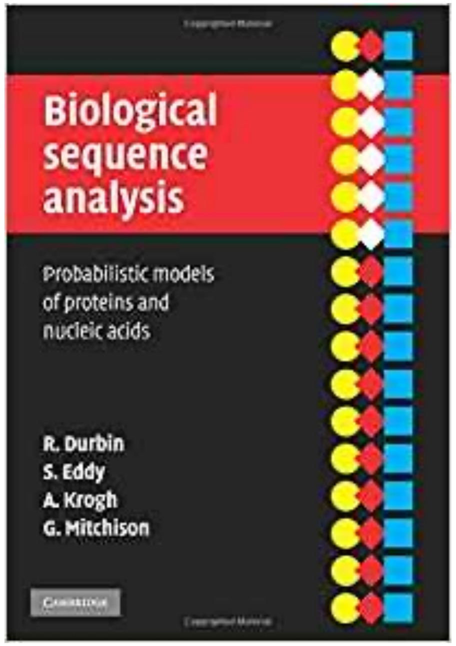
General references for the course


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. I
by Phillip Compeau & Pavel Pevzner


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. II
by Phillip Compeau & Pavel Pevzner

Course in 12 lectures;

largely based on P. Compeau and P. Pevzner: Bioinformatics algorithms;


Biological sequence analysis
Probabilistic models of proteins and nucleic acids
R. Durbin
S. Eddy
A. Krogh
G. Mitchison
CAMBRIDGE

also R. Durbin, at et al.: Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.

Bioinformatics Influence Animal Experimentation

Decades ago, legislation on the use of animals was enacted in many countries involving three R's: Reduction, refinement, and replacement of animal models.

Ever since this was enacted, there was a sudden buzz about laboratory animals and their use to be reduced, refined, and replaced wherever possible, for ethical and scientific reasons. The three R's concept was put forward by W.M.S. Russell and R.L. Burch in 1959 in The Principles of Humane Experimental Technique.

With the arrival of bioinformatics, the impact on animal experiments was slowly felt. The generation of high-throughput data in the form of genomics, transcriptomics, and metabolomics, biology has essentially transformed into a computational problem.

Due to this reason, we believe that the role of computation in biology leading to reducing, refining, and replacing animal experiments needs to be increased.

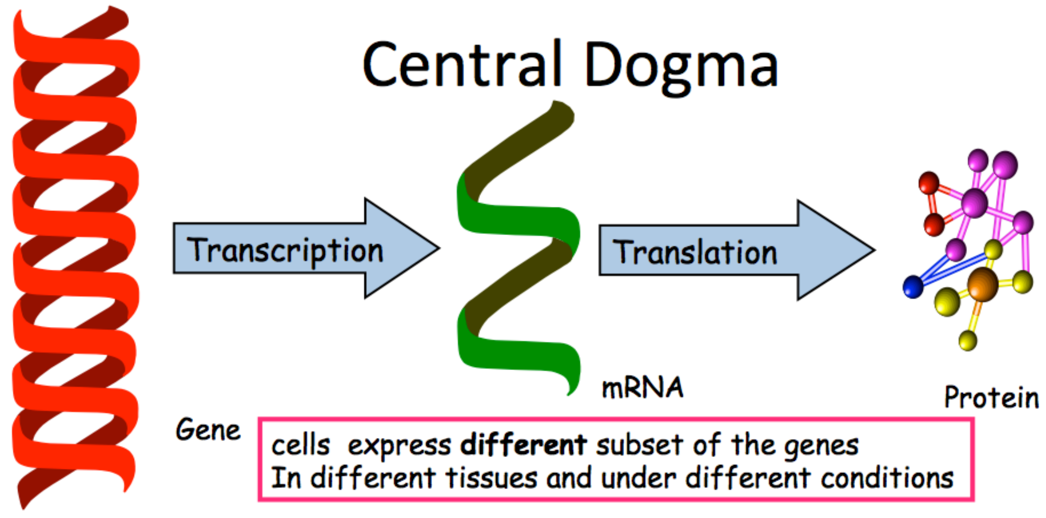
Section 1

Biological background

(non examinable)

- Structures and Models of DNA and proteins
- Multiple layers of information

Central Dogma and Genetic Code



cells express **different** subset of the genes
In different tissues and under different conditions

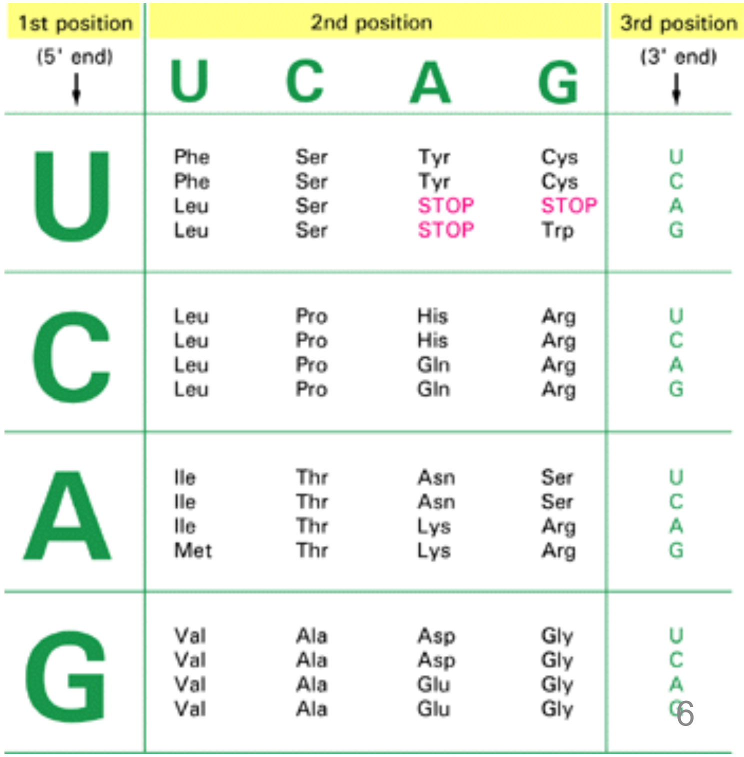
The central dogma of molecular biology explains the flow of genetic information, from DNA to RNA, to make a functional product, a protein.

The central dogma suggests that DNA contains the information needed to make all of our proteins, and that RNA is a messenger that carries this information to the ribosome

In transcription, the information in the DNA of every cell is converted into small, portable RNA messages.

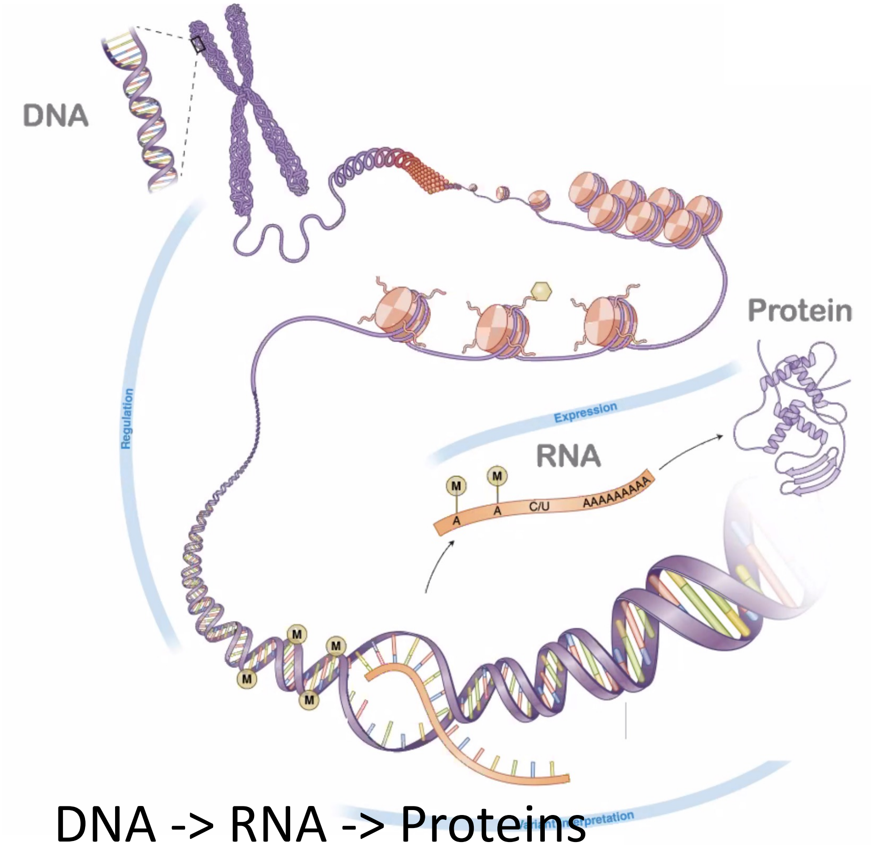
During translation, these messages travel from where the DNA is in the cell nucleus to the ribosomes where they are ‘read’ to make specific proteins using a genetic code (right).

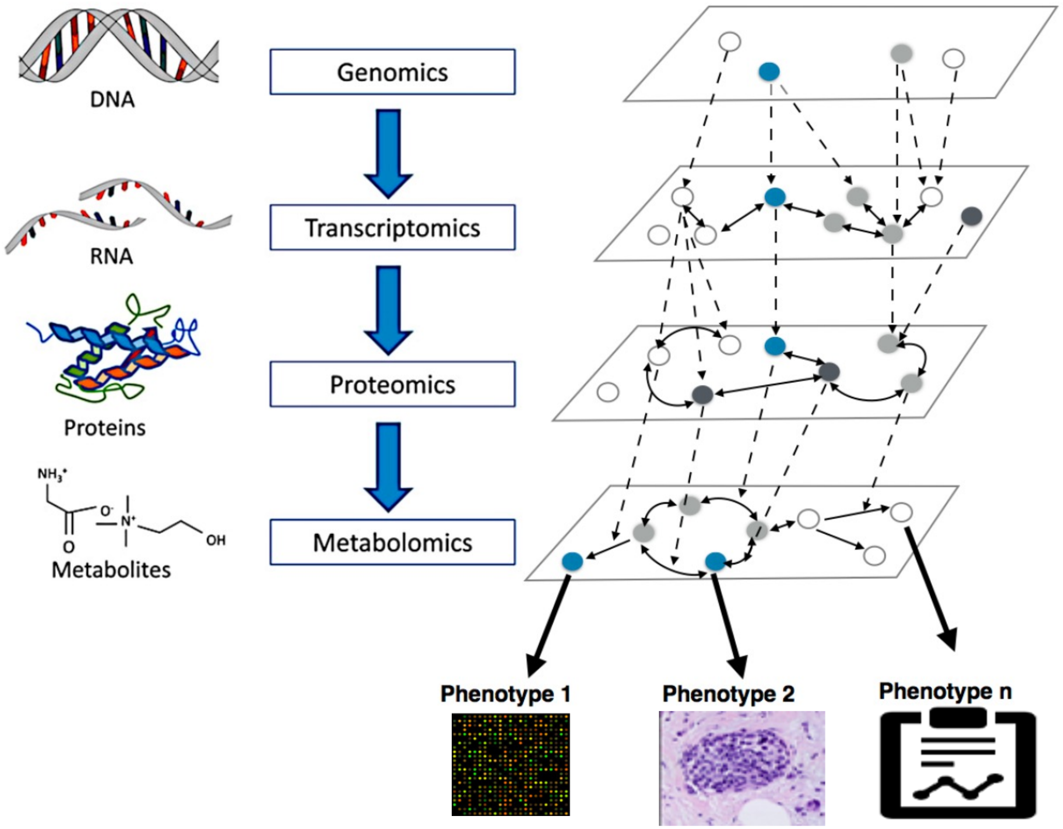
Gene expression is a tightly regulated process that increases or decreases the amount of proteins made.



1st position (5' end)	2nd position U	C	A	G	3rd position (3' end)
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	STOP	STOP	A
	Leu	Ser	STOP	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

Central Dogma and Genetic Code





DNA -> RNA -> Proteins

DNA encodes genes, most of which encode for proteins (via the genetic code)

Proteins perform much of the work of the cell.

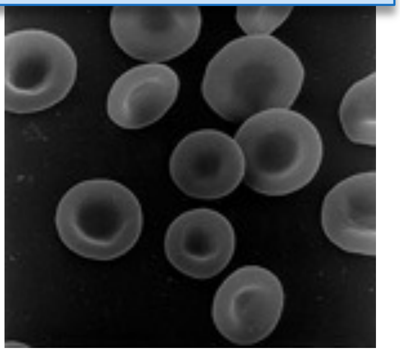
RNA acts as an intermediate step

(it also has other functions as well)

Huge amount of data now available, need algorithms to make sense of it.

Example: Healthy Individual

A gene is a string of DNA producing a “meaningful signal” for the cell/individual; most genes act as instructions to make proteins (others do not code for proteins); below the DNA sequence in Fasta format of the beta globin gene. It codes for a subunit (sequence below) of the hemoglobin protein.

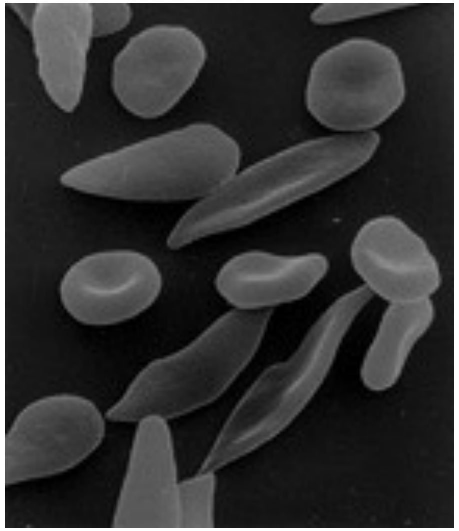


```
>gi|28302128|ref|NM_000518.4| Homo sapiens hemoglobin, beta (HBB), mRNA
ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACCATGGTGCATCTGACTCCTGA
GGAGAAGTCTGCCGTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGC
AGGCTGCTGGTGGTCTACCCTTGGACCCAGAGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATG
CTGTTATGGGCAACCCTAAGGTGAAGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGCCTGGC
TCACCTGGACAACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCACTGTGACAAGCTGCACGTGGAT
CCTGAGAACTTCAGGCTCCTGGGCAACGTGCTGGTCTGTGTGCTGGCCCATCACTTTGGCAAAGAATTCA
CCCCACCAGTGCAGGCTGCCTATCAGAAAGTGGTGGCTGGTGTGGCTAATGCCCTGGCCCACAAGTATCA
CTAAGCTCGCTTTCTTGCTGTCCAATTTCTATTAAAGGTTCCTTTGTTCCCTAAGTCCAACTACTAAACT
GGGGGATATTATGAAGGGCCTTGAGCATCTGGATTCTGCCTAATAAAAAACATTTATTTTCATTGC
```

```
>gi|4504349|ref|NP_000509.1| beta globin [Homo sapiens]

MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH
```

Databases: www.ebi.ac.uk, http://www.ncbi.nlm.nih.gov/ and others in China, Japan etc

Example: Individual with Sickle Cell Anemia

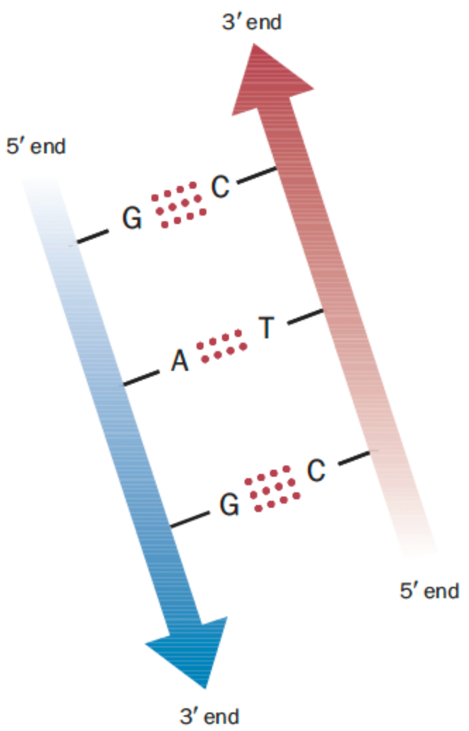


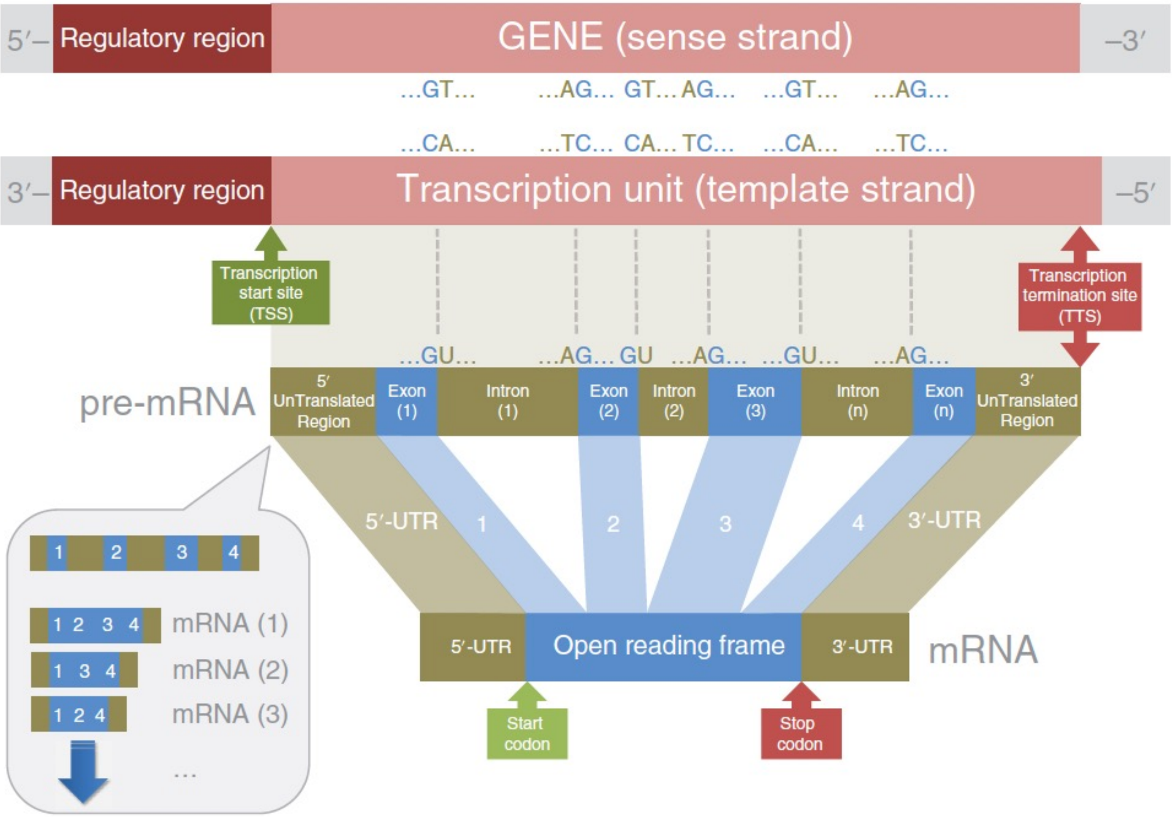
>gi|28302128|ref|NM_000518.4| Homo sapiens hemoglobin, beta (HBB), mRNA
ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACCATGGTGCATCTGACTCCTGA
GG**T**GAAGTCTGCCGTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGC
AGGCTGCTGGTGGTCTACCCTTGGACCCAGAGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATG
CTGTTATGGGCAACCCTAAGGTGAAGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGCCTGGC
TCACCTGGACAACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCACTGTGACAAGCTGCACGTGGAT
CCTGAGAACTTCAGGCTCCTGGGCAACGTGCTGGTCTGTGTGCTGGCCCATCACTTTGGCAAAGAATTCA
CCCCACCAGTGCAGGCTGCCTATCAGAAAGTGGTGGCTGGTGTGGCTAATGCCCTGGCCCACAAGTATCA
CTAAGCTCGCTTTCTTGCTGTCCAATTTCTATTAAAGGTTCCTTTGTTCCCTAAGTCCAACTACTAAACT
GGGGGATATTATGAAGGGCCTTGAGCATCTGGATTCTGCCTAATAAAAAACATTTATTTTCATTGC

>gi|4504349|ref|NP_000509.1| beta globin [Homo sapiens]

MVHLTP**V**EKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH

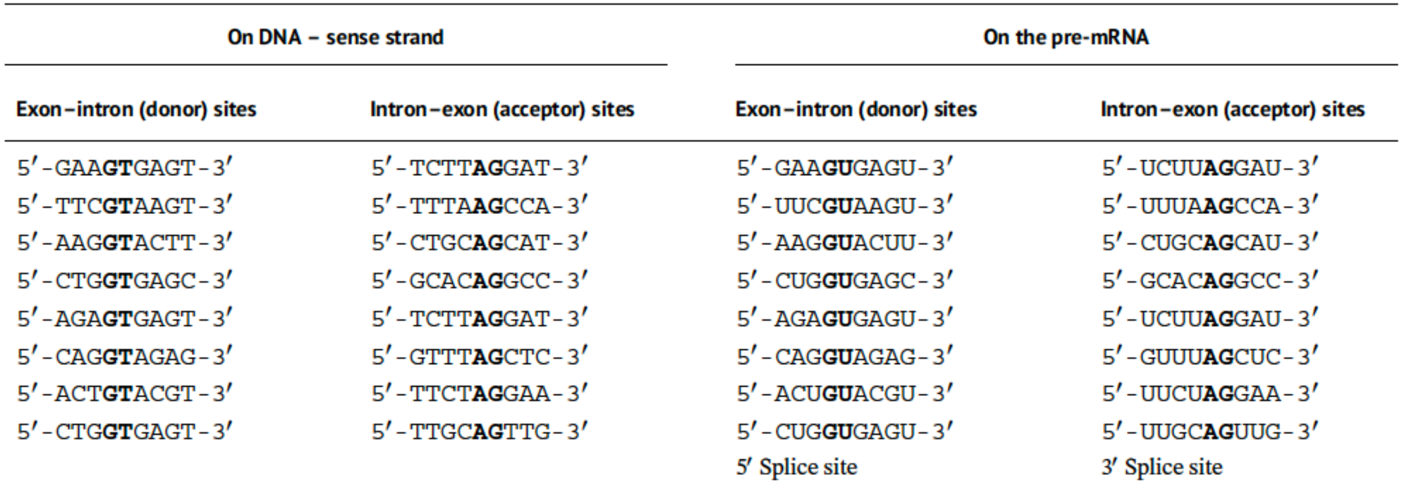
Structure of the gene





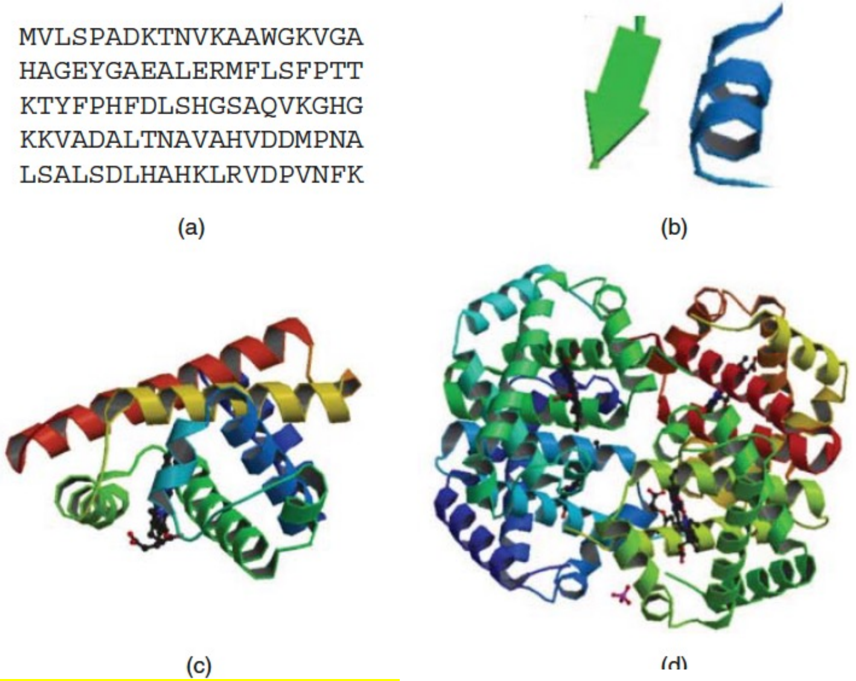
There is a pairing rule: A-T; C-G; the two strands of DNA are oriented differently

Examples of sequence containing an exon-intron boundary



On DNA – sense strand		On the pre-mRNA	
Exon–intron (donor) sites	Intron–exon (acceptor) sites	Exon–intron (donor) sites	Intron–exon (acceptor) sites
5′-GAA**GT**GAGT-3′	5′-TCTT**AG**GAT-3′	5′-GAA**GU**GAGU-3′	5′-UCUU**AG**GAU-3′
5′-TTC**GT**AAGT-3′	5′-TTTA**AG**CCA-3′	5′-UUC**GU**AAGU-3′	5′-UUUA**AG**CCA-3′
5′-AAG**GT**ACTT-3′	5′-CTGC**AG**CAT-3′	5′-AAG**GU**ACUU-3′	5′-CUGC**AG**CAU-3′
5′-CTG**GT**GAGC-3′	5′-GCAC**AG**GCC-3′	5′-CUG**GU**GAGC-3′	5′-GCAC**AG**GCC-3′
5′-AGA**GT**GAGT-3′	5′-TCTT**AG**GAT-3′	5′-AGA**GU**GAGU-3′	5′-UCUU**AG**GAU-3′
5′-CAG**GT**AGAG-3′	5′-GTTT**AG**CTC-3′	5′-CAG**GU**AGAG-3′	5′-GUUU**AG**CUC-3′
5′-ACT**GT**ACGT-3′	5′-TTCT**AG**GAA-3′	5′-ACU**GU**ACGU-3′	5′-UUCU**AG**GAA-3′
5′-CTG**GT**GAGT-3′	5′-TTGC**AG**TTG-3′	5′-CUG**GU**GAGU-3′	5′-UUGC**AG**UUG-3′
		5′ Splice site	3′ Splice site

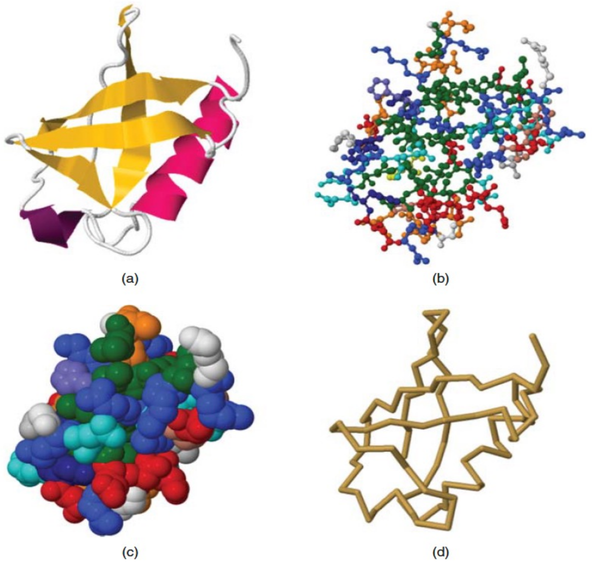
Proteins : from sequence to 3d structure



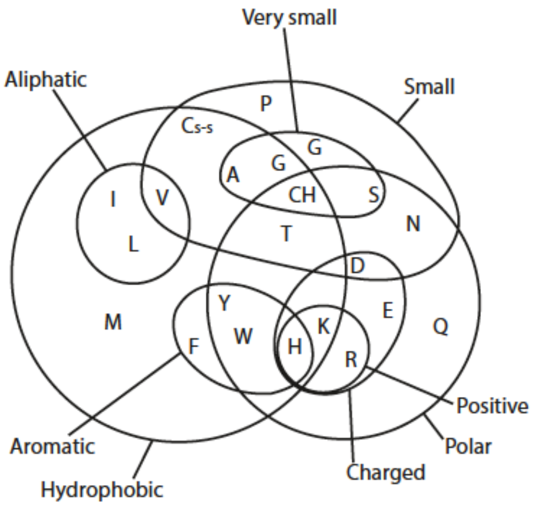
MVLSPADKTNVKAAWGKVGA
HAGEYGAEALERMFLSFPTT
KTYFPHFDLSHGSAQVKGHG
KKVADALTNAVAHVDDMPNA
LSALSDLHAHKLRVDPVNFK

Left: a) amino acid sequence; b) secondary structure; c) 3d structure; 4) quaternary structure (complex of proteins)

Below: various representations of 3d structure (a good free software is pymol)

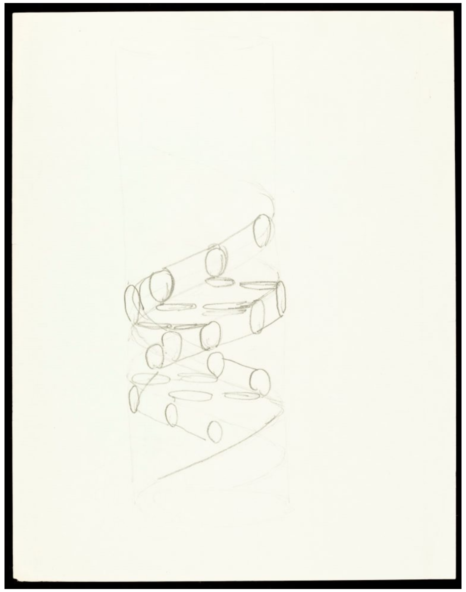


Clustering amino acids accordingly to physical and chemical properties: this provides evidences for effects of changes in proteins

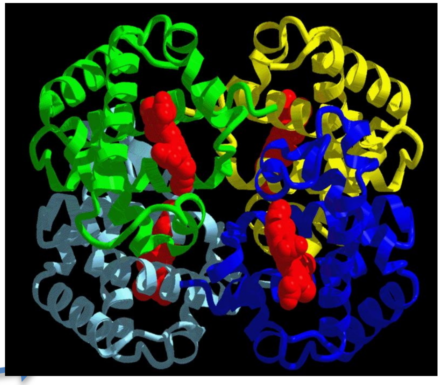


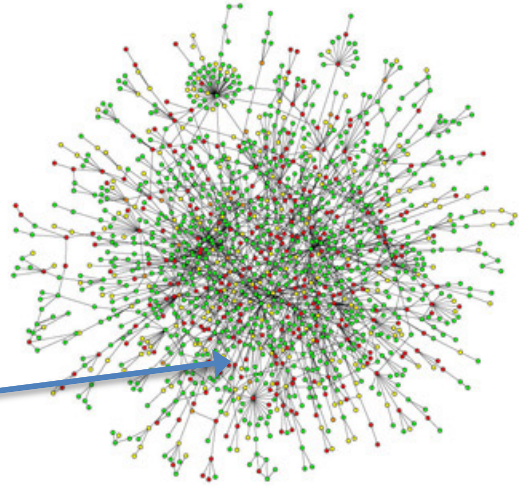
Structures and Models of DNA and proteins

Experimental data



5-CCTGAGCCAACTATTGATGAA-3
3-GGACTCGGTTGATAACTACTT-5





USEFUL ABSTRACTIONS:
DNA AS A STRING,
A PROTEIN AS A LABELLED GRAPH
DNA AND PROTEINS AS NETWORKS

sources: Photograph 51', March 1953, by Rosalind Franklin; pencil sketch of the DNA double helix by Francis Crick; replica of Crick and Watson's 1953 DNA Double Helix Model, source https://blog.sciencemuseum.org.uk/why-the-double-helix-is-still-relevant/

Parallel Technological evolution

High-performance computing

1979

today

Who has a computer?

- 1960s: Major research institutes
- 1970s: University departments
- 1980s: Companies and schools
- 2019: Almost everybody & always

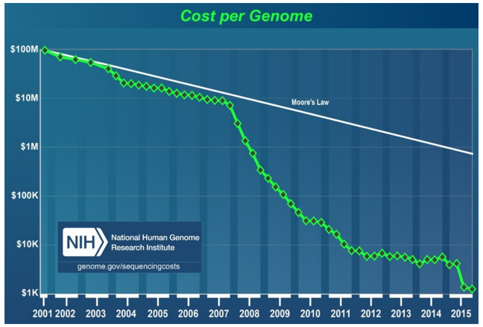
Genome sequencing

2006

today

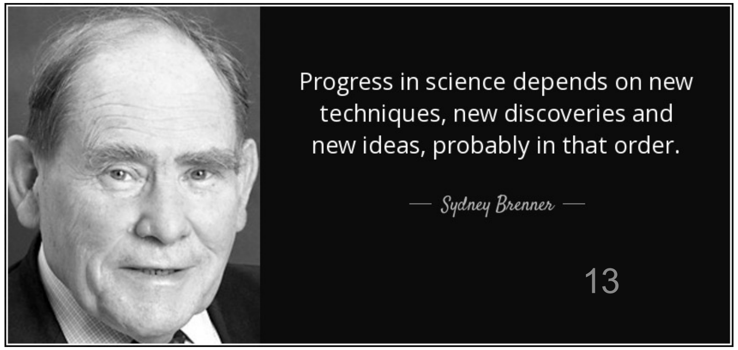
Whose genome has been sequenced?

- 1996: First bacterium (E. coli)
- 2001: Human reference genome
- 2007: First personal genomes
- 2020: Millions personal genomes



Your genome in your mobile for few hundred pounds:

https://www.genome.gov/sequencingcosts



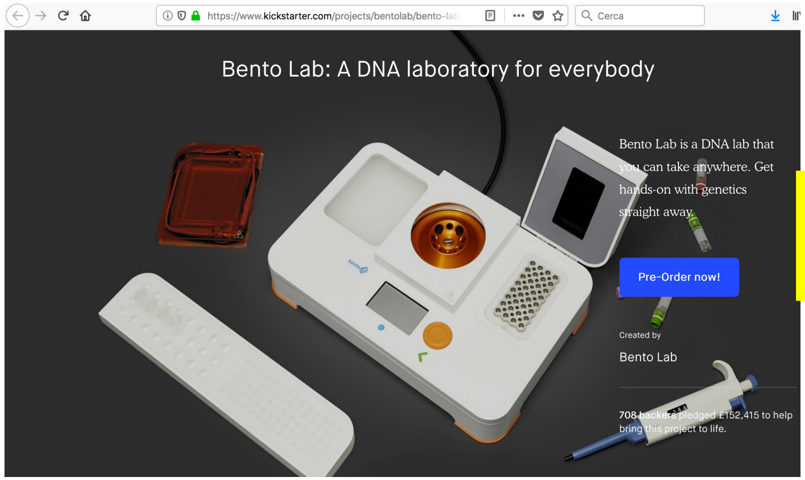
Progress in science depends on new techniques, new discoveries and new ideas, probably in that order.

— Sydney Brenner —

Determining the sequence of DNA is cheap and quick

Oxford nanopore

- The algorithms we study have impact on precision and personalised medicine:
- **Cancer:** Disease stratification based on driver mutations
- **Rare diseases:** Most patients now receive a genetic diagnosis
- **Drugs:** Patient-specific prediction of efficacy and side effects



Bento Lab: A DNA laboratory for everybody

Bento Lab is a DNA lab that you can take anywhere. Get hands-on with genetics straight away.

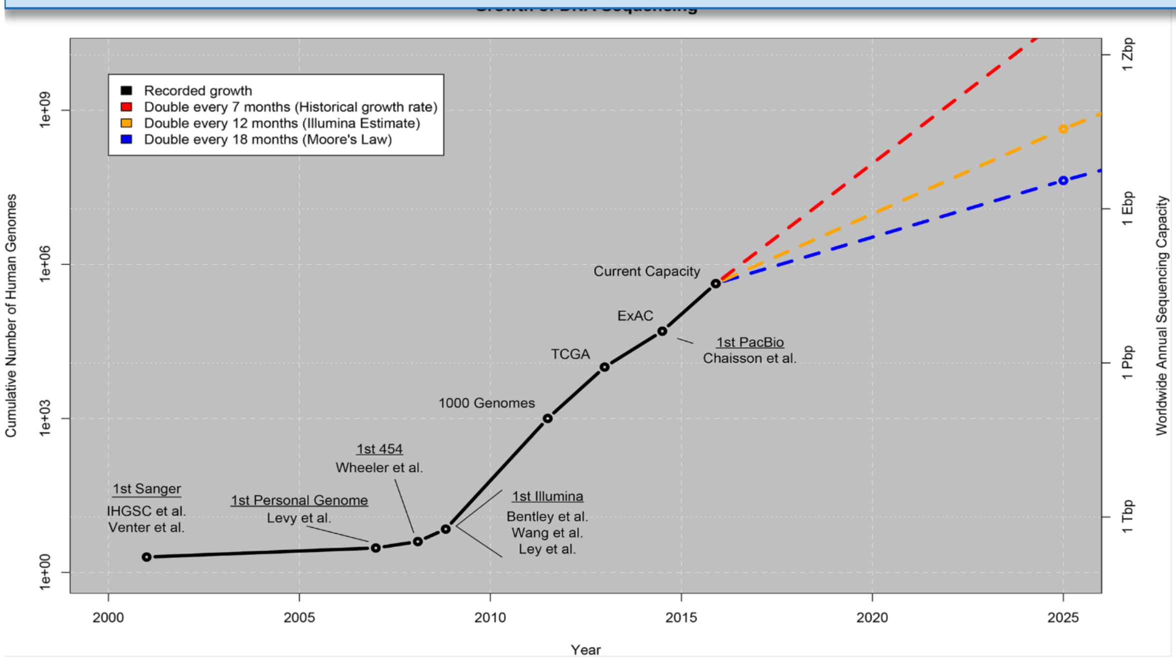
Created by

Bento Lab

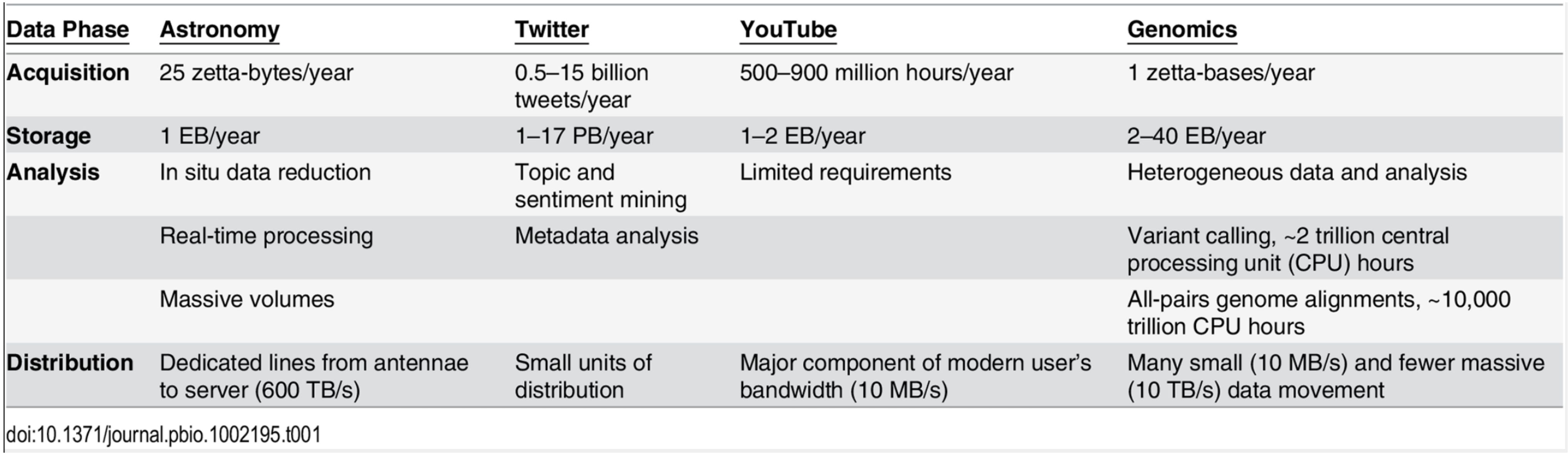
708 backers pledged £152,415 to help bring this project to life.

Garage genomics

DNA is big data



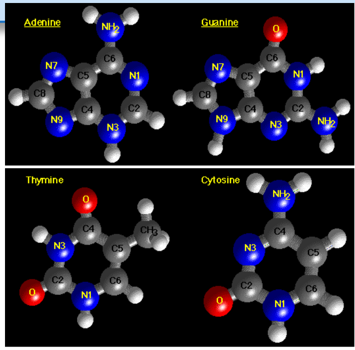
Data Repository: http://www.ebi.ac.uk; http://www.ncbi.nlm.nih.gov/ ; http://genome.ucsc.edu/ www.ensembl.org



Data Phase	Astronomy	Twitter	YouTube	Genomics
Acquisition	25 zetta-bytes/year	0.5–15 billion tweets/year	500–900 million hours/year	1 zetta-bases/year
Storage	1 EB/year	1–17 PB/year	1–2 EB/year	2–40 EB/year
Analysis	In situ data reduction	Topic and sentiment mining	Limited requirements	Heterogeneous data and analysis
	Real-time processing	Metadata analysis		Variant calling, ~2 trillion central processing unit (CPU) hours
	Massive volumes			All-pairs genome alignments, ~10,000 trillion CPU hours
Distribution	Dedicated lines from antennae to server (600 TB/s)	Small units of distribution	Major component of modern user's bandwidth (10 MB/s)	Many small (10 MB/s) and fewer massive (10 TB/s) data movement

doi:10.1371/journal.pbio.1002195.t001

Dense information



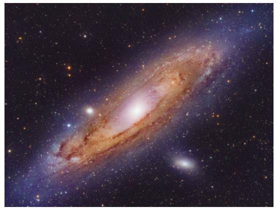
Purines
(1 bit)

Pyrimidine
(1 bit)

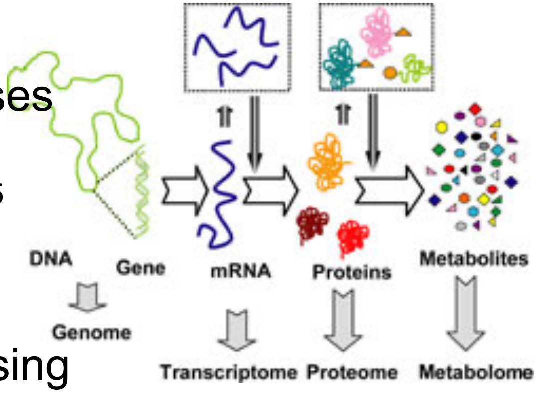
Each DNA base encodes 2 bits information (because you have to choose between purines and pyrimidines and then within each class).

You have 46 chromosomes in each (autosomal) cell.
If you tease out those 46 (double) strands and place them end to end they'd be about 2 metres long - but that's just one cell. Every time a cell replicates it has to copy 2 meters of DNA reliably. 3 billion base pairs, 2 meters long, 2nm thick, folded into a 6μm ball/cell.

As there are about 3.7×10^{13} cells in the human body (and hence 1.7×10^{15} chromosomes or strands), your entire DNA would stretch about 7.4×10^{10} km or fifty thousand million miles (133 Astronomical Units long) and the DNA in the current human population would be 20 million light years long (the Andromeda Galaxy is 2.5 Million light years).



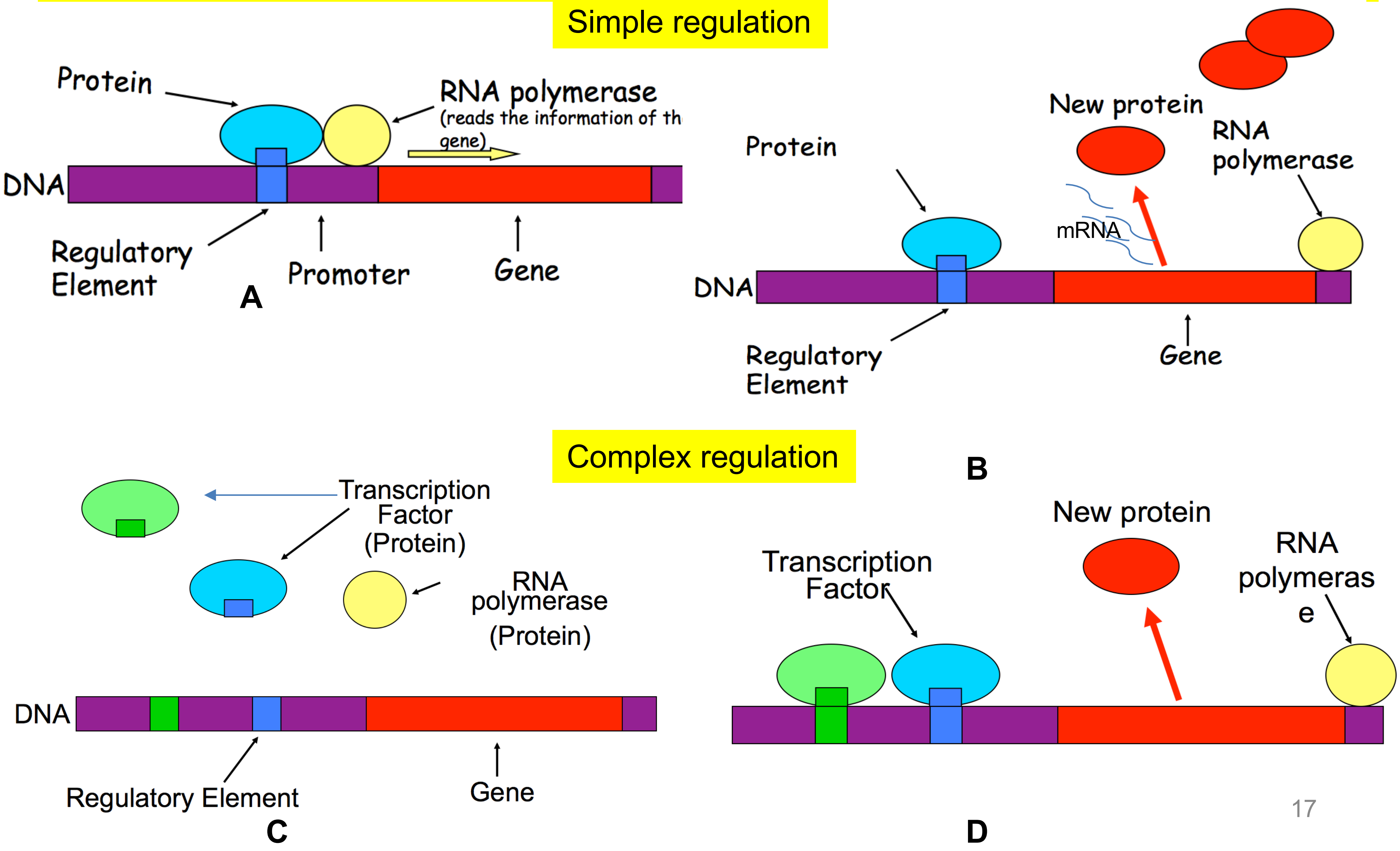
Big numbers also as information content: lower bound on the total information content in the biosphere: 5.3 × 10^{31} (±3.6 × 1031) megabases (Mb) of DNA. Taking the rate of DNA transcription as an analogy for processing speed, further estimated Earth's computational power: 10^{15} yottaNOPS (1024 Nucleotide Operations Per Seconds).



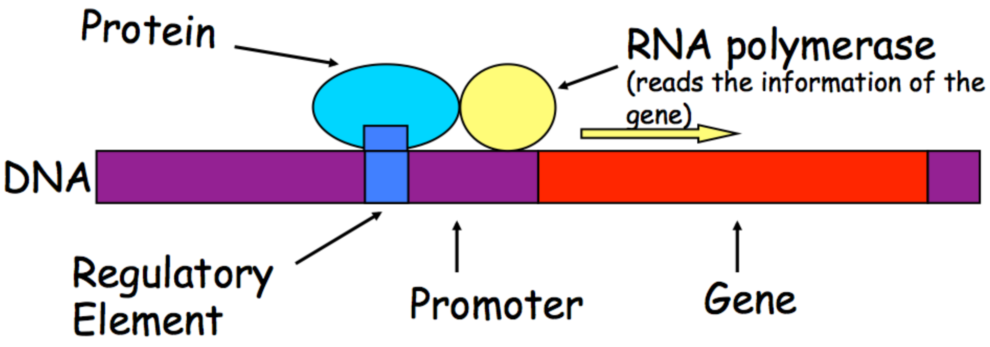
Then you can take into account all the other flow of information processing such as proteomics, metabolomics etc

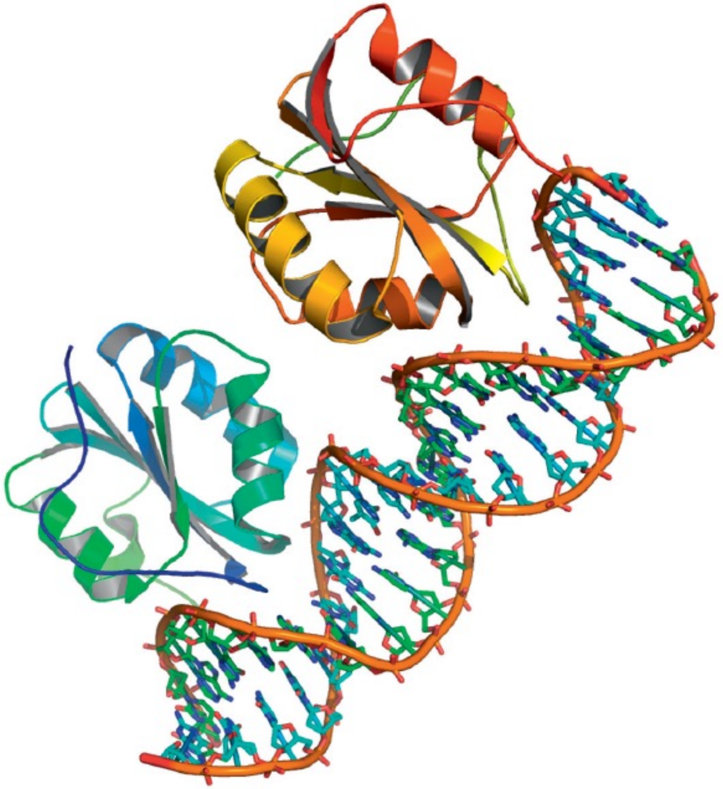
Gene and protein interactions as graphs

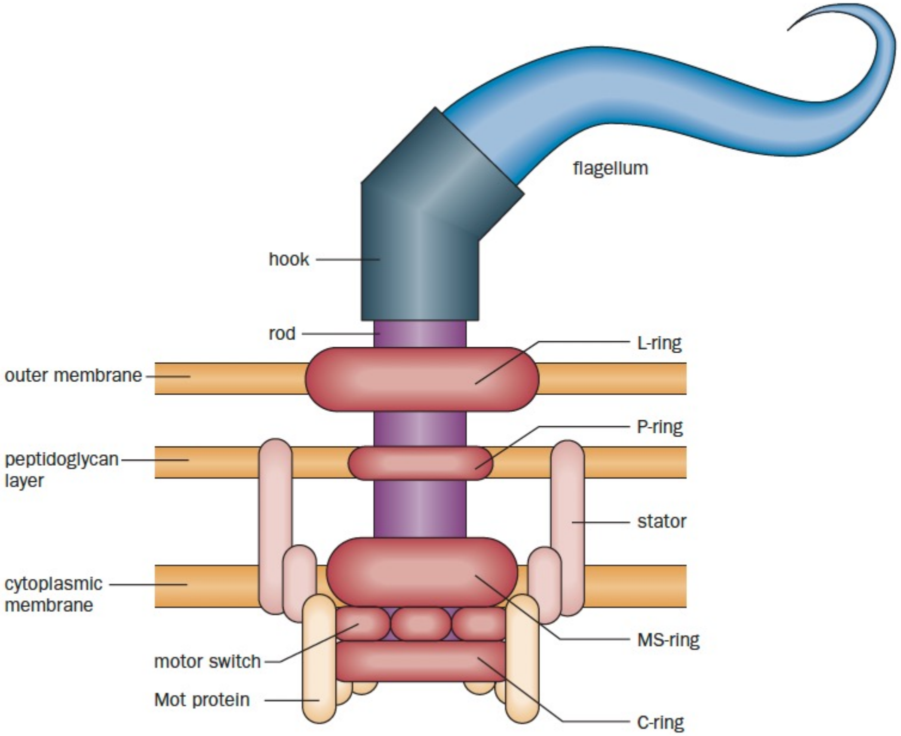
Genes are activated or repressed by regulatory proteins which bind to gene flanking sequences (promoter) and are coded by the same or other genes. What is below just multiply many times for each gene and for thousands of genes



Gene and protein interactions build devices







This complex protein arrangement allows bacteria to swim in different directions. Similar assemblies are found in sperm cells, in the fallopian tubes (where eggs need to be moved from the ovary to the uterus), and in the respiratory tract (where cilia clear the airways of mucus and debris).

Cells versus Computers

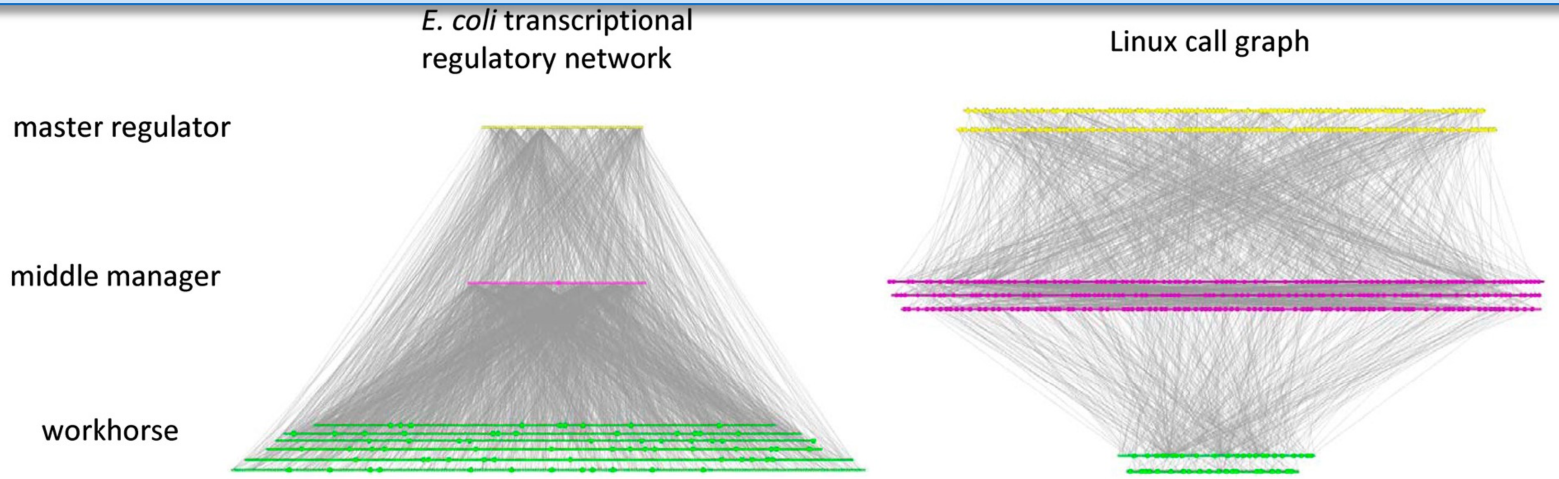
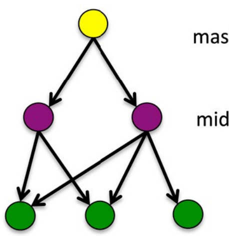


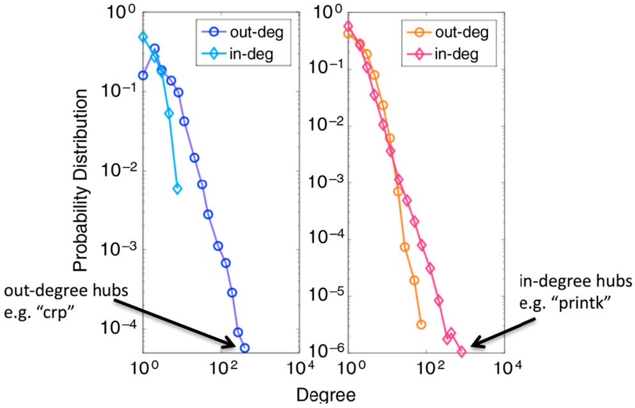
Fig. 1. The hierarchical layout of the *E. coli* transcriptional regulatory network and the Linux call graph. (*Left*) The transcriptional regulatory network of *E. coli*. (*Right*) The call graph of the Linux Kernel. Nodes are classified into three categories on the basis of their location in the hierarchy: 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



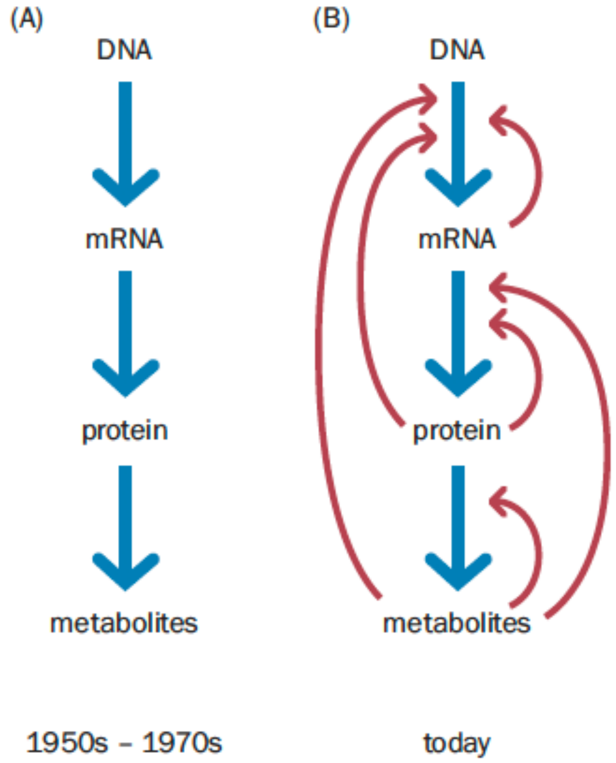
	percentage in E. coli regulatory network	percentage in Linux call graph
master regulator	4.6	29.6
middle manager	5.1	58.2
workhorse	90.2	12.3

B

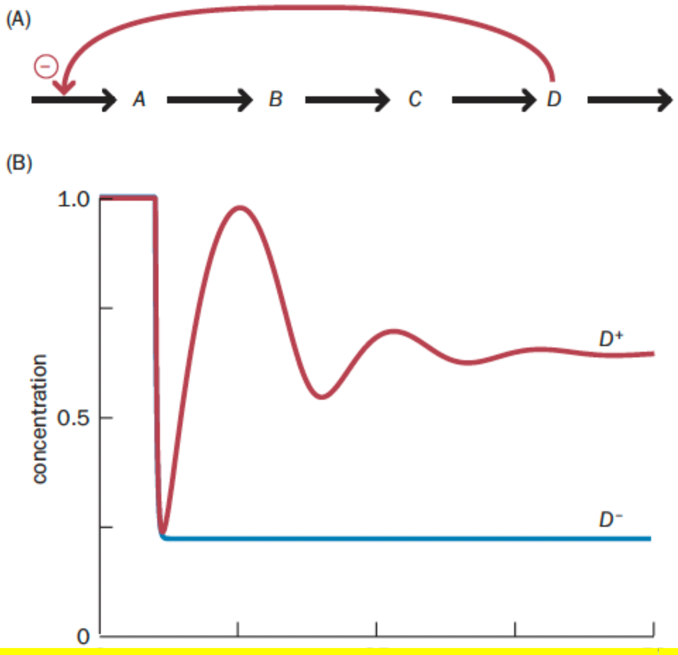


The transcriptional regulatory network (1,378 nodes) follows a conventional hierarchical picture, with a few top regulators and many workhorse proteins. The Linux call graph (12,391 nodes), on the other hand, possesses many regulators; the number of workhorse routines is much lower in proportion. The 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.

Central dogma revisited and Regulation Feedback

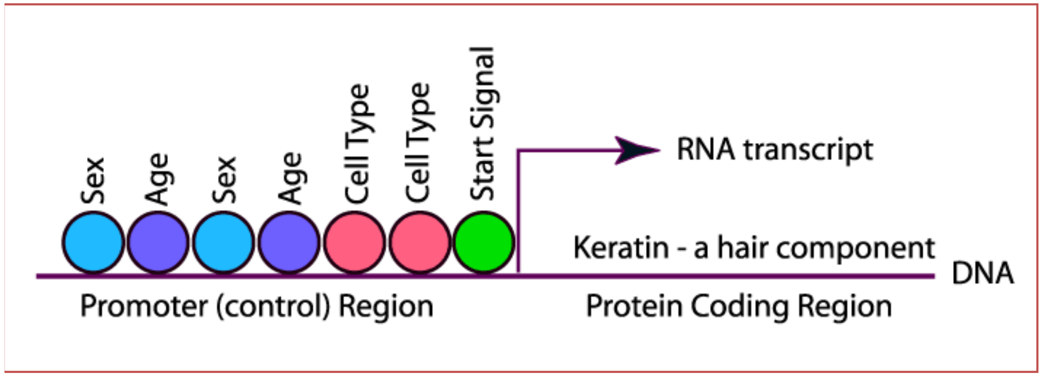


The original Central Dogma compared with modern understanding. In the original concept of the Central Dogma, transcription, translation, and enzymatic catalysis were proposed to form a linear chain of processes, although nobody doubted the role of regulation. We know now that a complex feedback structure at every level is crucial for appropriate cell functioning.

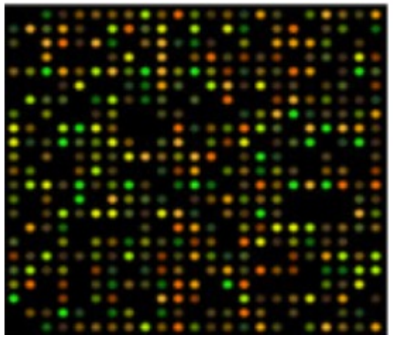


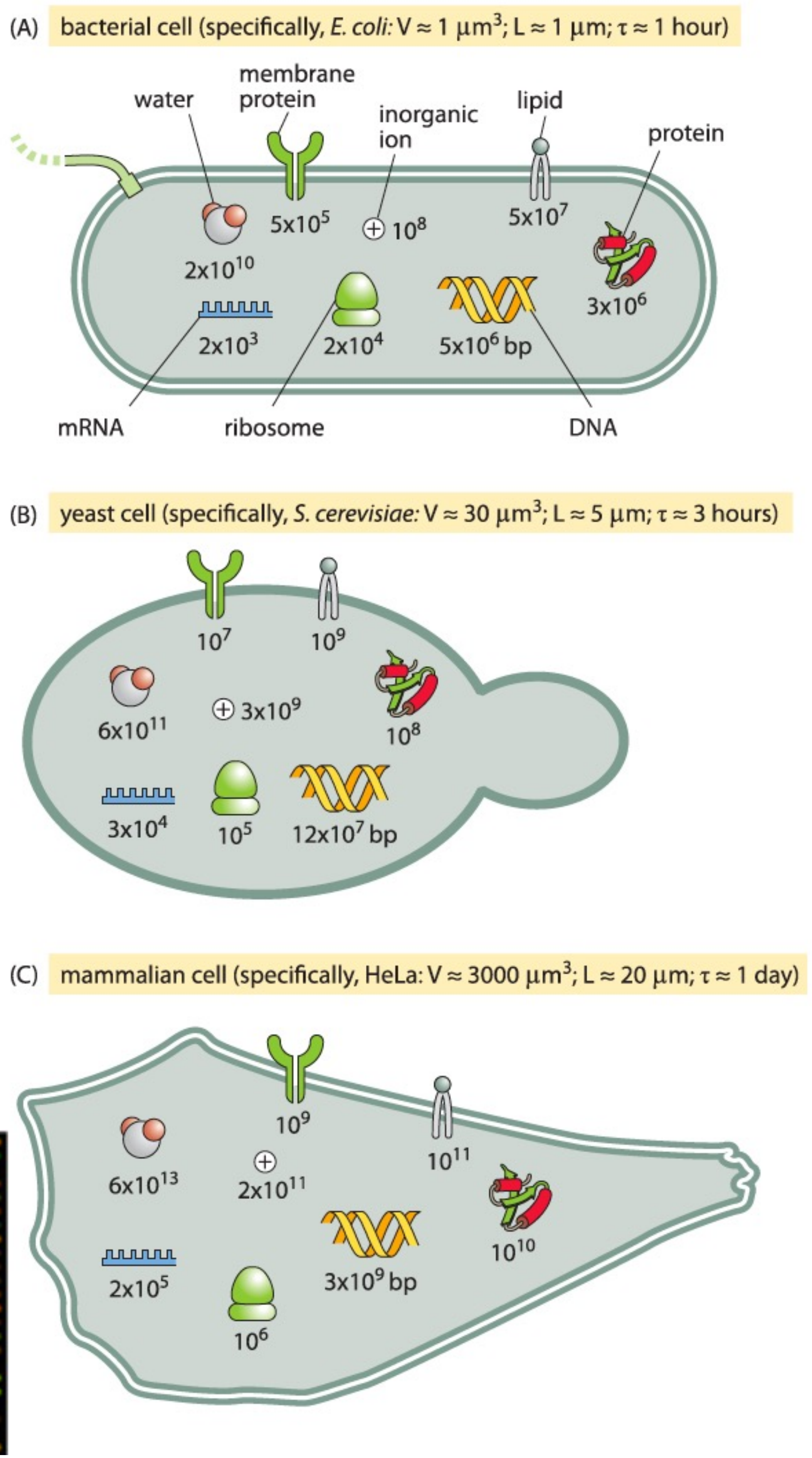
Linear pathway with feedback. (A) Reaction scheme with feedback inhibition of the initial step by the end product. (B) Comparison of responses to a sudden and persistent demand of metabolite D, starting at time t = 8. With feedback (D+), the concentration of D oscillates and converges to a level of about two-thirds of the initial value. Without feedback (D−), the concentration of D sinks to less than one-quarter.

The Cell is a Computer in Soup

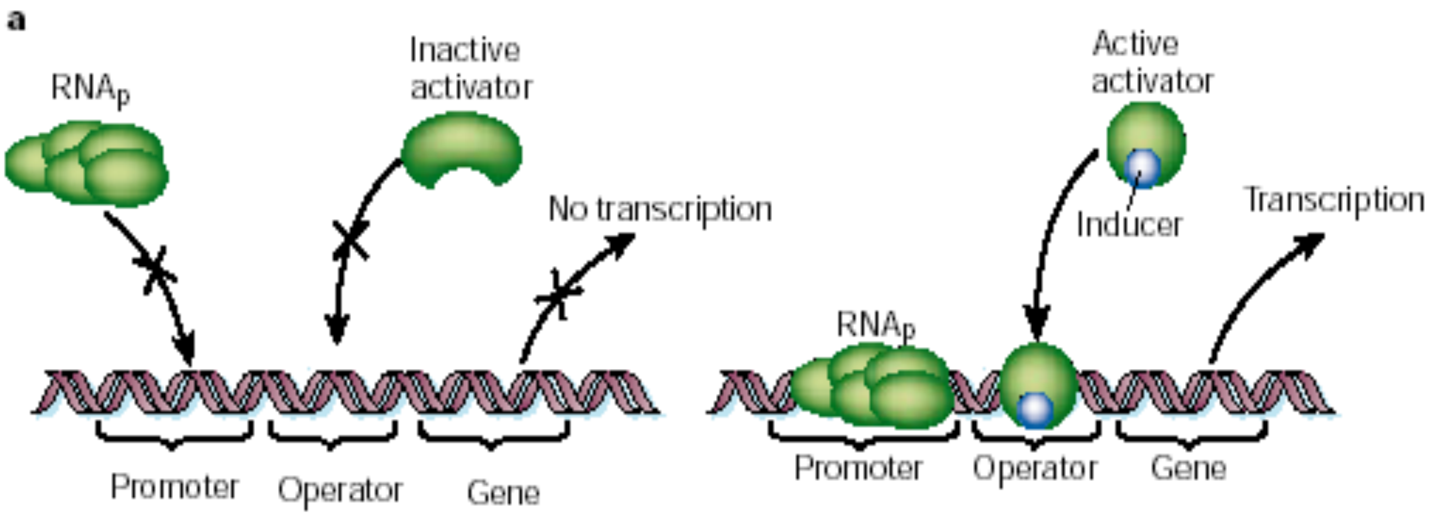


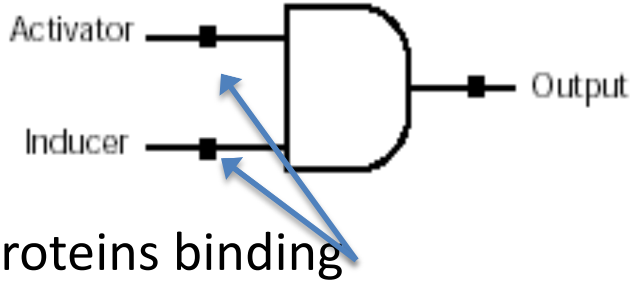
ABOVE: Idealized promoter for a gene involved in making hair. Proteins that bind to specific DNA sequences in the promoter region together turn a gene on or off. These proteins are themselves regulated by their own promoters leading to a gene regulatory network with many of the same properties as a neural network. We use chips (right) to measure the activity of all the genes (rows) in different conditions (gene Expression, columns).




(A) bacterial cell (specifically, E. coli: V ≈ 1 μm³; L ≈ 1 μm; τ ≈ 1 hour)
water
membrane protein
inorganic ion
lipid
protein
2x10¹⁰
5x10⁵
10⁸
5x10⁷
3x10⁶
2x10³
2x10⁴
5x10⁶ bp
mRNA
ribosome
DNA
(B) yeast cell (specifically, S. cerevisiae: V ≈ 30 μm³; L ≈ 5 μm; τ ≈ 3 hours)
10⁷
10⁹
6x10¹¹
3x10⁹
10⁸
3x10⁴
10⁵
12x10⁷ bp
(C) mammalian cell (specifically, HeLa: V ≈ 3000 μm³; L ≈ 20 μm; τ ≈ 1 day)
10⁹
10¹¹
6x10¹³
2x10¹¹
10¹⁰
2x10⁵
3x10⁹ bp
10⁶

Logic gates: The Cell as an information processing device

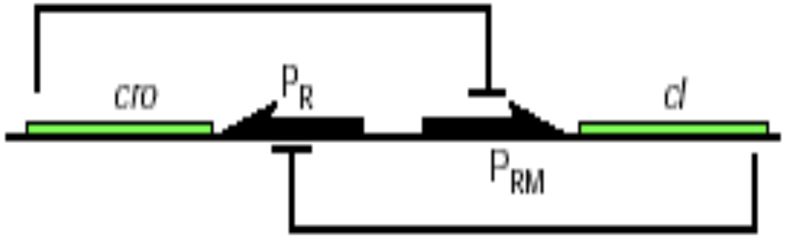




Activator	Inducer	Output
0	0	0
0	1	0
1	0	0
1	1	1

proteins binding regulatory elements

Toggle switch (cro and cl are genes; Pr and Prm are binding sites for the proteins of genes cro and cl)



Biology as Network of networks?

It is no doubt that reductionism applied to biology has scored some notable successes. For example, specific defective genes have been linked to a number of heritable conditions such as Tay-Sachs syndrome.

But it soon became clear that there is generally no simple connection between a gene, or a set of genes, and a biological trait at the level of the organism.

Many traits emerge only when the system as a whole is taken into account, including entire networks of genes in interaction, plus many non-genetic, or so-called epigenetic, factors that may also involve the environment (a topic to which I shall return in the next chapter).

And when it comes to social organisms – for example, ants, bees and humans – a complete account requires consideration of the collective organization of the whole community

Complexity of living systems: the cell, the fundamental unit in biology, as a network of genes and proteins

The detailed inventory of genes, proteins, and metabolites is not sufficient to understand the cell's complexity

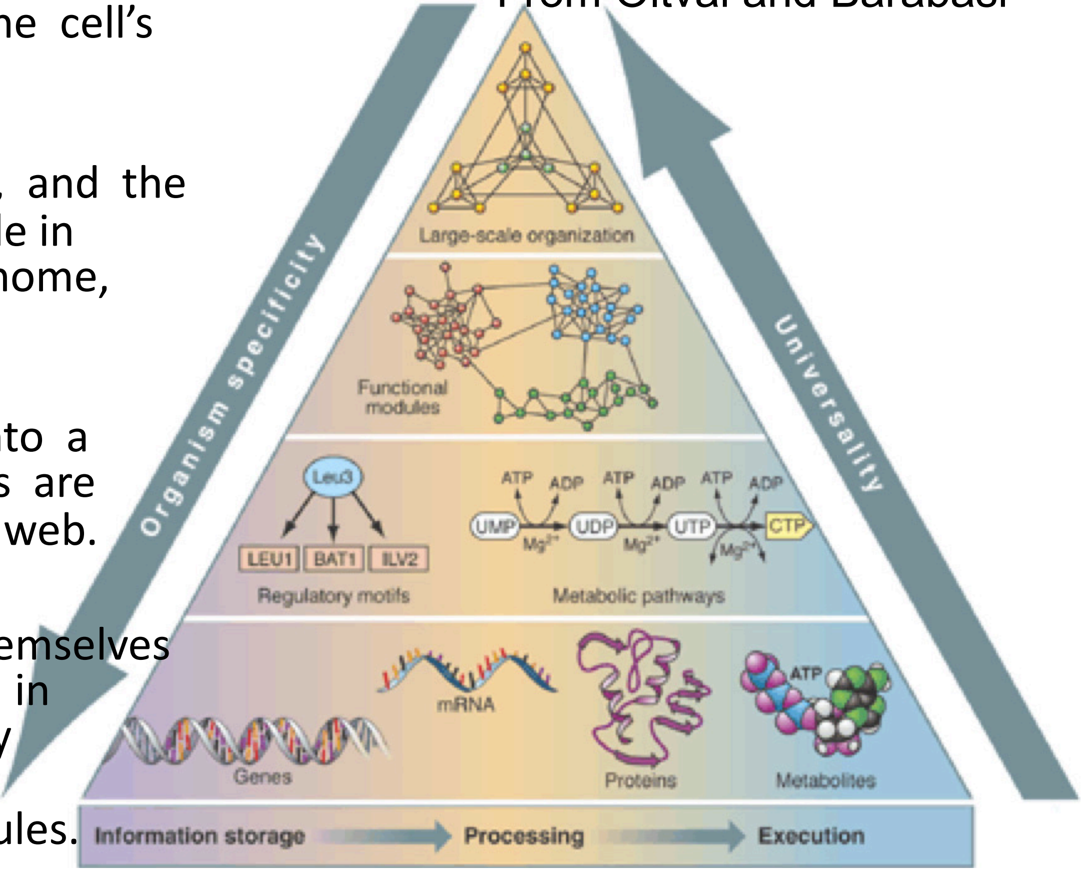
Information storage, information processing, and the execution of various cellular programs reside in distinct levels of organization: the cell's genome, transcriptome, proteome and metabolome.

For example, the proteome organizes itself into a protein interaction network and metabolites are interconverted through an intricate metabolic web.

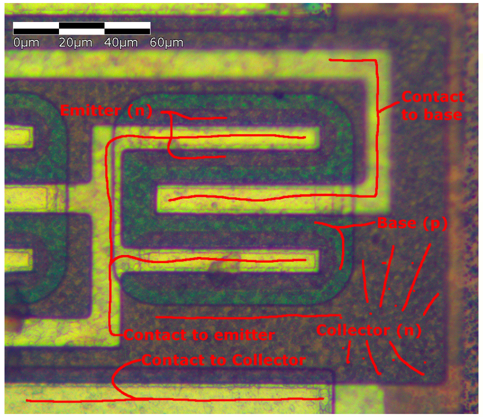
The elementary building blocks organize themselves into small recurrent patterns, called pathways in metabolism and motifs in genetic-regulatory networks. In turn, motifs and pathways are seamlessly integrated to form functional modules.

These modules are nested, generating a scale-free hierarchical architecture . Although the individual components are unique to a given organism,the topologic properties of cellular networks share surprising similarities with those of natural and social networks.

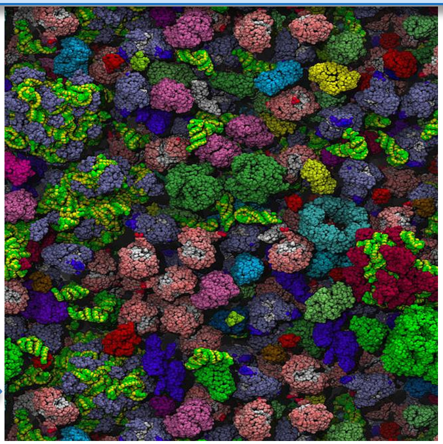
From Oltvai and Barabási

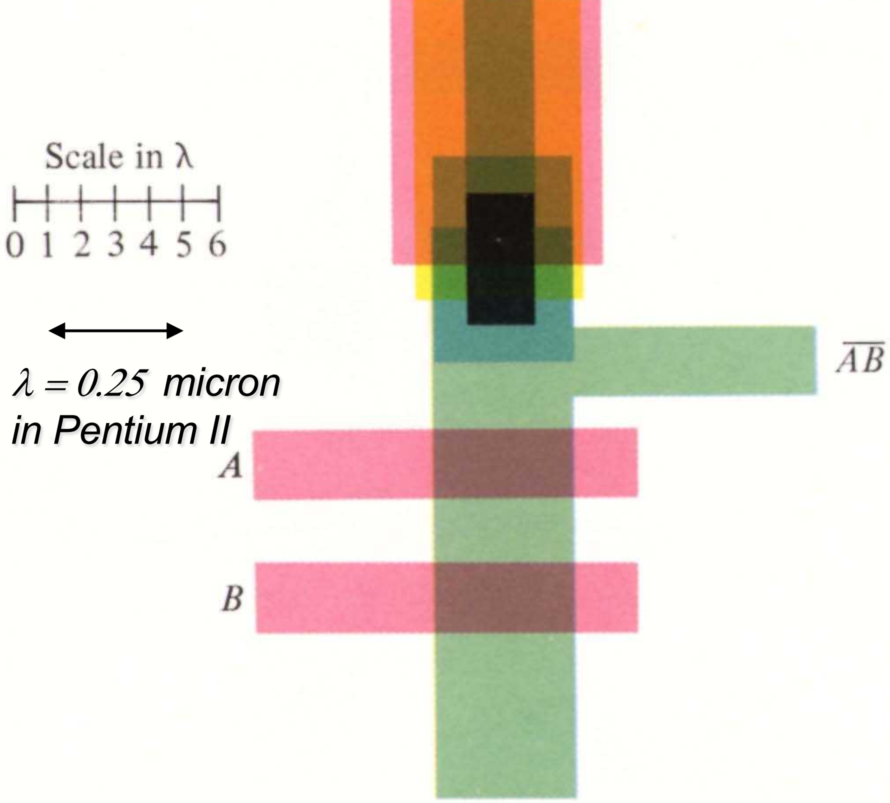


Scales of electronic and bio devices



proteins inside a bacterium



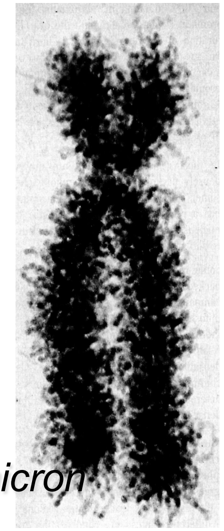


$\lambda = 0.25$ *micron in Pentium II*

(a) NAND gate layout geometry.

1 micron

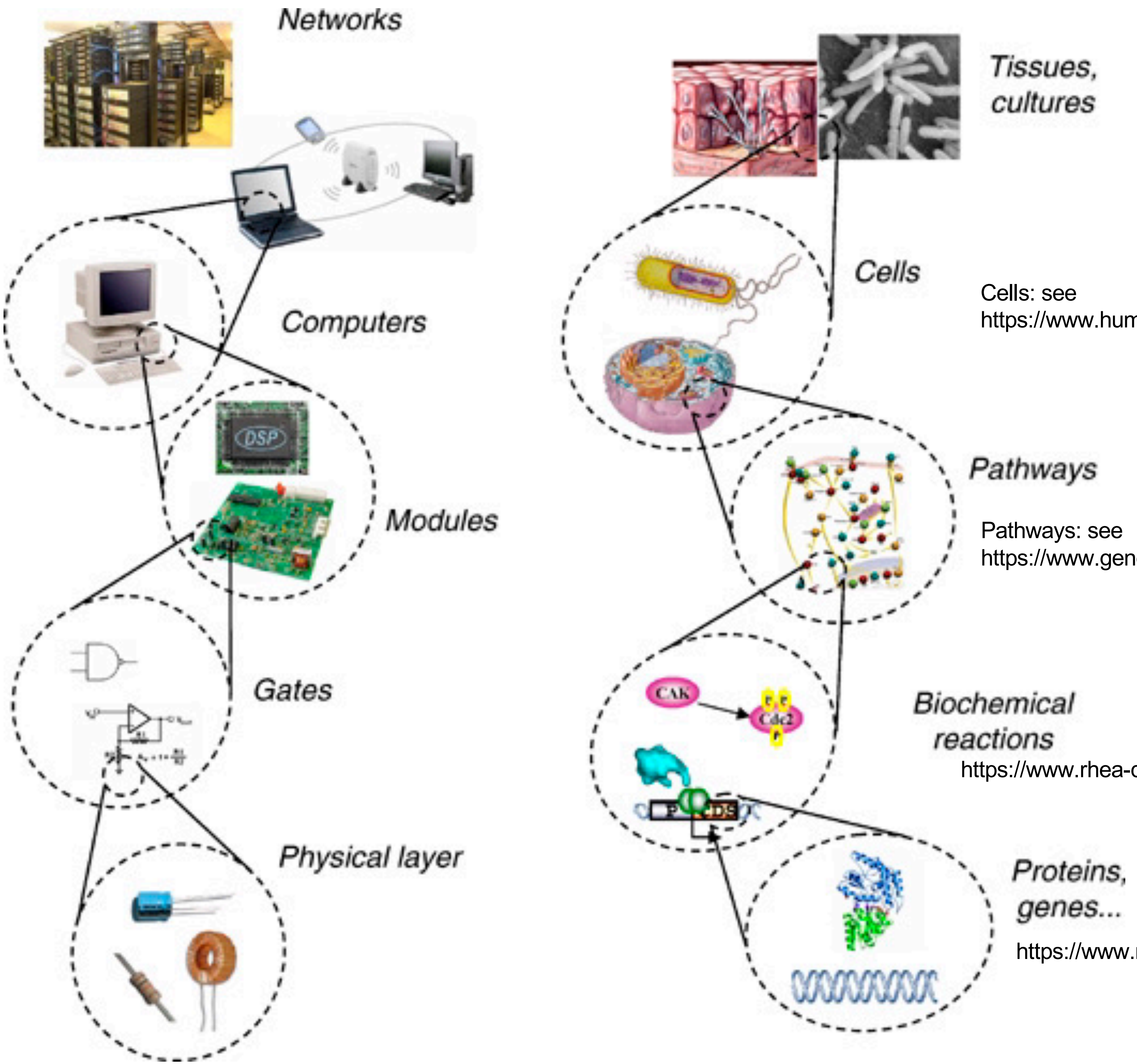
Bacterium



1 micron

Human chromo some.

CS –Bio parallelism



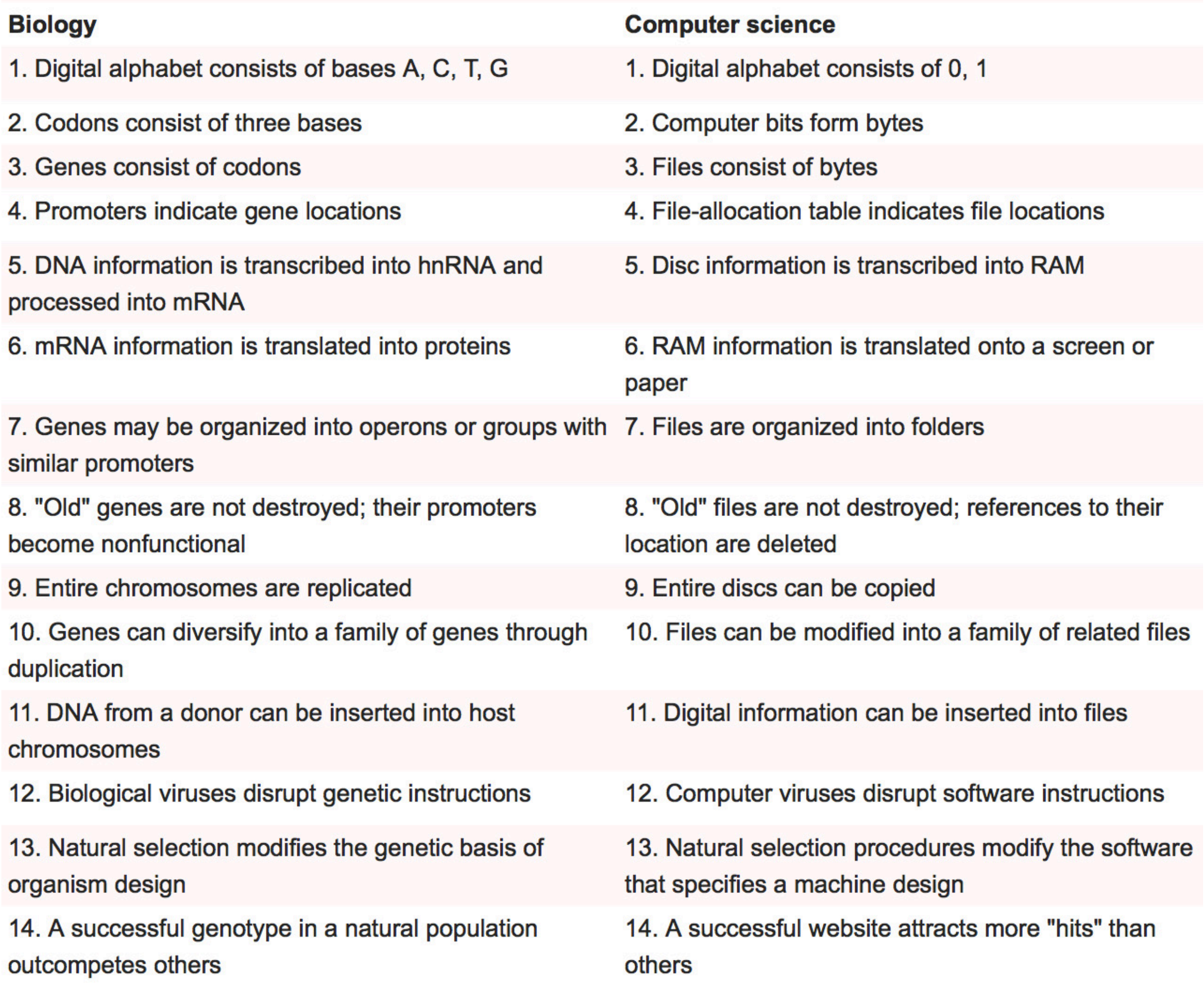
Cells: see
https://www.humancellatlas.org/

Pathways: see
https://www.genome.jp/kegg/pathway.html

https://www.rhea-db.org/

https://www.ncbi.nlm.nih.gov/

Cells versus Computers

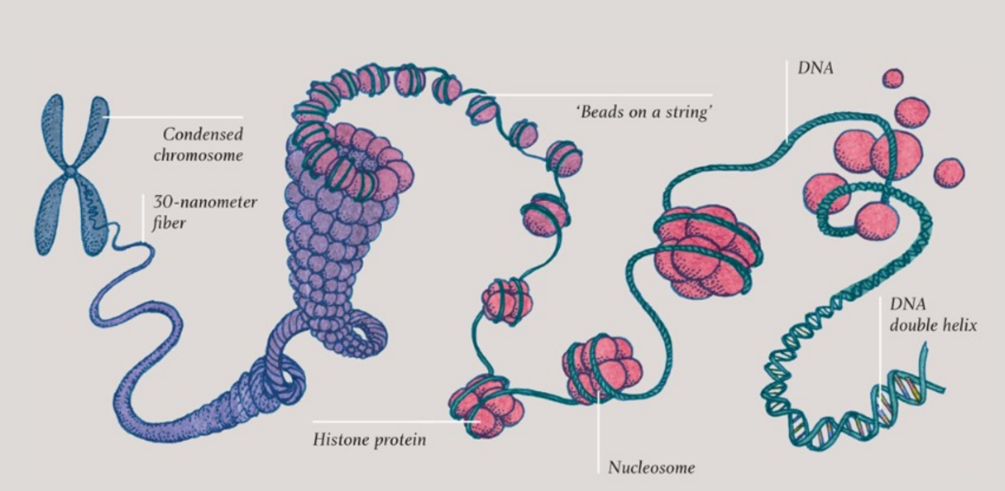


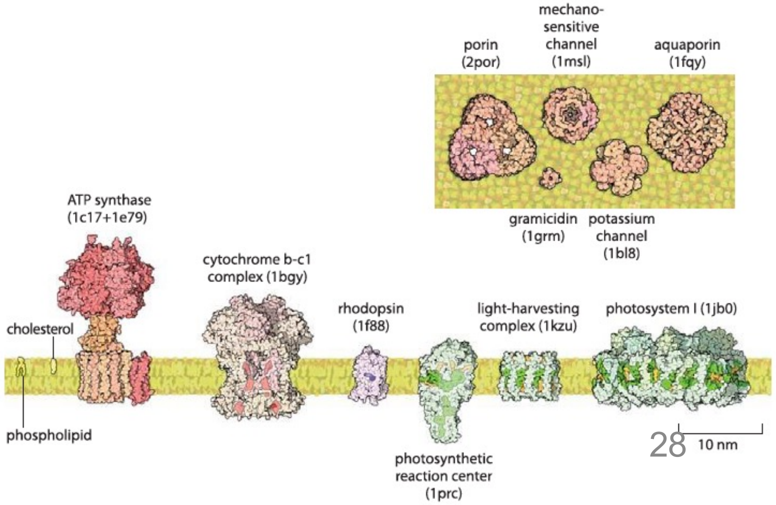
Biology	Computer science
1. Digital alphabet consists of bases A, C, T, G	1. Digital alphabet consists of 0, 1
2. Codons consist of three bases	2. Computer bits form bytes
3. Genes consist of codons	3. Files consist of bytes
4. Promoters indicate gene locations	4. File-allocation table indicates file 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

Challenges: quantify information in a cell; building computers inspired by cells information flow

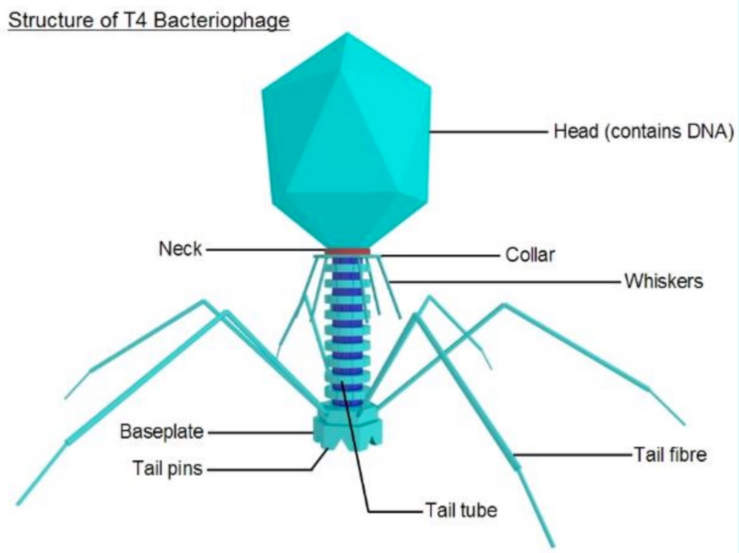
Nature is programmed for self-assemble;
Bioinformatics is needed to identify the key elements

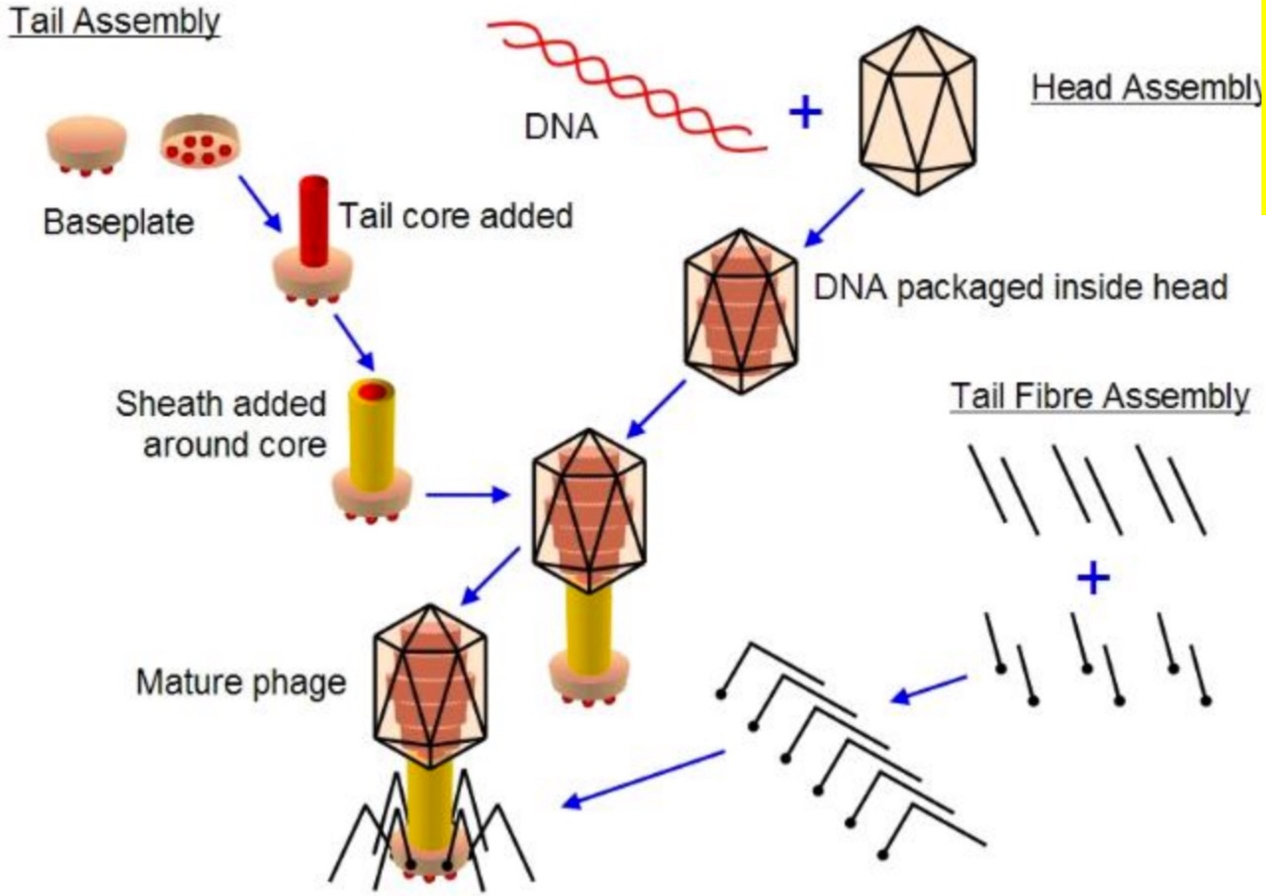
- DNA, RNA and proteins can:
- Organise themselves to self assemble different types of devices (mechanisms such rotors, motors) or structures with different shapes across time and space scales.
- Organise other types of molecules such as lipids, sugars and artificial ones.
- Organise large set of reactions (such as metabolic networks) and Execute different kinetics
- Self-Assemble control devices





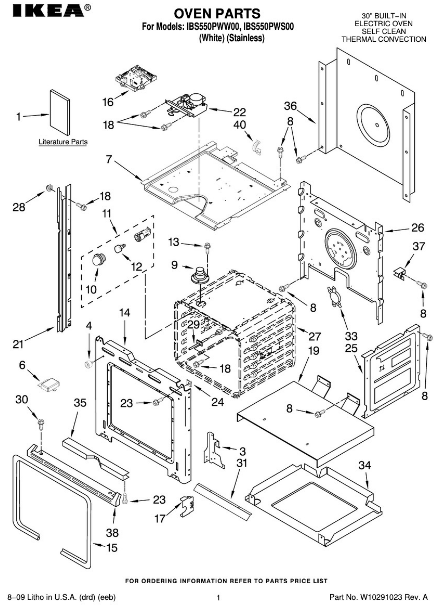
Nature is programmed for self assembly



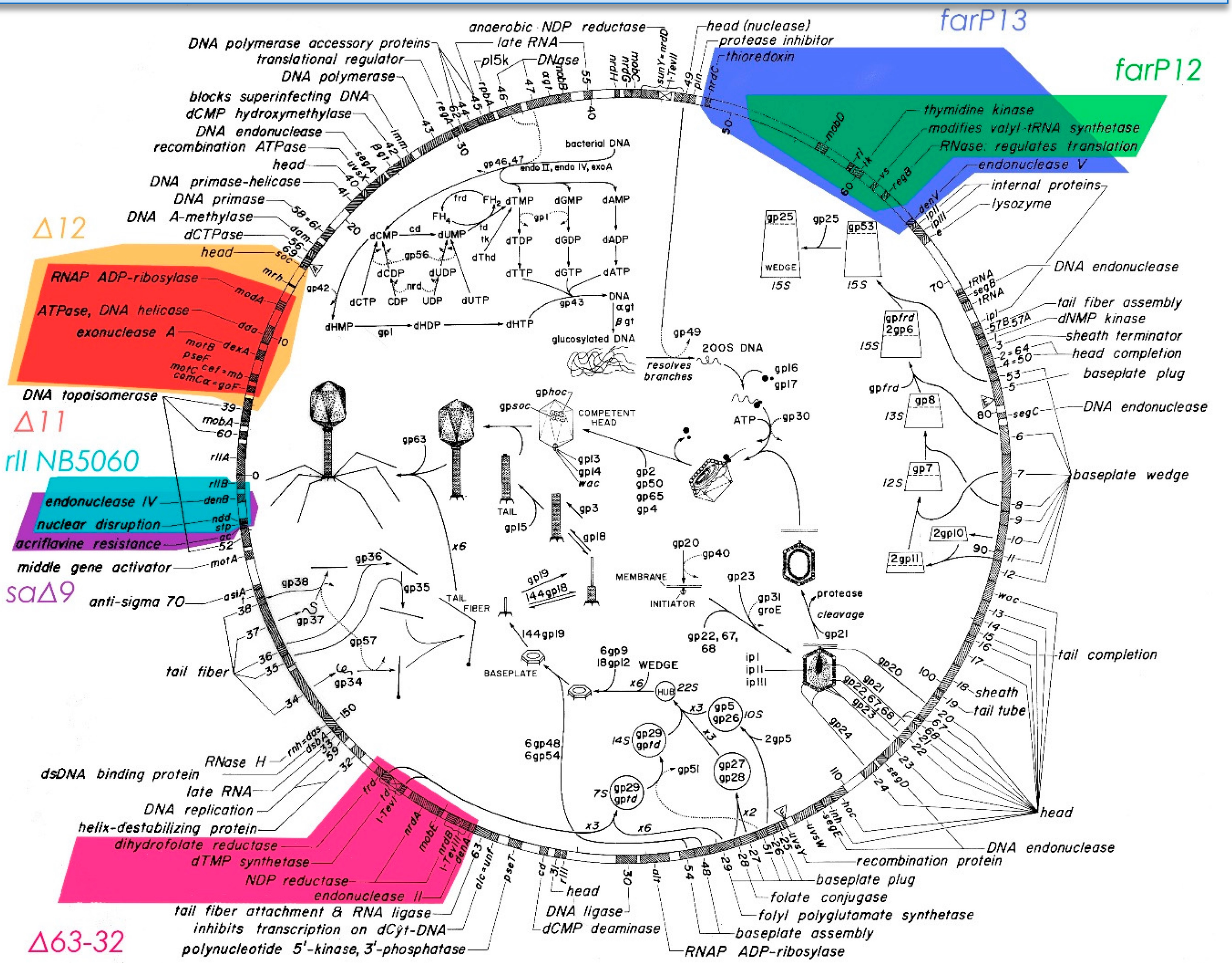


Size: 24 to 200 nanometers they’re 10 to 100 times smaller than the average bacterium, much too small to see with an ordinary light microscope.

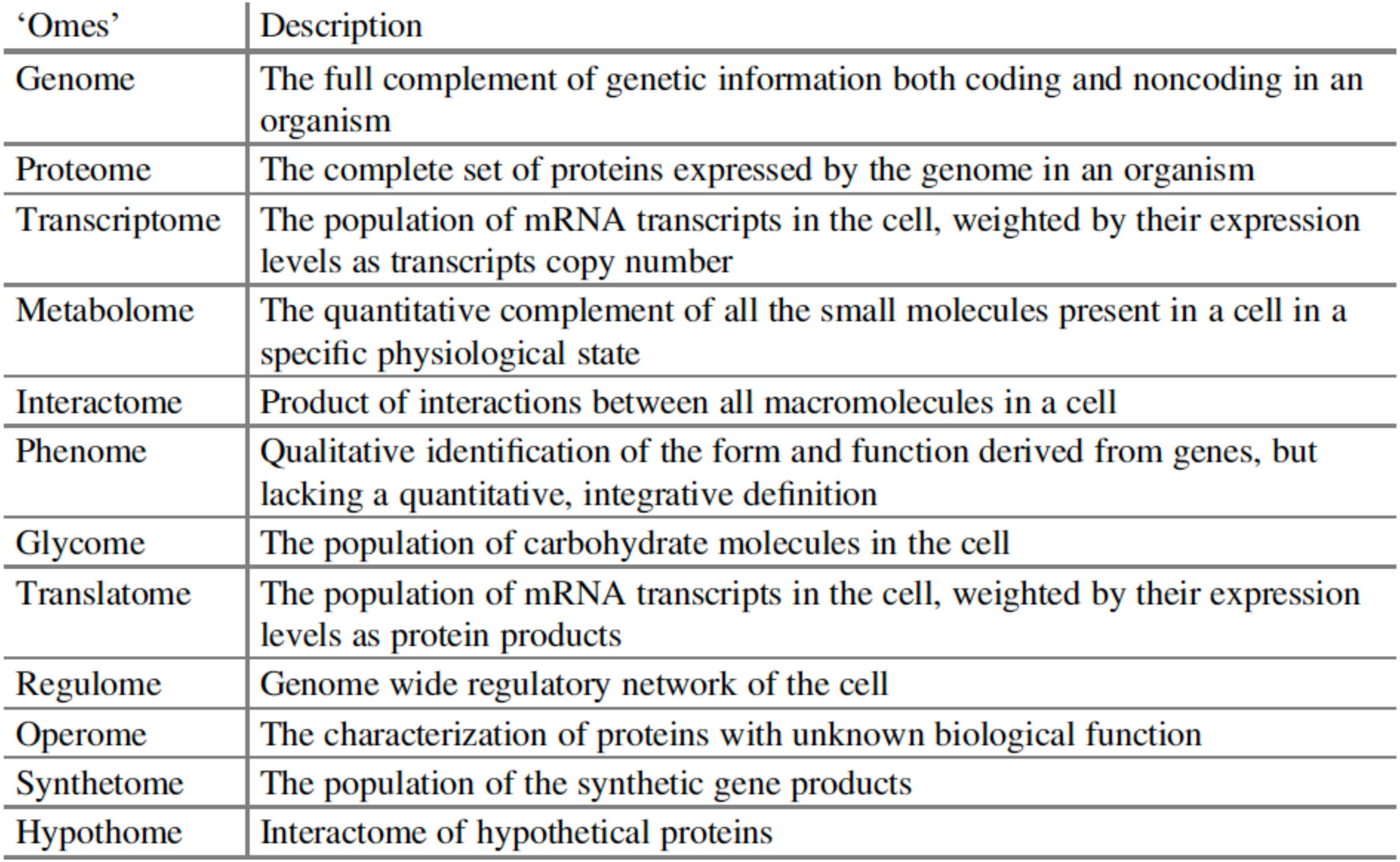
We absorb about 30 billion phages into our bodies every day. They form an integral part of our microbial ecosystem.



The genome contains both the instructions for assembly and for the parts and it is shipped with the virus



What is in a name/Different layers of information



'Omes'	Description
Genome	The full complement of genetic information both coding and noncoding in an organism
Proteome	The complete set of proteins expressed by the genome in an organism
Transcriptome	The population of mRNA transcripts in the cell, weighted by their expression levels as transcripts copy number
Metabolome	The quantitative complement of all the small molecules present in a cell in a specific physiological state
Interactome	Product of interactions between all macromolecules in a cell
Phenome	Qualitative identification of the form and function derived from genes, but lacking a quantitative, integrative definition
Glycome	The population of carbohydrate molecules in the cell
Translatome	The population of mRNA transcripts in the cell, weighted by their expression levels as protein products
Regulome	Genome wide regulatory network of the cell
Operome	The characterization of proteins with unknown biological function
Synthetome	The population of the synthetic gene products
Hypothome	Interactome of hypothetical proteins

Single cell/ single tissue repositories

Tabula Sapiens

https://tabula-sapiens-portal.ds.czbiohub.org/

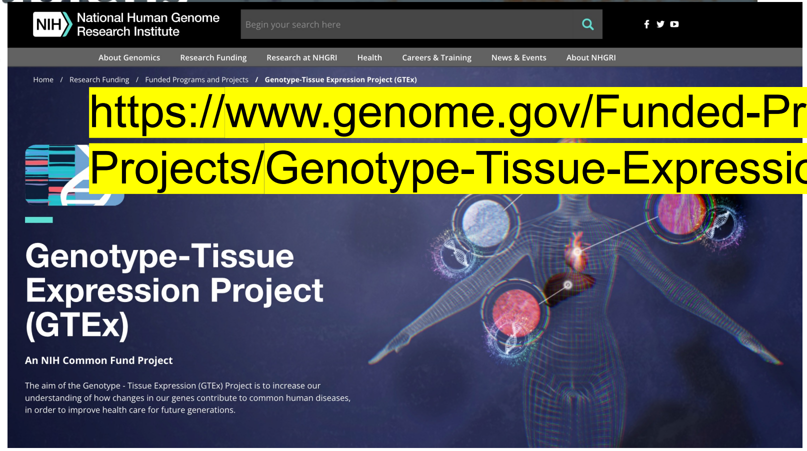
Tabula Sapiens

HUMAN CELL ATLAS

About HCA ▾ COVID-19 ▾ Research News ▾ Publications Data ▾ Resources ▾

https://www.humancellatlas.org

HUMAN CELL ATLAS



NIH National Human Genome Research Institute

Begin your search here

About Genomics | Research Funding | Research at NHGRI | Health | Careers & Training | News & Events | About NHGRI

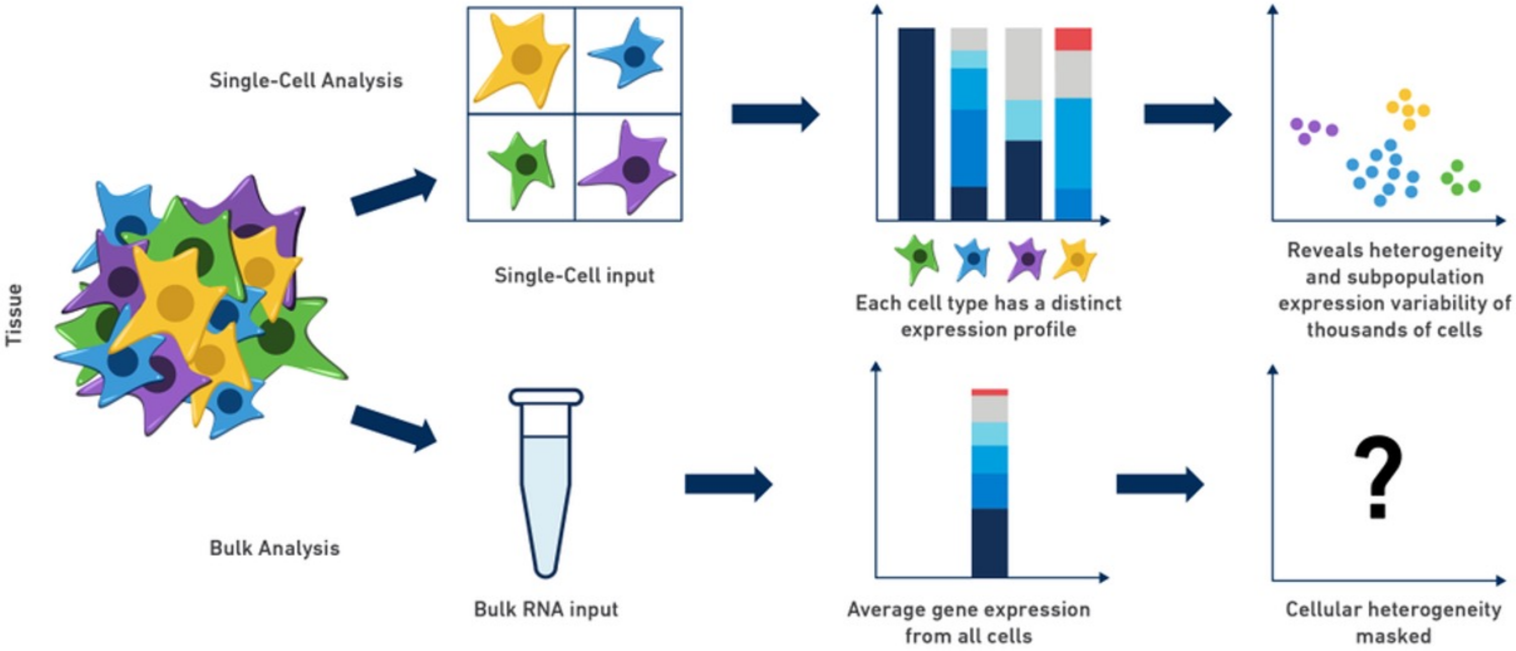
Home / Research Funding / Funded Programs and Projects / Genotype-Tissue Expression Project (GTEx)

https://www.genome.gov/Funded-Programs-Projects/Genotype-Tissue-Expression-Project

Genotype-Tissue Expression Project (GTEx)

An NIH Common Fund Project

The aim of the Genotype - Tissue Expression (GTEx) Project is to increase our understanding of how changes in our genes contribute to common human diseases, in order to improve health care for future generations.

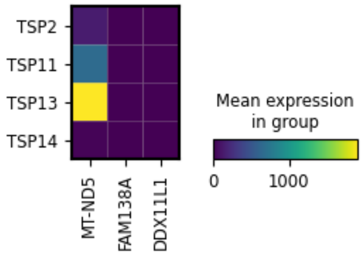


Single-Cell Analysis
Tissue
Single-Cell input
Each cell type has a distinct expression profile
Reveals heterogeneity and subpopulation expression variability of thousands of cells
Bulk Analysis
Bulk RNA input
Average gene expression from all cells
?
Cellular heterogeneity masked

The figure shows the differences between single cell and classic microarray analysis

Extracting Single cell data from h5ad file

```
import sys
import argparse
import scanpy

from matplotlib import pyplot as plt

args_parser = argparse.ArgumentParser(description='Get expression matrix for a given h5ad file from Tabula Sapiens')
args_parser.add_argument('--path', type=str, help='Path of the h5ad file')
args_parser.add_argument('--list_of_genes', nargs="+", help='List of genes expected for the expression matrix. (default: ALL, warning this may take a lot of time) ')
args_parser.add_argument('--output', type=str, help='Path of the output png.')

if len(sys.argv)==1:
    args_parser.print_help(sys.stderr)
    sys.exit(1)

args = args_parser.parse_args()

df = scanpy.read_h5ad(args.path)

fig = scanpy.pl.matrixplot(df, args.list_of_genes, groupby='donor', return_fig=True)

fig.savefig(args.output)
```



Reference for this section

A nice free book : cell biology by the numbers

http://book.bionumbers.org/

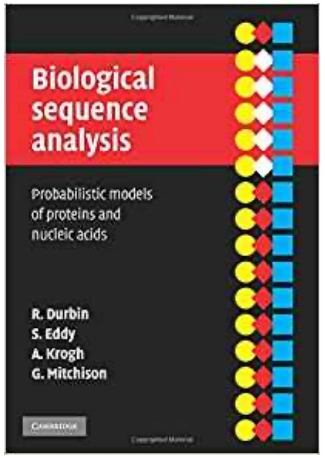
Others:

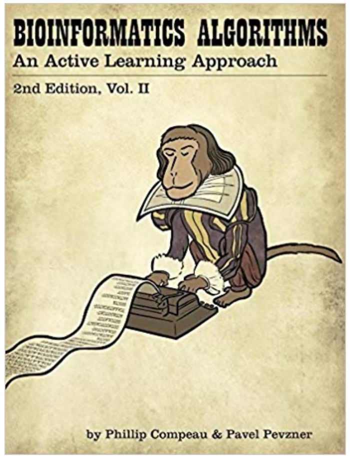
https://www.cs.helsinki.fi/group/genetics/Genetics_for_CS_March_04.pdf

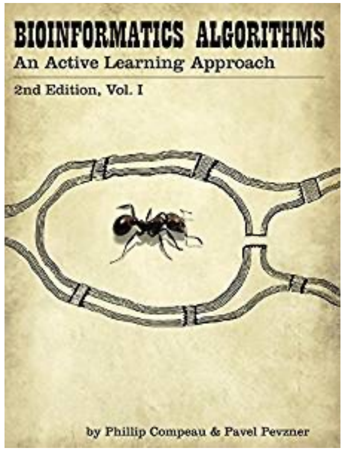
http://tandy.cs.illinois.edu/Hunter_MolecularBiology.pdf

Biology and Computers: A lesson in what is possible

https://ethw.org/; https://www.wehi.edu.au/wehi-tv/; good resources at https://www.ncbi.nlm.nih.gov/home/tutorials/ and ebi.ac.uk

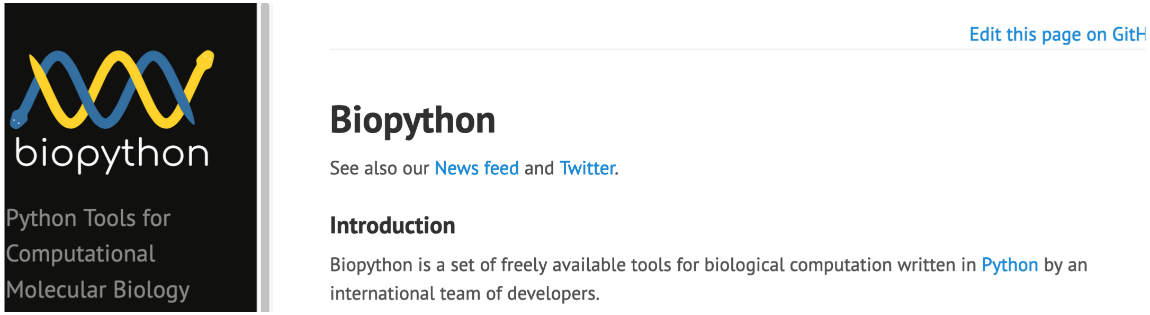

Biological sequence analysis
Probabilistic models of proteins and nucleic acids
R. Durbin
S. Eddy
A. Krogh
G. Mitchison
CAMBRIDGE


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. II
by Phillip Compeau & Pavel Pevzner


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. I
by Phillip Compeau & Pavel Pevzner

Check the chapters corresponding to the slides

If you want to code you may have a look at these libraries:



biopython

Python Tools for Computational Molecular Biology

Edit this page on GitH

Biopython

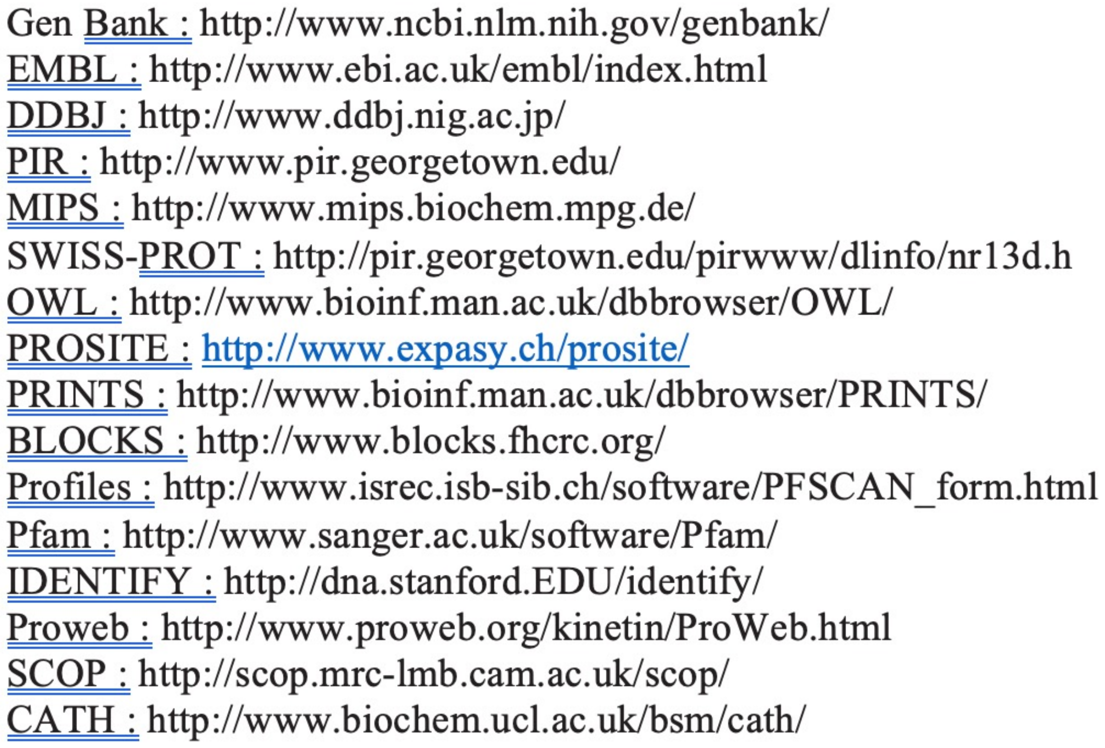
See also our News feed and Twitter.

Introduction

Biopython is a set of freely available tools for biological computation written in Python by an international team of developers.

BioJava – www.biojava.org
BioPerl – www.bioperl.org
BioPython – www.biopython.org
BioCorba – www.biocorba.org
BioRuby – www.bioruby.org
BioHaskell – www.biohaskell.org
C++ www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/
http://www.bioinformatics.org/biococoa/wiki/pmwiki.php

If you want data you may have a look at these repositories:



Gen Bank : http://www.ncbi.nlm.nih.gov/genbank/
EMBL : http://www.ebi.ac.uk/embl/index.html
DDBJ : http://www.ddbj.nig.ac.jp/
PIR : http://www.pir.georgetown.edu/
MIPS : http://www.mips.biochem.mpg.de/
SWISS-PROT : http://pir.georgetown.edu/pirwww/dlinfo/nr13d.h
OWL : http://www.bioinf.man.ac.uk/dbbrowser/OWL/
PROSITE : http://www.expasy.ch/prosite/
PRINTS : http://www.bioinf.man.ac.uk/dbbrowser/PRINTS/
BLOCKS : http://www.blocks.fhcrc.org/
Profiles : http://www.isrec.isb-sib.ch/software/PFSCAN_form.html
Pfam : http://www.sanger.ac.uk/software/Pfam/
IDENTIFY : http://dna.stanford.EDU/identify/
Proweb : http://www.proweb.org/kinetin/ProWeb.html
SCOP : http://scop.mrc-lmb.cam.ac.uk/scop/
CATH : http://www.biochem.ucl.ac.uk/bsm/cath/

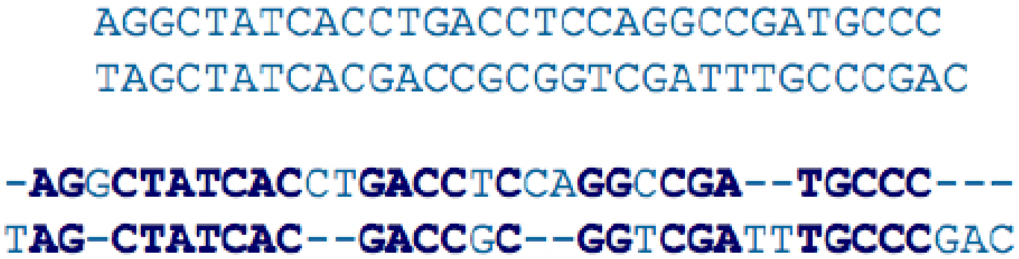
Section 2

Measuring sequence similarity through the use of alignment algorithms

- Algorithm: Longest Common Subsequence
- Algorithm: Waterman
- Algorithm: Hirshberg
- Algorithm: Four Russians
- Algorithm: Nussinov

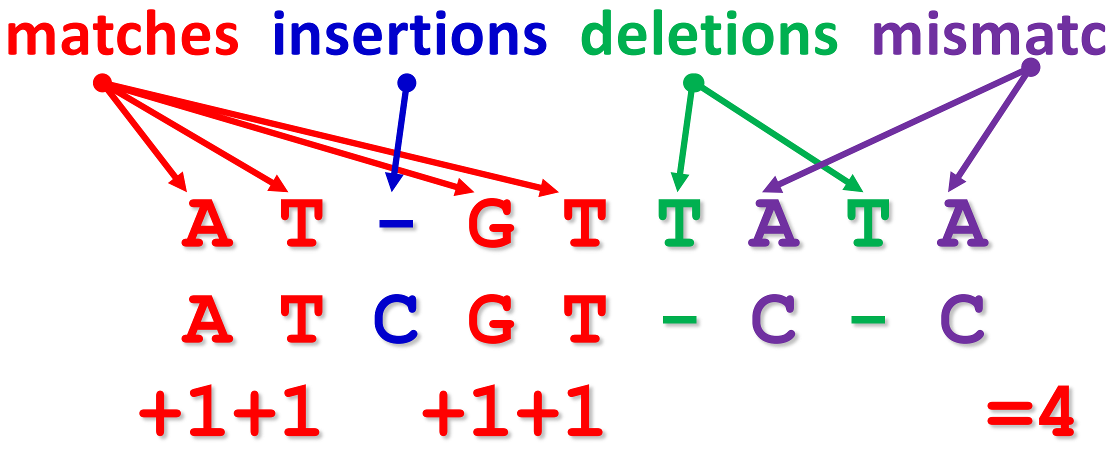
What is sequence alignment

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

What Is the Sequence Alignment?



matches insertions deletions mismatches

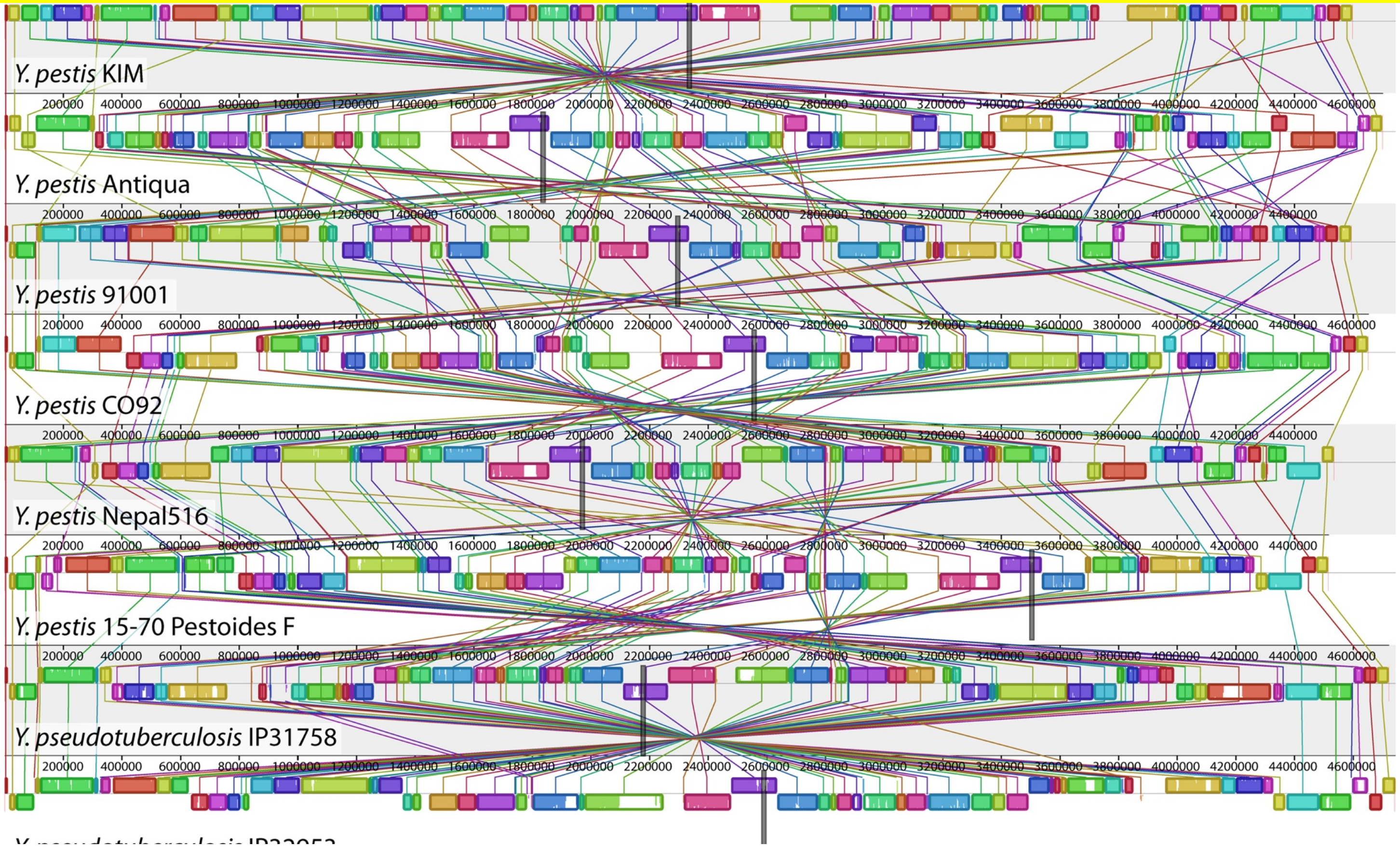
```
A T - G T T A T A
A T C G T - C - C
+1+1  +1+1      =4
```

Alignment of two sequences is a two-row matrix:

1st row: symbols of the 1st sequence (in order) interspersed by “-”
2nd row: symbols of the 2nd sequence (in order) interspersed by “-”

Why biologists need algorithms to do the alignment

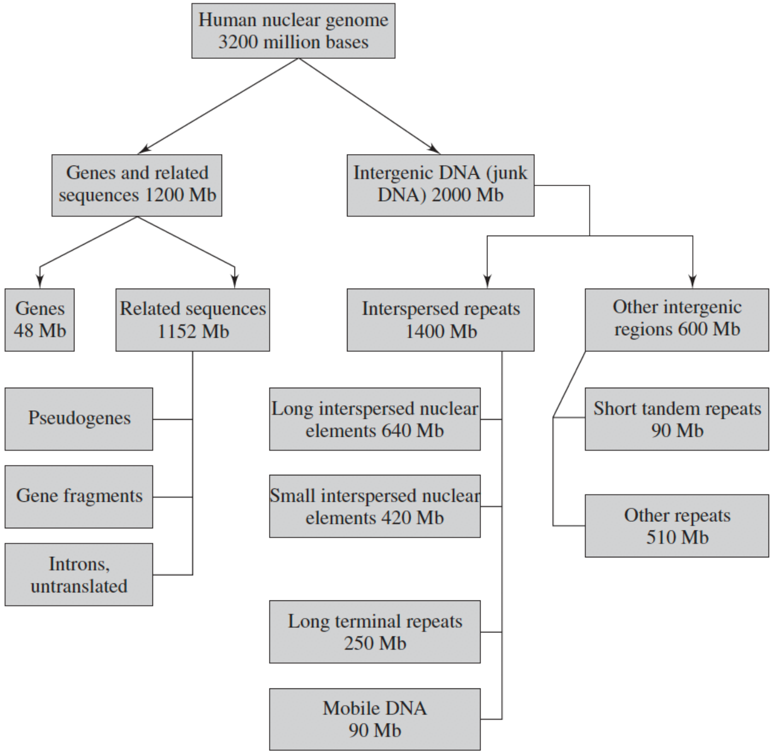
The sequence and structures of genes and proteins are conserved in nature. It is common to observe strong sequence similarity between a protein and its counterpart in another species that diverged hundreds of millions of years ago. Accordingly, the best method to identify the function of a new gene or protein is to find its sequence- related genes or proteins whose functions are already known.



sequence changes in the bacteria causing plague. Different regions of the genome of the bacterium could be represented with different colours. This makes easier to show changes (you can retrieve sequences at www.ebi.ac.uk).

Source: By Aaron E. Darling, István Miklós, Mark A. Ragan - Figure 1 from Darling AE, Miklós I, Ragan MA (2008). "Dynamics of Genome Rearrangement in Bacterial Populations". PLOS Genetics. DOI:10.1371/journal.pgen.1000128., CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=30550950

All genomes are littered with repeated sequences of different length (they form families) so alignment of large sequences is difficult



Human nuclear genome 3200 million bases

Genes and related sequences 1200 Mb

Intergenic DNA (junk DNA) 2000 Mb

Genes 48 Mb

Related sequences 1152 Mb

Interspersed repeats 1400 Mb

Other intergenic regions 600 Mb

Pseudogenes

Gene fragments

Introns, untranslated

Long interspersed nuclear elements 640 Mb

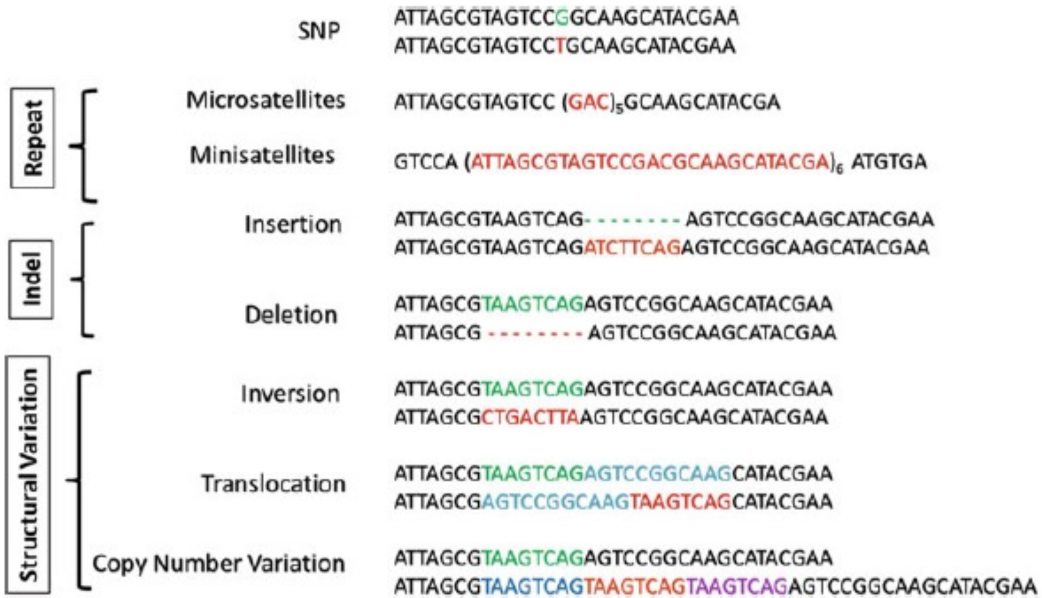
Small interspersed nuclear elements 420 Mb

Long terminal repeats 250 Mb

Mobile DNA 90 Mb

Short tandem repeats 90 Mb

Other repeats 510 Mb



SNP
ATTAGCGTAGTCCGGCAAGCATACGAA
ATTAGCGTAGTCCTGCAAGCATACGAA

Repeat

Microsatellites
ATTAGCGTAGTCC (GAC)$_5$GCAAGCATACGA

Minisatellites
GTCCA (ATTAGCGTAGTCCGACGCAAGCATACGA)$_6$ ATGTGA

Indel

Insertion
ATTAGCGTAAGTCAG- - - - - - - - AGTCCGGCAAGCATACGAA
ATTAGCGTAAGTCAGATCTTCAGAGTCCGGCAAGCATACGAA

Deletion
ATTAGCGTAAGTCAGAGTCCGGCAAGCATACGAA
ATTAGCG - - - - - - - - AGTCCGGCAAGCATACGAA

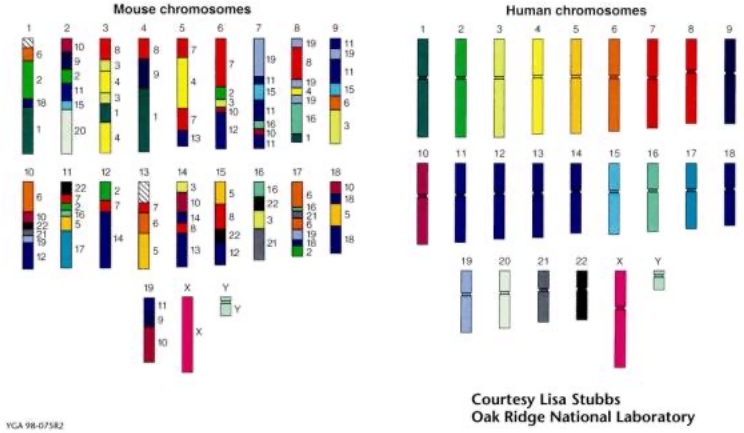
Structural Variation

Inversion
ATTAGCGTAAGTCAGAGTCCGGCAAGCATACGAA
ATTAGCGCTGACTTAAGTCCGGCAAGCATACGAA

Translocation
ATTAGCGTAAGTCAGAGTCCGGCAAGCATACGAA
ATTAGCGAGTCCGGCAAGTAAGTCAGCATACGAA

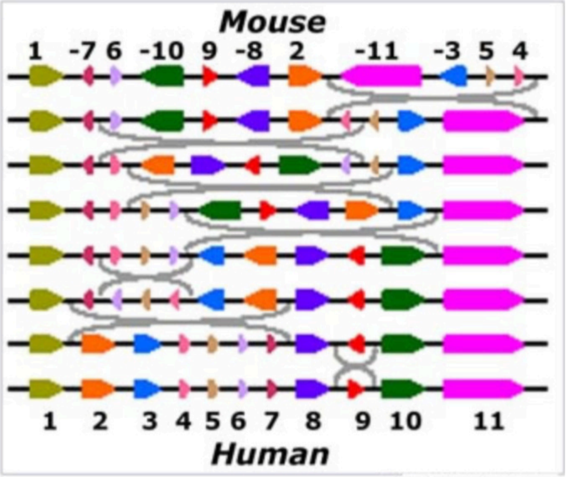
Copy Number Variation
ATTAGCGTAAGTCAGAGTCCGGCAAGCATACGAA
ATTAGCGTAAGTCAGTAAGTCAGTAAGTCAGAGTCCGGCAAGCATACGAA

Mouse and Human Genetic Similarities



Courtesy Lisa Stubbs
Oak Ridge National Laboratory

Left: mapping human chromosomes onto mouse chromosome reveal many similar regions (low resolution).
Right: a higher resolution, each region reveals many local rearrangements



©Genome Research

Longest Common Subsequence

A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

```
A T - G T T A T A
A T C G T - C - C
```

Matches in alignment of two sequences (**ATGT**) form their

Common Subsequence

Longest Common Subsequence Problem: Find a longest common subsequence of two strings.

- **Input:** Two strings.
- **Output:** A longest common subsequence of these strings.

Alignment: 2 row representation

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

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

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

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

4 matches

2 insertions

2 deletions

Longest Common Subsequence

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

- Given two sequences **v** = $v_1\ v_2 \ldots v_m$ and **w** = $w_1\ w_2 \ldots w_n$
- The LCS of **v** and **w** is a sequence of positions in

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

and a sequence of positions in

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

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

Longest Common Subsequence

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

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

Matches shown in red

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

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

Every common subsequence is a path in 2-D grid

Edit distance

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

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

```
v  =  ATATATAT
      | | |  | |  | | |
w  =  TATATATA
```

Just one shift

Make it all line up

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

```
v  =   -ATATATAT
         | | |  | |  | |
w  =   TATATATA-
```

Hamming distance:

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

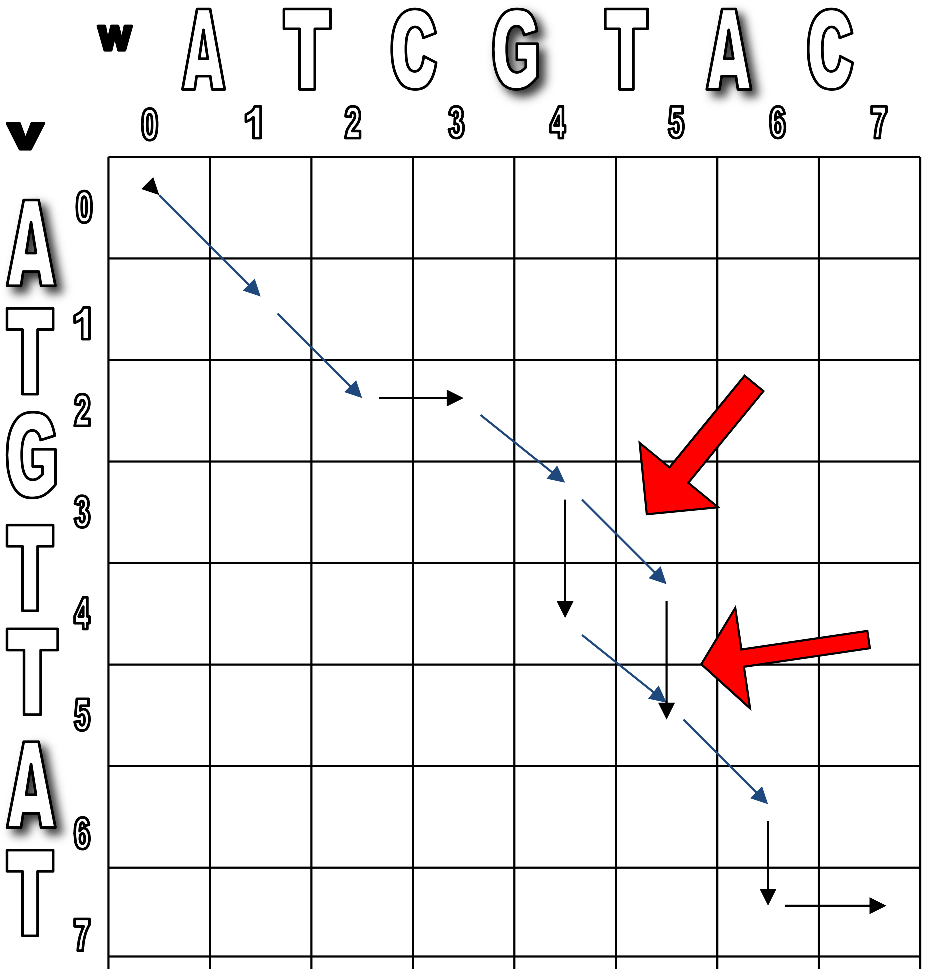
Computing Hamming distance is a trivial task

Edit distance:

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

Computing edit distance is a non-trivial task

Alignment as a Path in the Edit Graph



Old Alignment

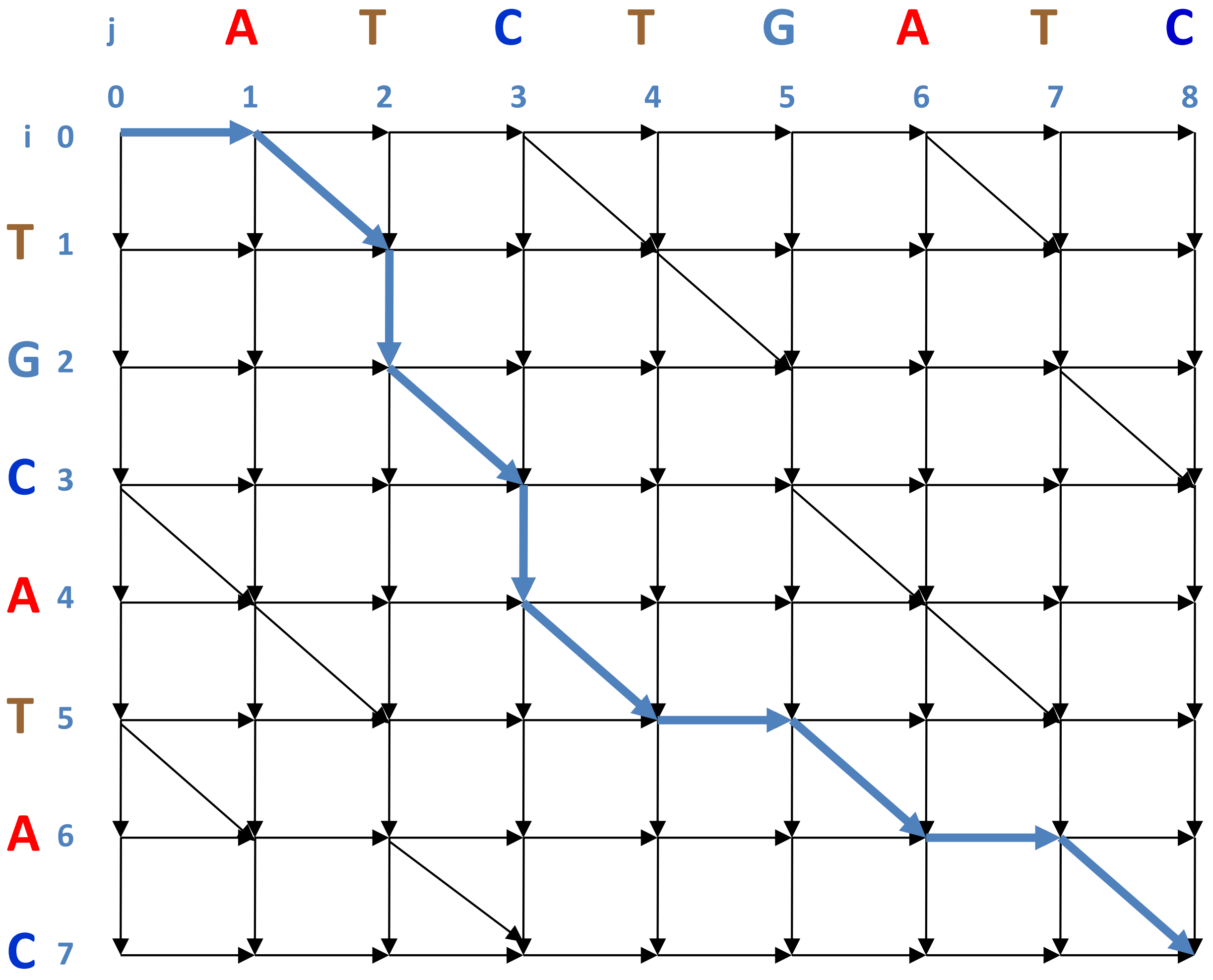
```
    0122345677
v=   AT_GTTAT_
w=   ATCGT_A_C
    0123455667
```

New Alignment

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

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

LCS Problem as - Edit Graph



Every path is a common subsequence.

Every diagonal edge adds an extra element to common subsequence

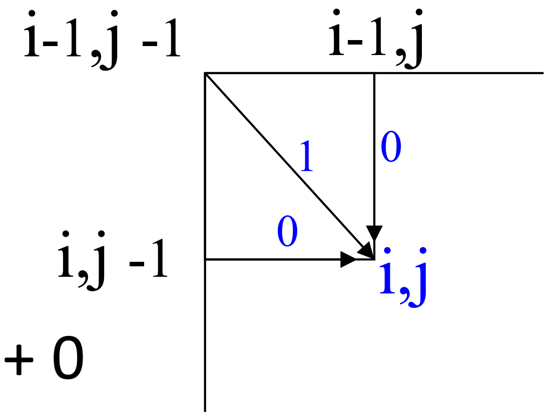
LCS Problem: Find a path with maximum number of diagonal edges

Computing LCS

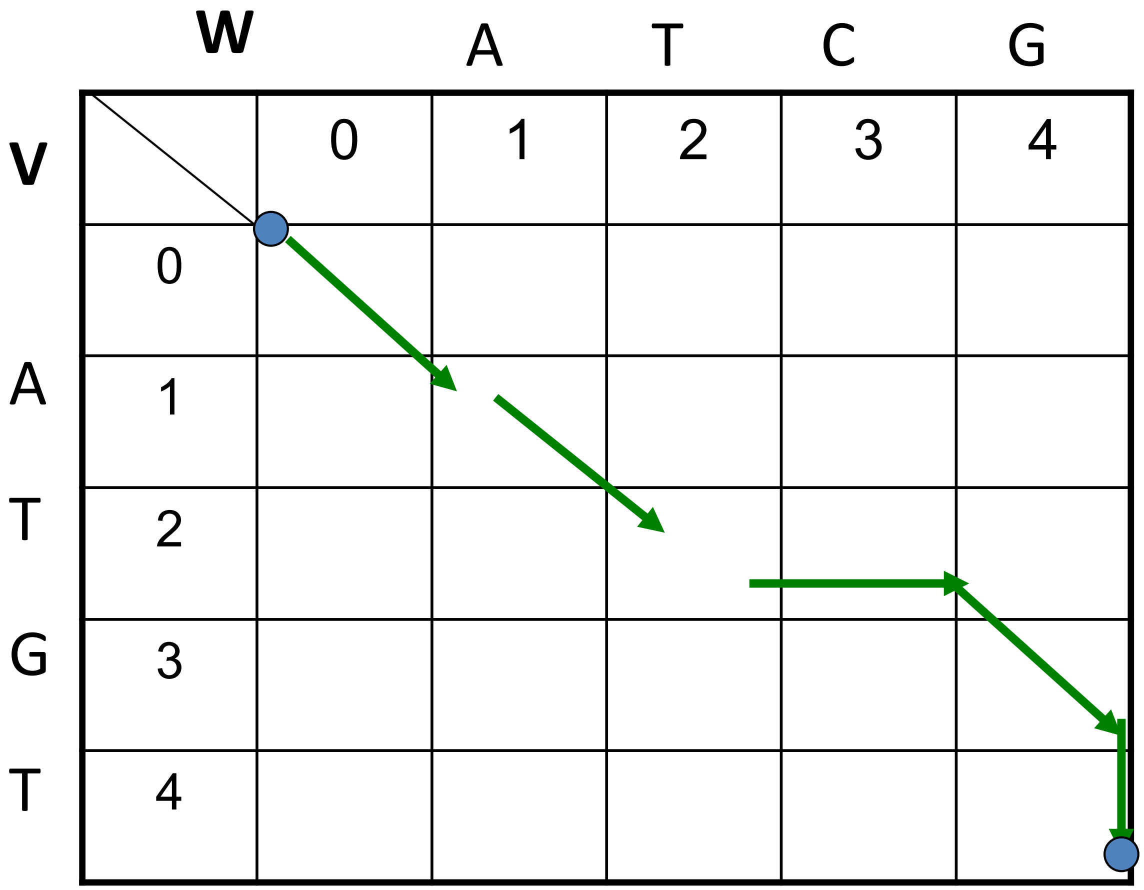
Let $\mathbf{v}_i$ = prefix of **v** of length i: $v_1 \ldots v_i$

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

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



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



Every Path in the Grid Corresponds to an Alignment

↘ ↘ → ↘ ↓

0 1 2 2 3 4

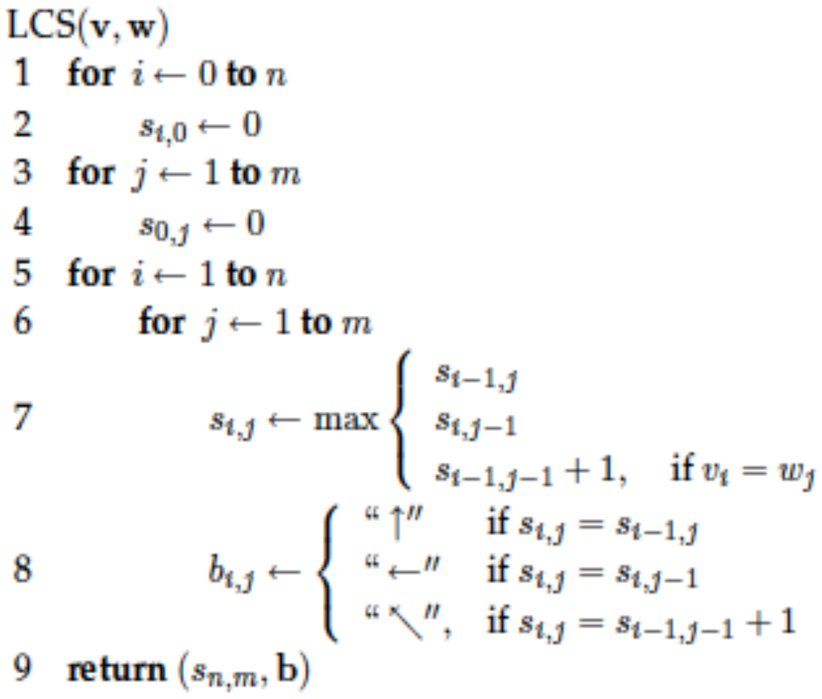
V = A T - G T

| | |

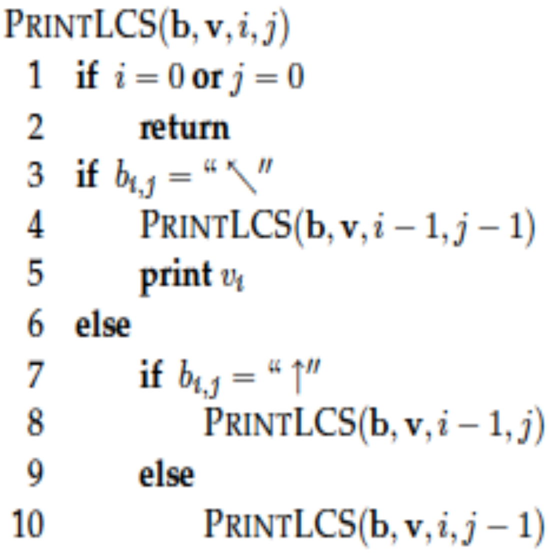
W= A T C G –

0 1 2 3 4 4

LCS pseudocode



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



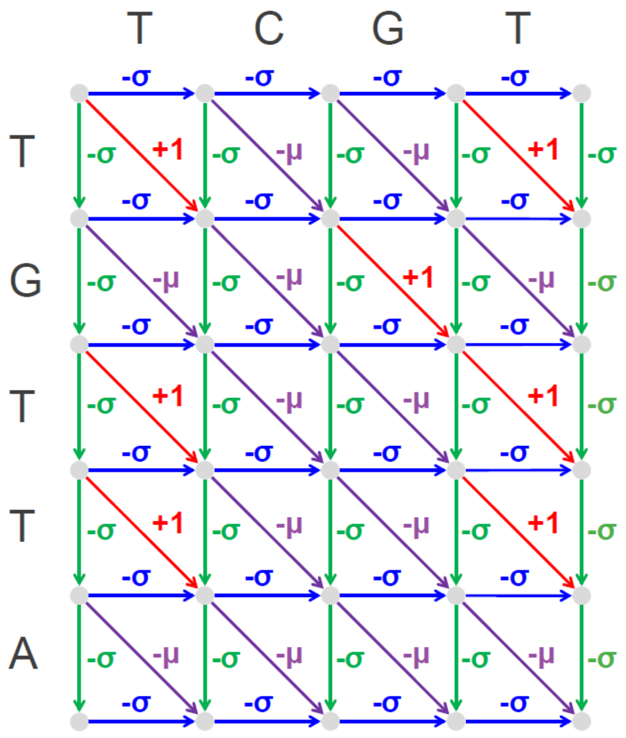
```
PRINTLCS(b, v, i, j)
if $i = 0$ or $j = 0$
        return
if $b_{i,j} = \text{"}\nwarrow\text{"}$
        PRINTLCS(b, v, $i - 1$, $j - 1$)
        print $v_i$
else
        if $b_{i,j} = \text{"}\uparrow\text{"}$
                PRINTLCS(b, v, $i - 1$, $j$)
        else
                PRINTLCS(b,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

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

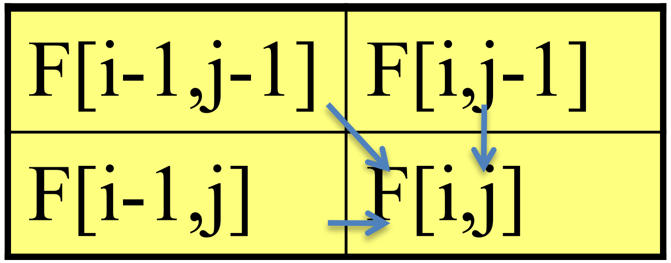
A speedup is the **Method of Four Russians,** to partition the matrix into small square blocks of size *t* × *t* for some parameter *t*, and to use a lookup table to perform the algorithm quickly within each block. The algorithm may be performed by operating on only $(n/t)^2$ blocks instead of on n^2 matrix cells, where n is the side length of the matrix. In order to keep the size of the lookup tables (and the time needed to initialize them) sufficiently small, t is typically chosen to be O(log n).

General Alignment Graph

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \\ s_{i-1,j-1} - \mu, \text{ if } v_i \neq w_j \end{cases}$$



Towards an algorithm to align biological sequences



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

Notice three possible cases:

1. x_i aligns to y_j

   $x_1 \ldots\ldots x_{i-1} \quad x_i$

   $y_1 \ldots\ldots y_{j-1} \quad y_j$

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

2. x_i aligns to a gap

   $x_1 \ldots\ldots x_{i-1} \quad x_i$

   $y_1 \ldots\ldots y_j \quad -$

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

3. y_j aligns to a gap

   $x_1 \ldots\ldots x_i \quad -$

   $y_1 \ldots\ldots y_{j-1} \quad y_j$

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

Dynamic Programming: 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-problem.It is a way of solving problems (involving recurrence relations) by storing partial results.

Alignment

- How do we know which case is correct?

F[i-1,j-1]	**F[i,j-1]**
F[i-1,j]	F[i,j]

<u>**Inductive assumption:**</u>

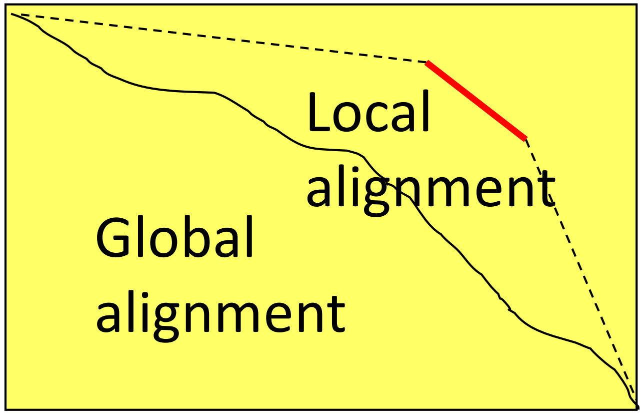
F(i, j-1), F(i-1, j), F(i-1, j-1) are optimal

Then,

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

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

Alignment



- Global Alignment: tries to find the longest path between vertices (0,0) and (n,m) in the edit graph.

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

- Local Alignment—better alignment to find highly conserved segments; The Local Alignment Problem tries to find the longest path among paths between arbitrary vertices (i,j) and (i’, j’) in the edit graph

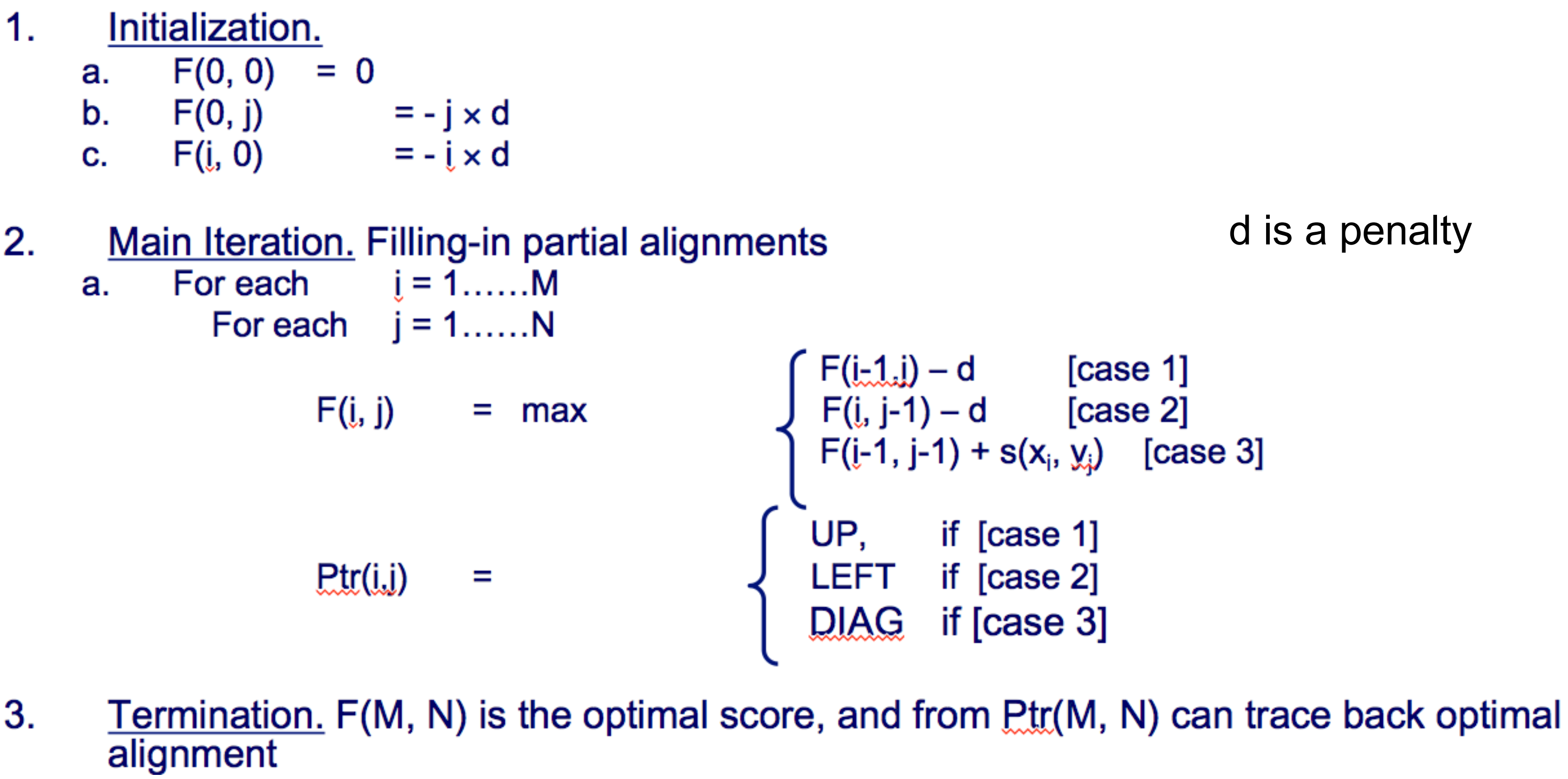
```
                  tccCAGTTATGTCAGggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

Global Alignment

Global Alignment Problem: Find the highest-scoring alignment between two strings by using a scoring matrix.

- **Input:** Strings *v* and *w* as well as a matrix ***score***.
- **Output:** An alignment of *v* and *w* whose alignment score (as defined by the scoring matrix ***score***) is maximal among all possible alignments of *v* and *w*.

The Needleman-Wunsch Algorithm (Global alignment)



1. Initialization.
   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
   a. For each $i = 1 \ldots\ldots M$
      For each $j = 1 \ldots\ldots N$

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

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

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

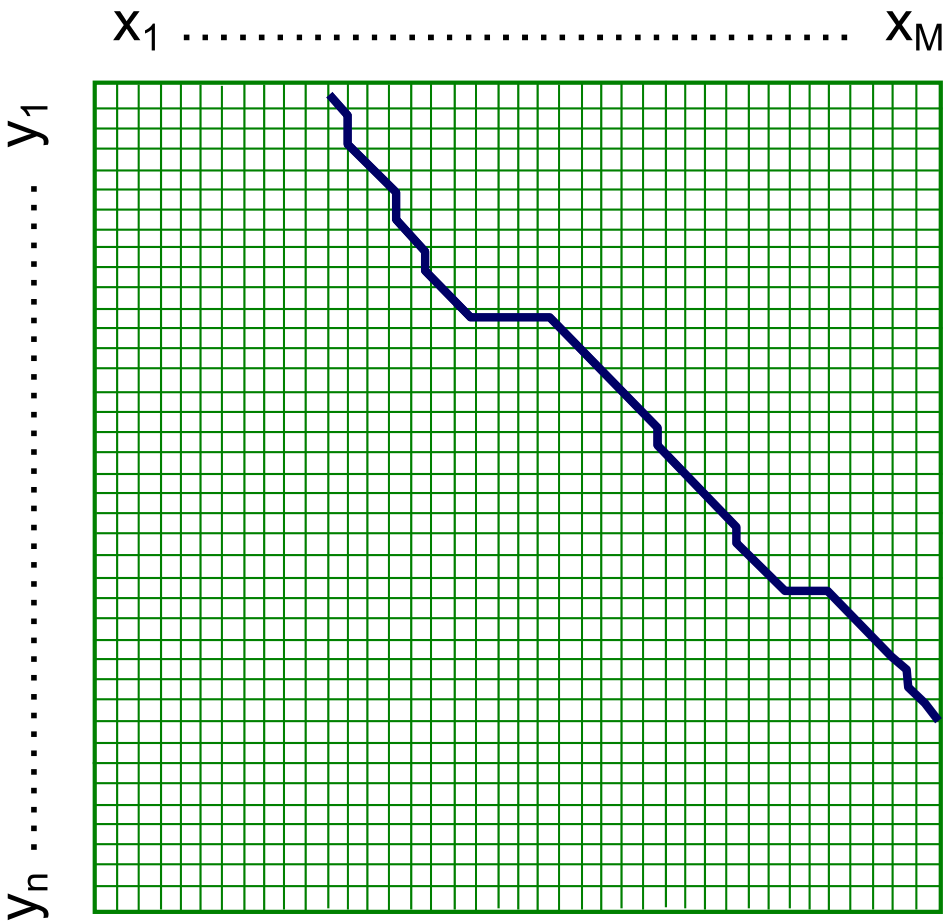
d is a penalty

Complexity: Space: O(mn); Time: O(mn)
Filling the matrix O(mn)
Backtrace O(m+n)

The Overlap Detection variant

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

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



Changes:

1. Initialization

   For all i, j,

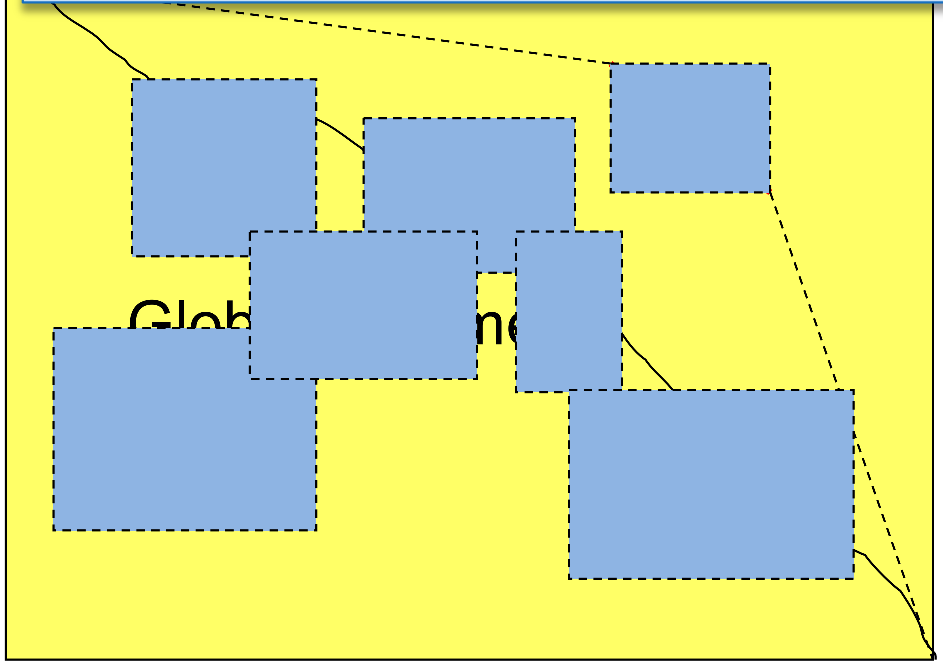
   $F(i, 0) = 0$

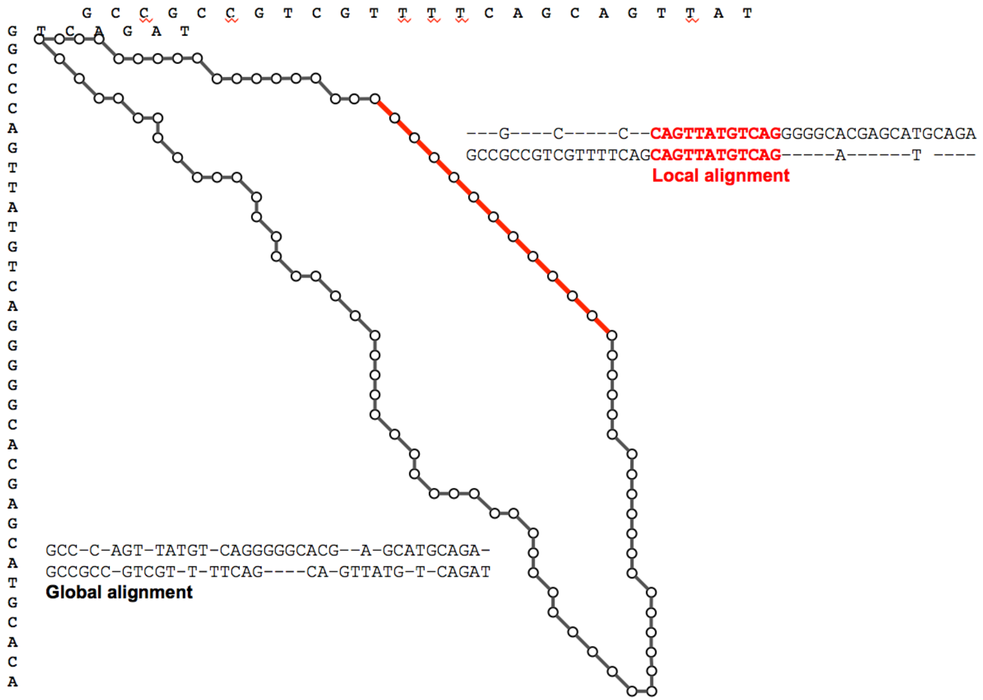
   $F(0, j) = 0$

2. Termination

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

Local Alignment= Global Alignment in a subrectangle




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

```
---G----C-----C--CAGTTATGTCAGGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTCAG-----A------T ----
```

Local alignment

```
GCC-C-AGT-TATGT-CAGGGGGCACG--A-GCATGCAGA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG-T-CAGAT
```

Global alignment

Local Alignment Problem

Local Alignment Problem: Find the highest-scoring local alignment between two strings.

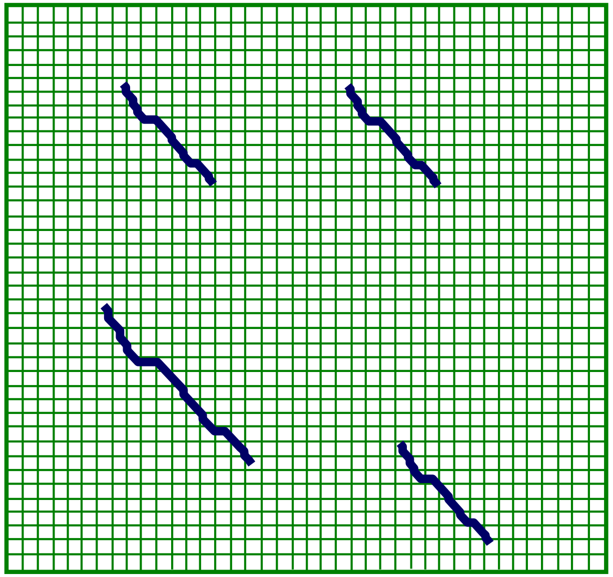
- **Input:** Strings *v* and *w* as well as a matrix ***score****.*

- **Output:** Substrings of *v* and *w* whose global alignment (as defined by the matrix ***score***), is maximal among all global alignments of all substrings of *v* and *w*.

The local alignment: Smith-Waterman algorithm

T.F. Smith, M.S.Waterman, Identification of common molecular subsequences, J Mol Biol vol 147,195-197, 1981.

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



e.g. x = aaaacc**cccggg**g

y = **cccggg**aaccaacc

Initialization: F(0,0)=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

David Waterman

Which Alignment is Better?

- Alignment 1: score = 22 (matches) - 20 (indels)=2.

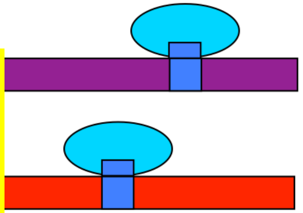
```
GCC-C-AGT--TATGT-CAGGGGGCACG--A-GCATGCAGA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT
```

- Alignment 2: score = 17 (matches) - 30 (indels)=-13.

```
---G----C------C--CAGTTATGTCAGGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTCAG-----A------T-----
```

local alignment

Biologists are interested in the local alignment to detect a region common to two genes which could suggest the same regulatory control: Perhaps the two similar regions are bound to the same proteins



Scoring Gaps

- We previously assigned a fixed penalty *σ* to each indel.
- However, this fixed penalty may be too severe for a series of 100 consecutive indels.
- A series of *k* indels often represents a single evolutionary event (**gap**) rather than *k* events:

two gaps	**GATCCAG**	**GATCCAG**	a single gap
(lower score)	**GA-C-AG**	**GA--CAG**	(higher score)

Mismatches and Indel Penalties

#matches − μ · #mismatches − σ · #indels

```
A T - G T T A T A
A T C G T - C - C
```

+1+1-2+1+1-2-3-2-3=-7

	A	C	G	T	−
A	+1	$-\mu$	$-\mu$	$-\mu$	$-\sigma$
C	$-\mu$	+1	$-\mu$	$-\mu$	$-\sigma$
G	$-\mu$	$-\mu$	+1	$-\mu$	$-\sigma$
T	$-\mu$	$-\mu$	$-\mu$	+1	$-\sigma$
−	$-\sigma$	$-\sigma$	$-\sigma$	$-\sigma$	

Scoring matrix

	A	C	G	T	−
A	+1	−3	−5	−1	-3
C	−4	+1	−3	−2	-3
G	−9	−7	+1	−1	-3
T	−3	−5	−8	+1	-4
−	-4	-2	-2	-1	

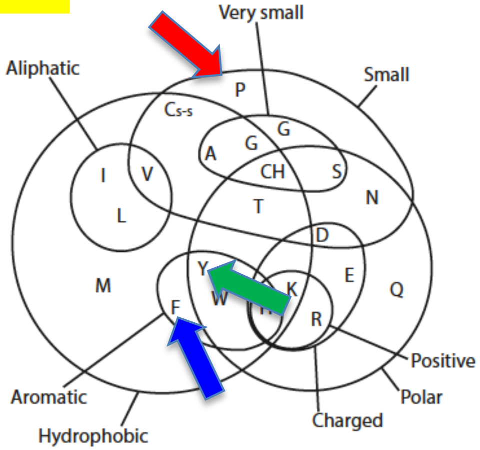
Even more general scoring matrix

Scoring matrices to compare amino acid sequences: PAM250 is a log odds matrix

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C Cys	12																			
S Ser	0	2																		
T Thr	-2	1	3																	
P Pro	-3	1	0	6																
A Ala	-2	1	1	1	2															
G Gly	-3	1	0	-1	1	5														
N Asn	-4	1	0	-1	0	0	2													
D Asp	-5	0	0	-1	0	1	2	4												
E Glu	-5	0	0	-1	0	0	1	3	4											
Q Gln	-5	-1	-1	0	0	-1	1	2	2	4										
H His	-3	-1	-1	0	-1	-2	2	1	1	3	6									
R Arg	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6								
K Lys	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5							
M Met	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6						
I Ile	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5					
L Leu	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2					
V Val	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4			
F Phe	-4	-3	-3	-5	-5	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9		
Y Tyr	0	-3	-3	-5	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	7	10	
W Trp	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17

Positive exchange values denote mutations that are more likely than randomly expected, while negative numbers correspond to avoided mutations compared to the randomly expected situation

Margaret Dayhoff



example: Y (Tyr) often mutates into F (score +7) but rarely mutates into P (score -5)

Scoring matrices

A scoring matrix contains values proportional to the probability that amino acid i mutates into amino acid j for all pairs of amino acids.

Scoring matrices are constructed by assembling a large and diverse sample of verified pairwise alignments (or multiple sequence alignments) of amino acids. Scoring matrices should reflect the true probabilities of mutations occurring through a period of evolution.

PAM (point accepted mutations) matrices are based on global alignments of closely related proteins. The PAM1 is the matrix calculated from comparisons of sequences with no more than 1% divergence. At an evolutionary interval of PAM1, one change has occurred over a length of 100 amino acids.

Other PAM matrices are extrapolated from PAM1. For PAM250, 250 changes have occurred for two proteins over a length of 100 amino acids. All the PAM data come from closely related proteins (>85% amino acid identity).

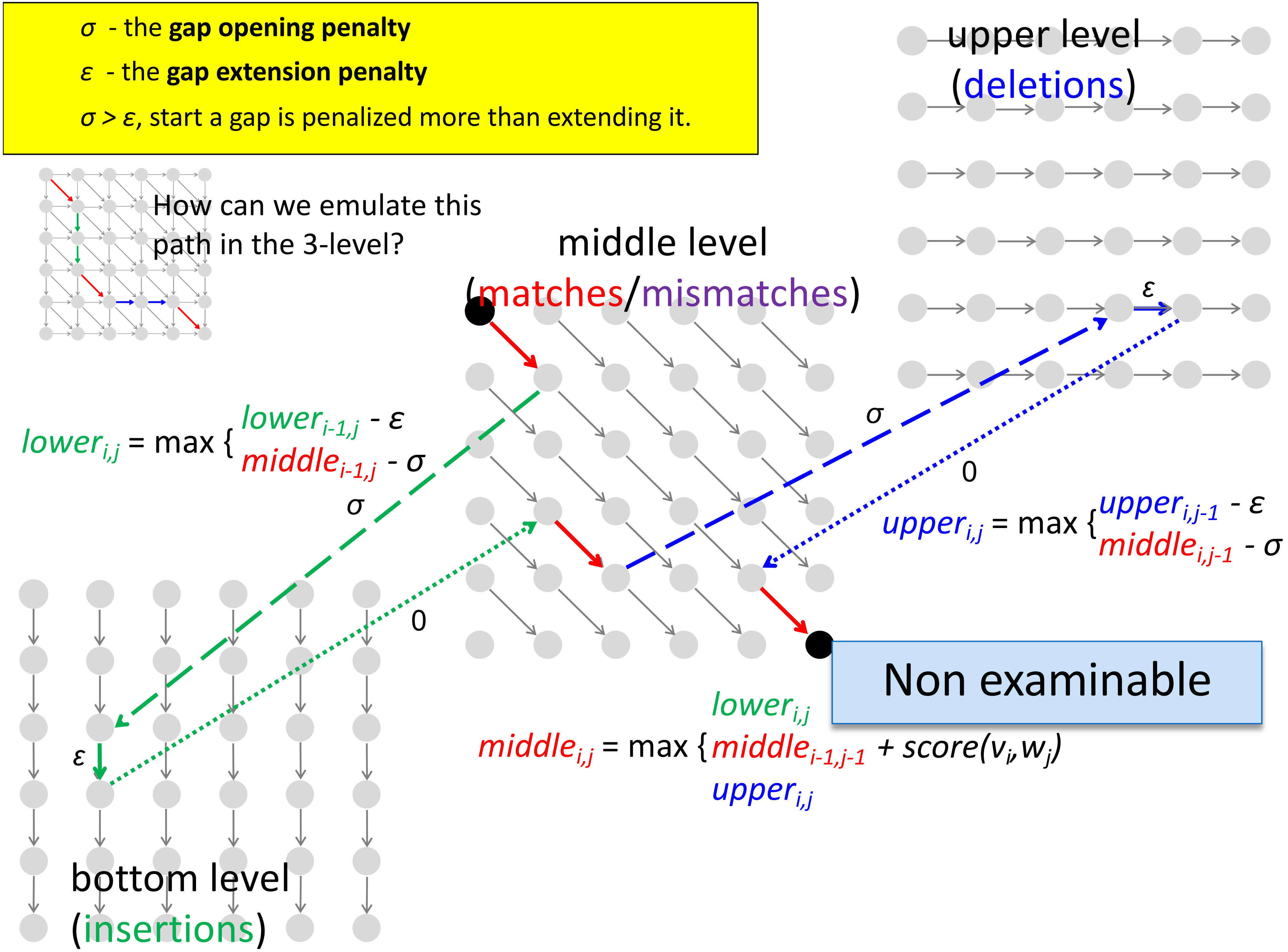
A log odds matrix is the logarithmic form of the relatedness odds matrix.
S_{ij} is the score for aligning any two residues in a pairwise alignment.
M_{ij} is of the observed frequency of substitutions for each pair of amino acids.
f_i is the probability of amino acid residue i occurring in the second sequence by chance.

$S_{ij}=10 \log (M_{ij}/f_j)$.

σ - the **gap opening penalty**

ε - the **gap extension penalty**

$\sigma > \varepsilon$, start a gap is penalized more than extending it.



How can we emulate this path in the 3-level?

$lower_{i,j} = \max \begin{cases} lower_{i-1,j} - \varepsilon \\ middle_{i-1,j} - \sigma \end{cases}$

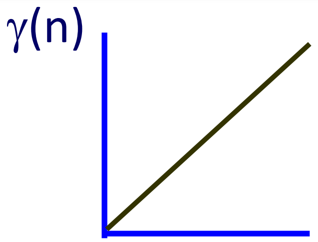
$upper_{i,j} = \max \begin{cases} upper_{i,j-1} - \varepsilon \\ middle_{i,j-1} - \sigma \end{cases}$

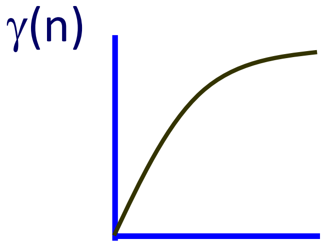
$middle_{i,j} = \max \begin{cases} lower_{i,j} \\ middle_{i-1,j-1} + score(v_i, w_j) \\ upper_{i,j} \end{cases}$

Non examinable

Models of gaps; Alignment with gaps

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





Initialization: same

Iteration:

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

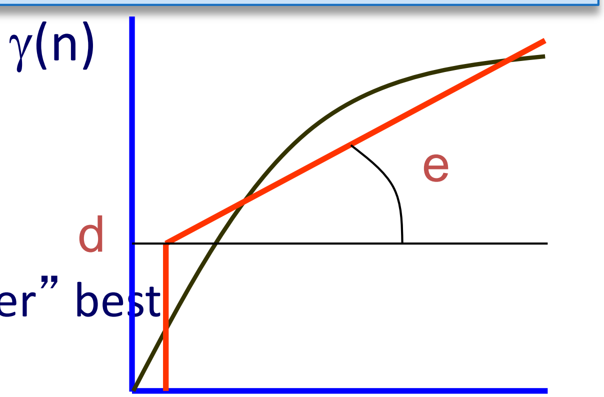
Termination: same

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

Space: $O(NM)$

A compromise: affine gaps

$$\gamma(n) = \underbrace{d}_{\text{gap open}} + (n-1) \times \underbrace{e}_{\text{gap extend}}$$



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

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

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

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

Iteration:

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

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

Termination: same

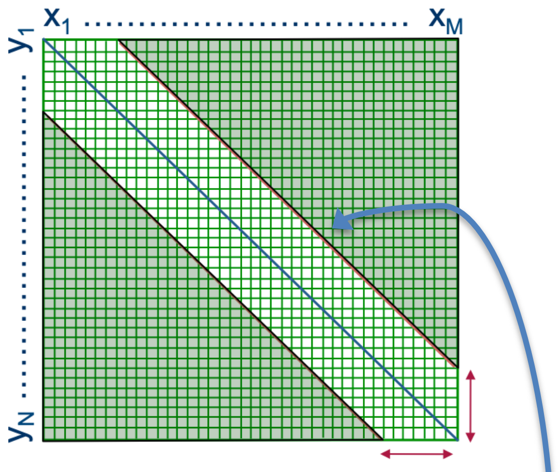
Banded Dynamic Programming: a special case

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

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

x_i
| implies $| i – j | < k(N)$
y_j

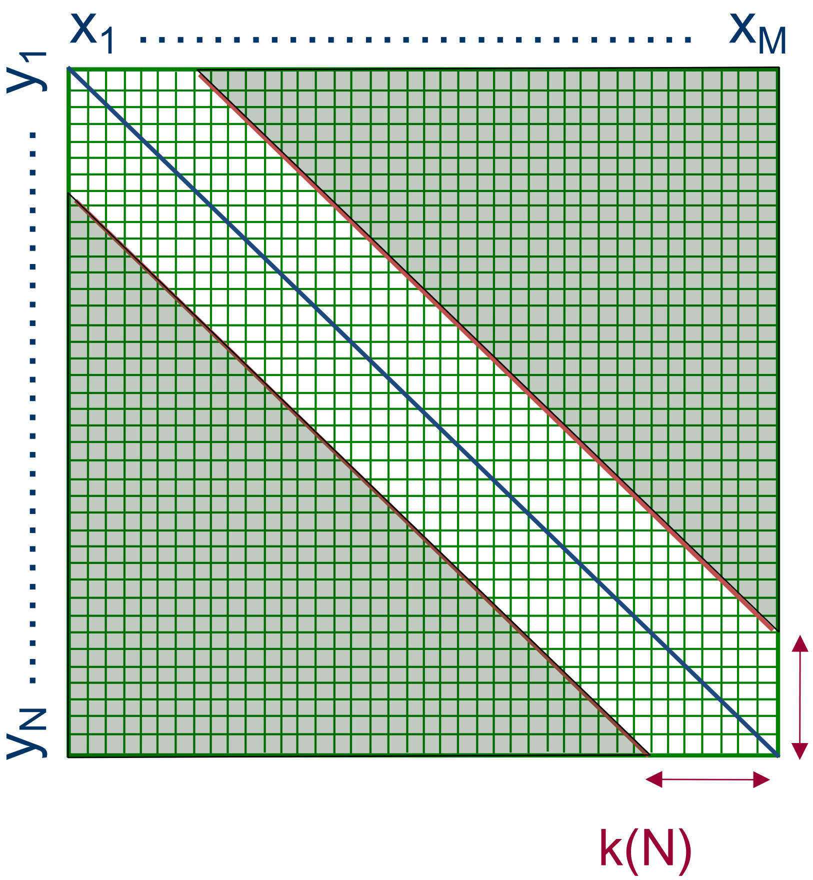
Time, Space: $O(N \times k(N)) << O(N^2)$



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

Note that for diagonals, i-j = constant.

Banded Dynamic Programming



Initialization:

F(i,0), F(0,j) undefined for i, j > k

Iteration:

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

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

Termination: same

Easy to extend to the affine gap case

match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	← -1					
C 1							
A 2							
T 3							
G 4							
T 5							

A

–

Example

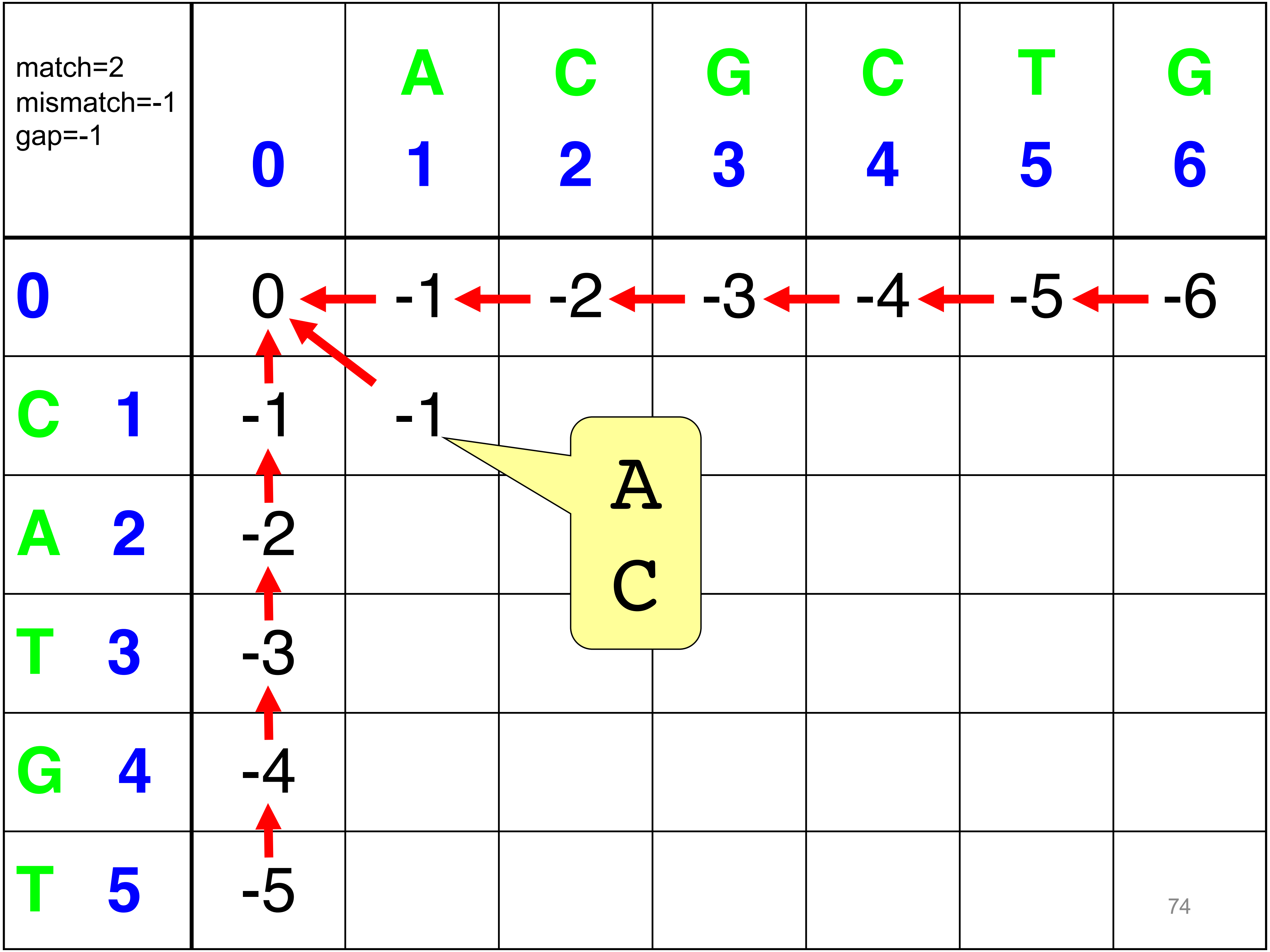
Example global alignment

match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	← -1	← -2	← -3	← -4	← -5	← -6
C 1							
A 2							
T 3							
G 4							
T 5							

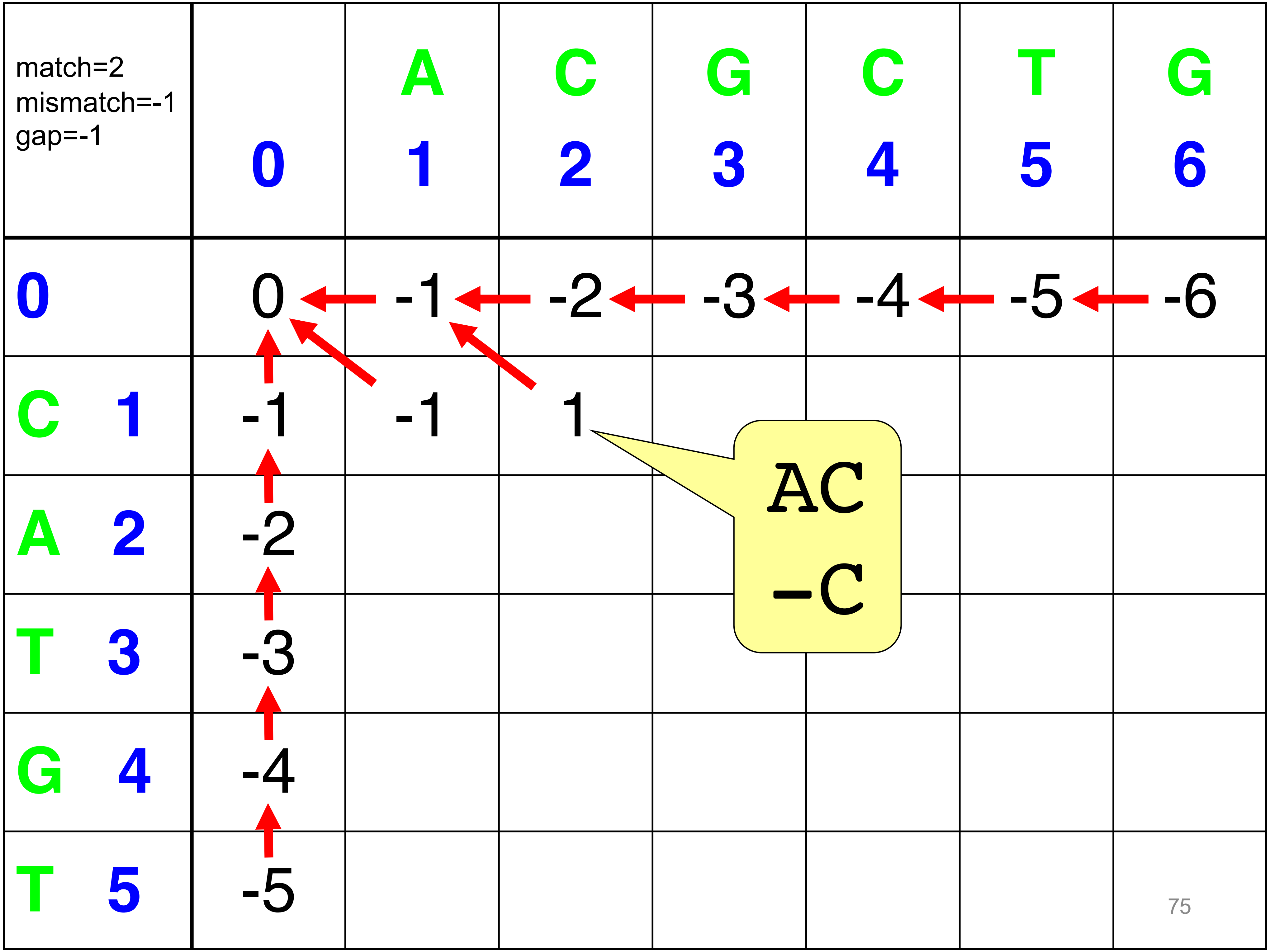
```
ACGCTG
------
```

match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	← -1	← -2	← -3	← -4	← -5	← -6
C 1	↑ -1						
A 2	↑ -2						
T 3	↑ -3						
G 4	↑ -4						
T 5	↑ -5						

```
-----
CATGT
```



match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1	-2	-3	-4	-5	-6
C 1	-1	-1					
A 2	-2						
T 3	-3						
G 4	-4						
T 5	-5						



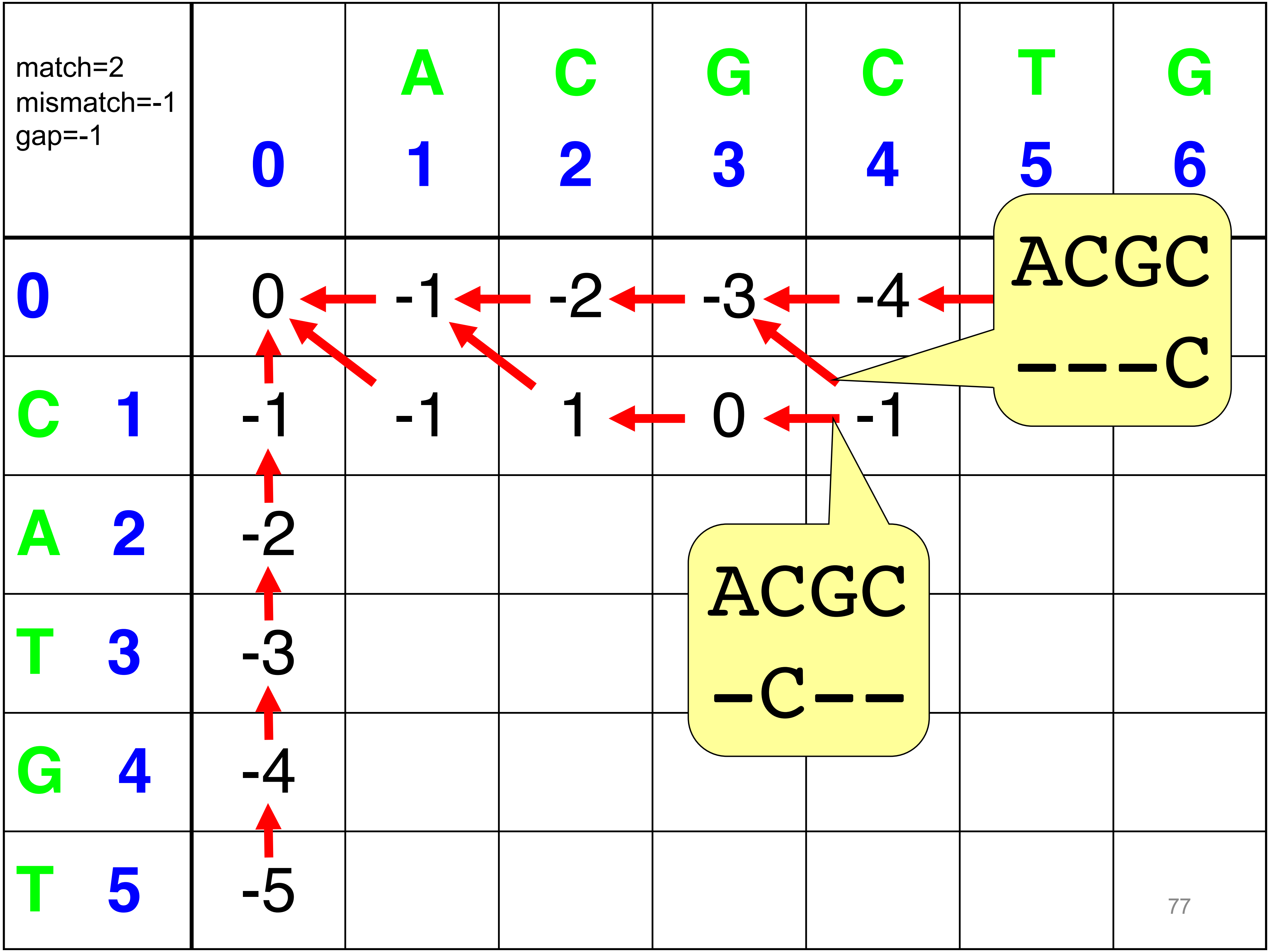
match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1	-2	-3	-4	-5	-6
C 1	-1	-1	1				
A 2	-2						
T 3	-3						
G 4	-4						
T 5	-5						

```
AC
-C
```

match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	← -1	← -2	← -3	← -4	← -5	← -6
C 1	↑ -1	↖ -1	↖ 1	← 0			
A 2	↑ -2						
T 3	↑ -3						
G 4	↑ -4						
T 5	↑ -5						

ACG

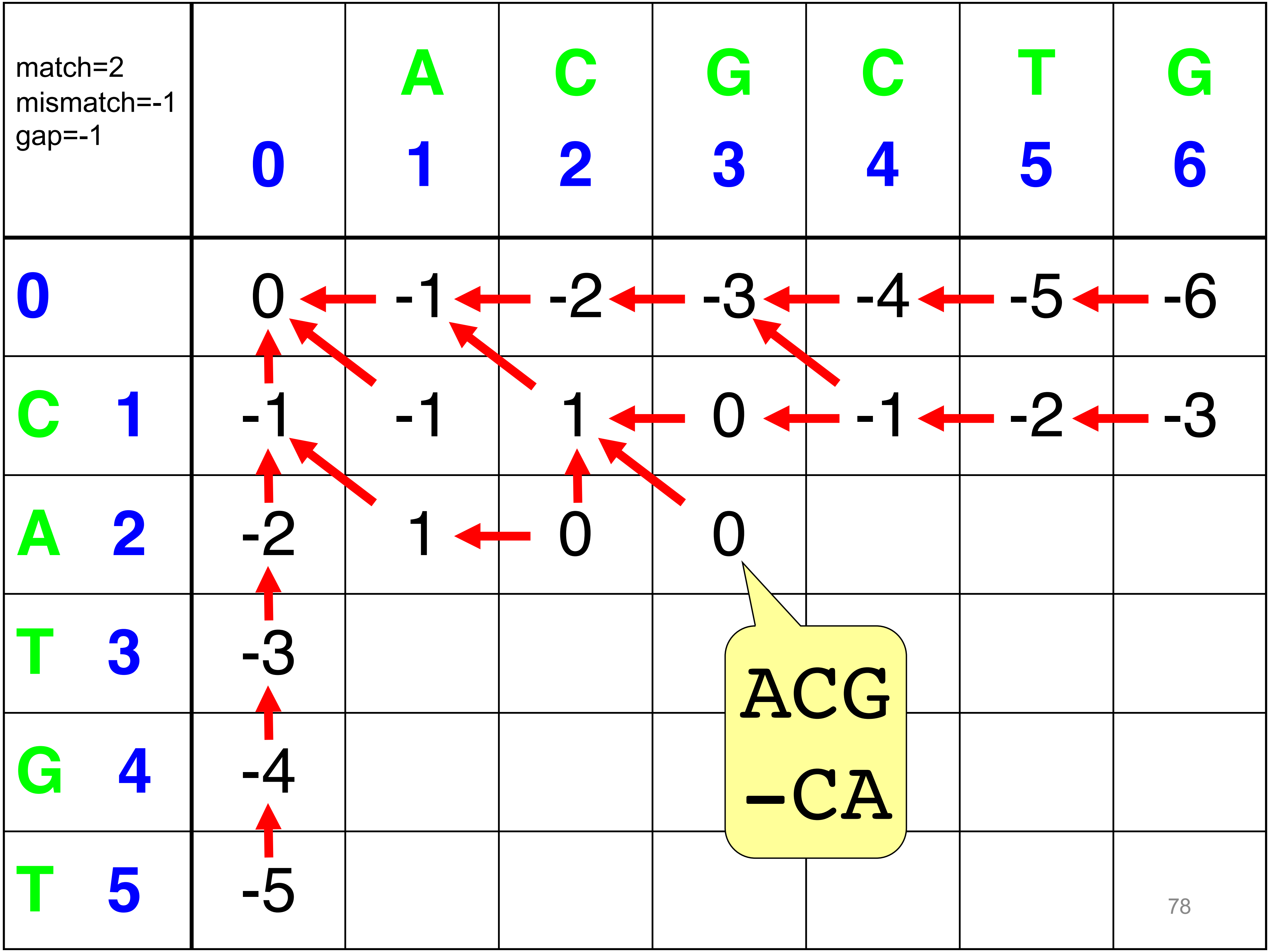
–C–



match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1	-2	-3	-4		
C 1	-1	-1	1	0	-1		
A 2	-2						
T 3	-3						
G 4	-4						
T 5	-5						

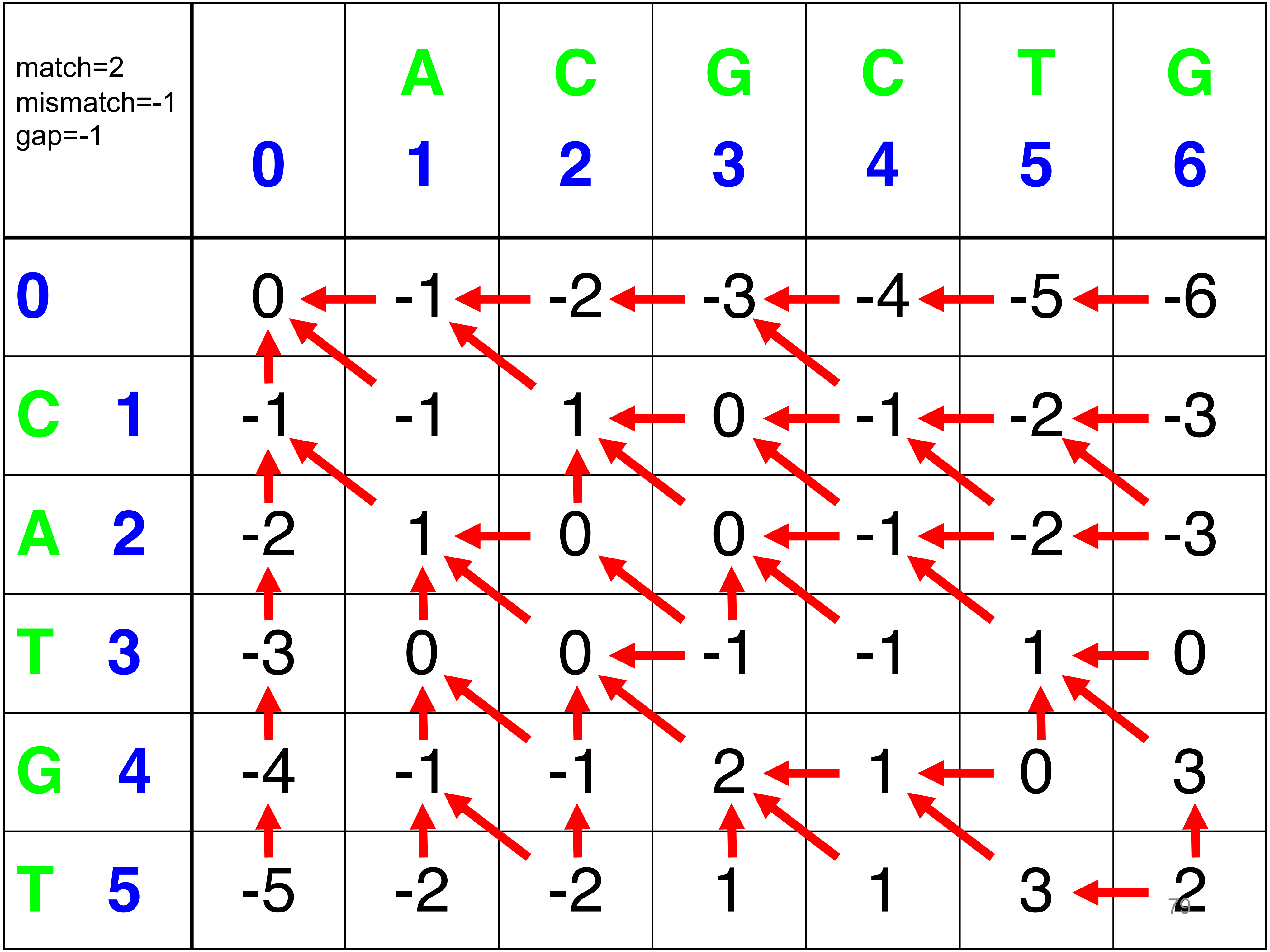
ACGC
---C

ACGC
-C--

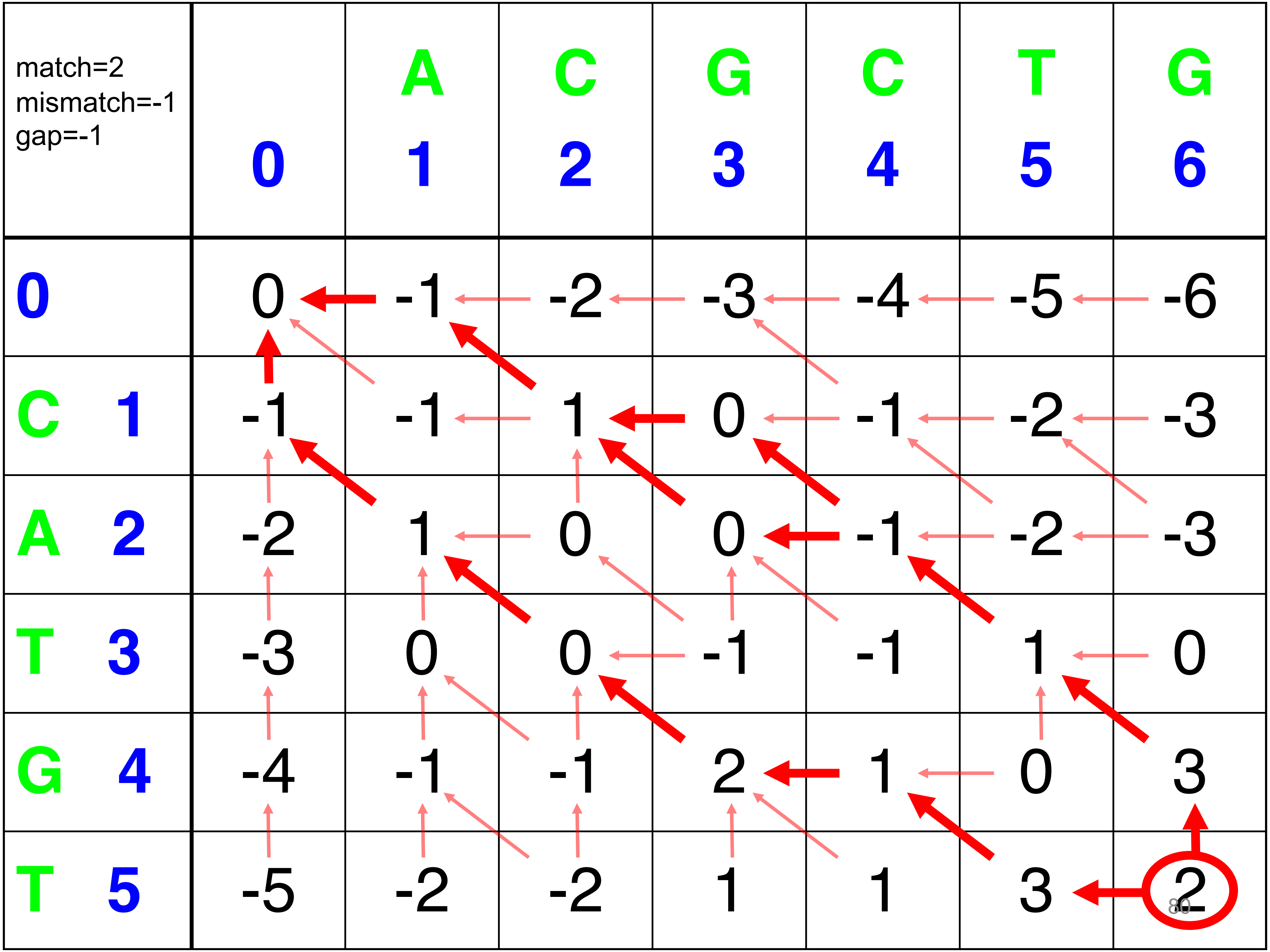


match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1	-2	-3	-4	-5	-6
C 1	-1	-1	1	0	-1	-2	-3
A 2	-2	1	0	0			
T 3	-3						
G 4	-4						
T 5	-5						

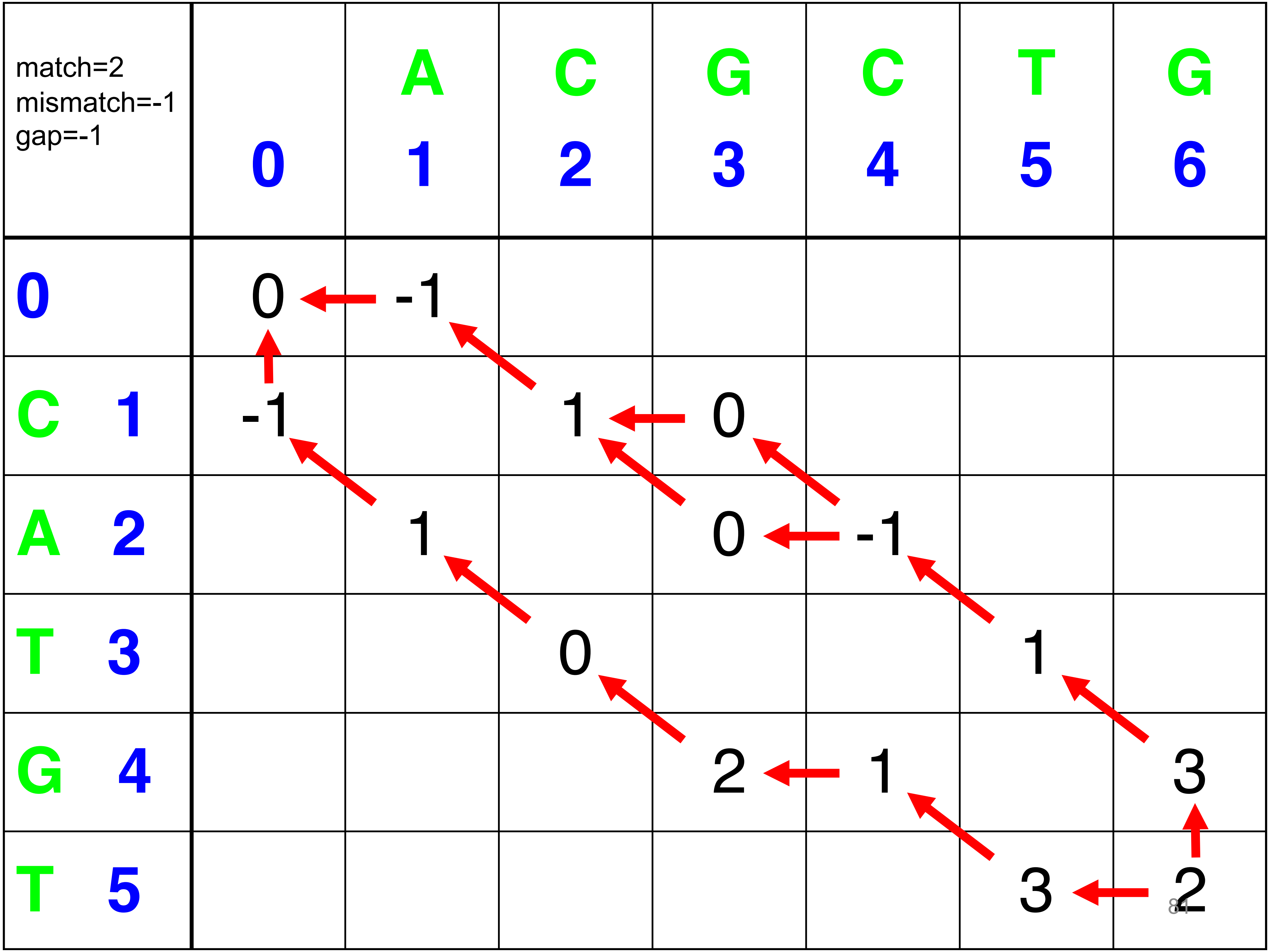
ACG
–CA



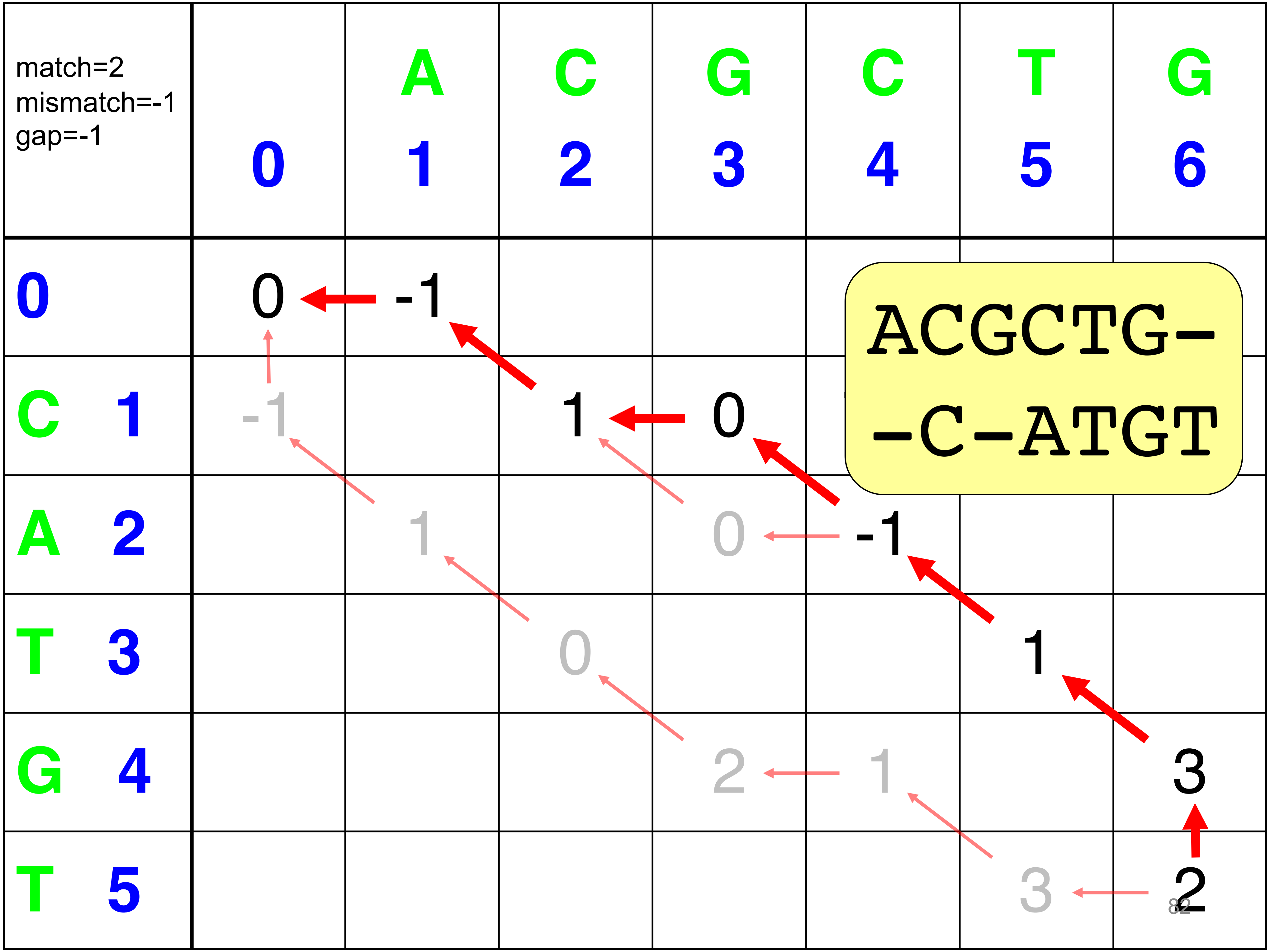
match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1	-2	-3	-4	-5	-6
C 1	-1	-1	1	0	-1	-2	-3
A 2	-2	1	0	0	-1	-2	-3
T 3	-3	0	0	-1	-1	1	0
G 4	-4	-1	-1	2	1	0	3
T 5	-5	-2	-2	1	1	3	2



match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1	-2	-3	-4	-5	-6
C 1	-1	-1	1	0	-1	-2	-3
A 2	-2	1	0	0	-1	-2	-3
T 3	-3	0	0	-1	-1	1	0
G 4	-4	-1	-1	2	1	0	3
T 5	-5	-2	-2	1	1	3	2

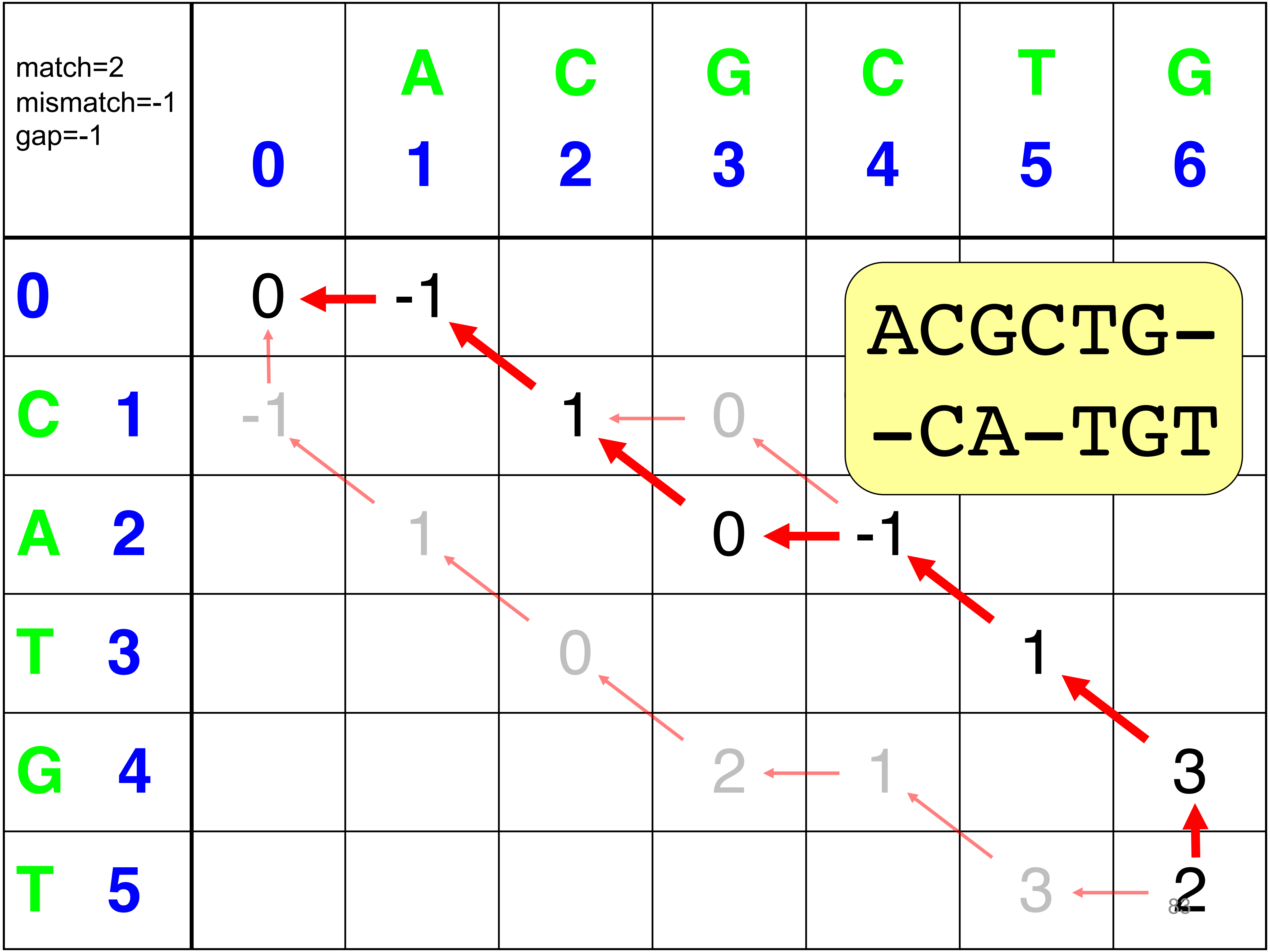


match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1					
C 1	-1		1	0			
A 2		1		0	-1		
T 3			0			1	
G 4				2	1		3
T 5						3	2



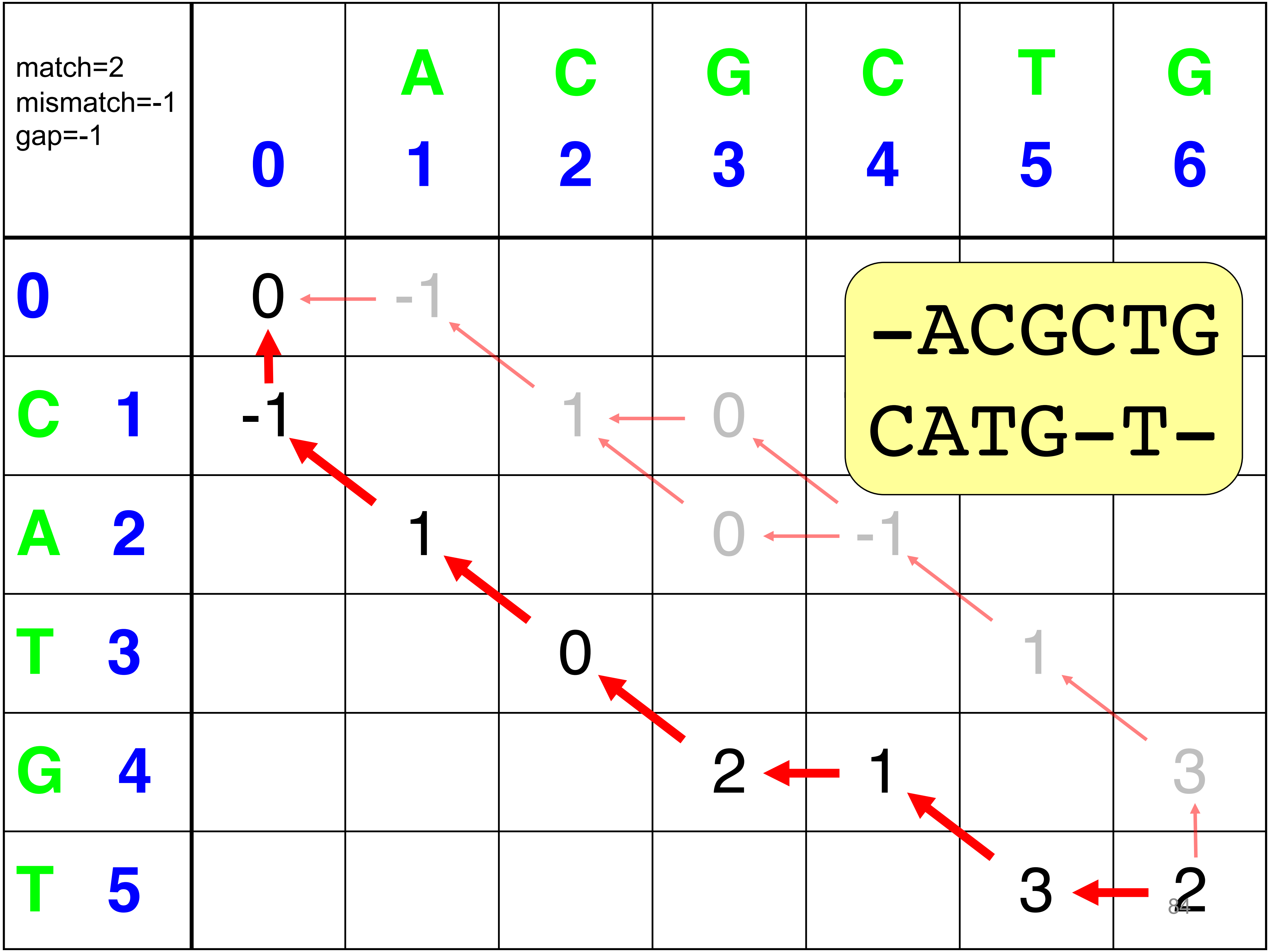
match=2 mismatch=-1 gap=-1	 0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1					
C 1	-1		1	0			
A 2		1		0	-1		
T 3			0			1	
G 4				2	1		3
T 5						3	2

```
ACGCTG-
-C-ATGT
```



match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1					
C 1	-1		1	0			
A 2		1		0	-1		
T 3			0			1	
G 4				2	1		3
T 5						3	2

```
ACGCTG-
-CA-TGT
```



match=2 mismatch=-1 gap=-1	0	A 1	C 2	G 3	C 4	T 5	G 6
0	0	-1					
C 1	-1		1	0			
A 2		1		0	-1		
T 3			0			1	
G 4				2	1		3
T 5						3	2

```
-ACGCTG
CATG-T-
```

Local Alignment Example

match=1
mismatch=-1
gap=-1

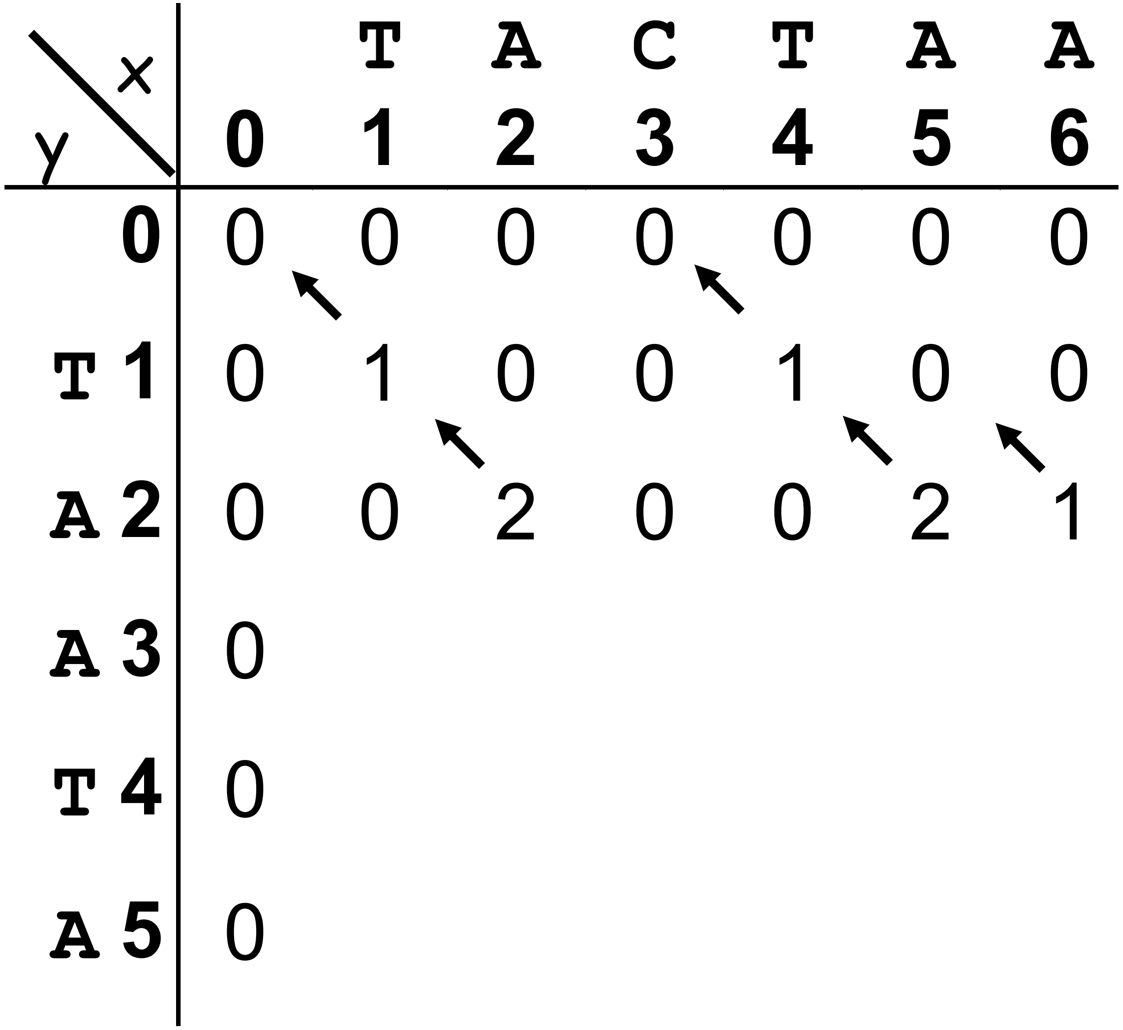
$y =$ TAATA
$x =$ TACTAA

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

Example

Local Alignment Example

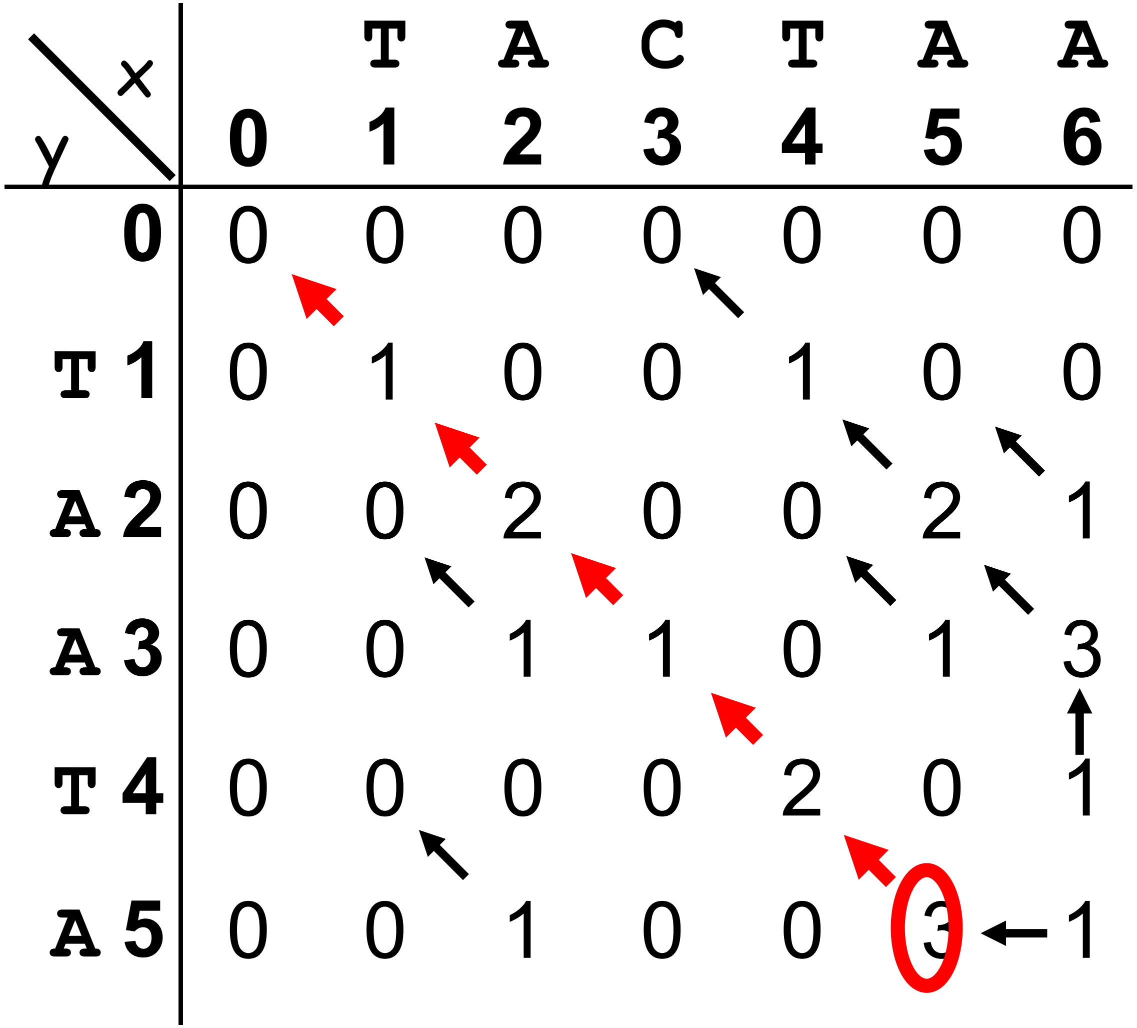
y = TAATA
x = TACTAA



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						
T	4	0						
A	5	0						

Local Alignment Example

y = TAATA-
x = TACTAA

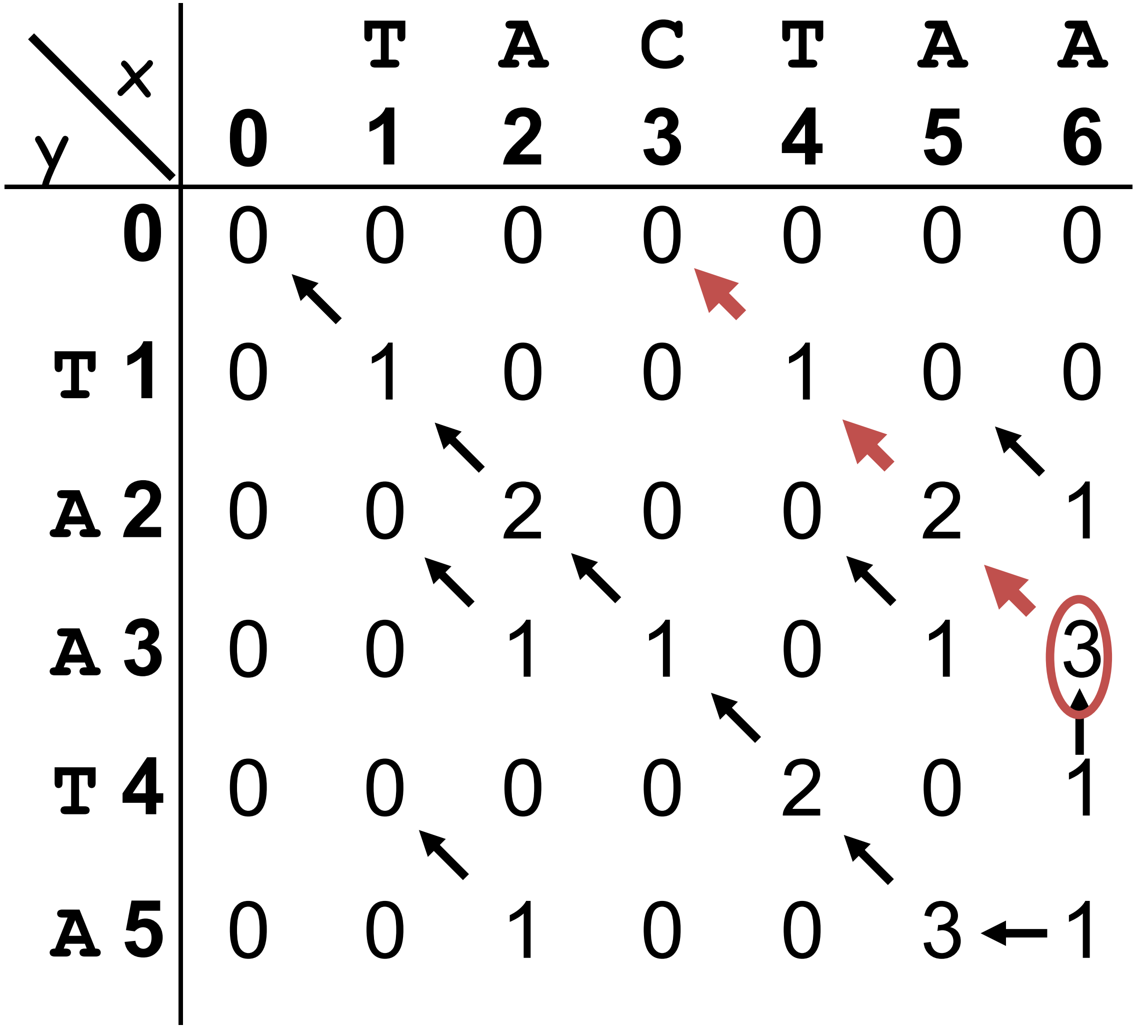


y \ x	0	T 1	A 2	C 3	T 4	A 5	A 6
0	0	0	0	0	0	0	0
T 1	0	1	0	0	1	0	0
A 2	0	0	2	0	0	2	1
A 3	0	0	1	1	0	1	3
T 4	0	0	0	0	2	0	1
A 5	0	0	1	0	0	3	1

Local Alignment Example

y = ---TAATA

x = TACTAA--

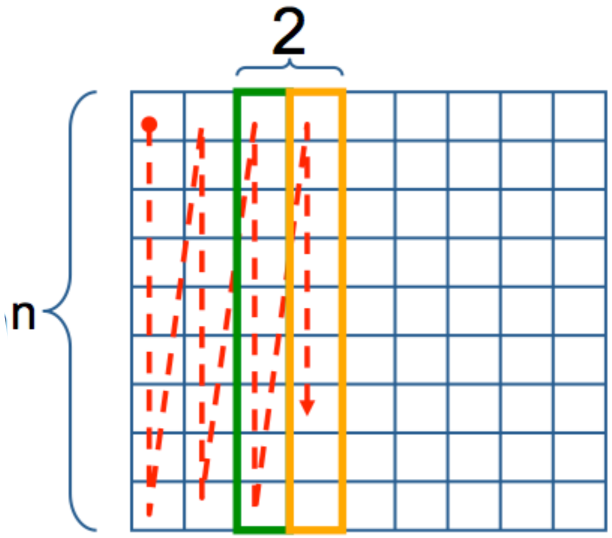


y \ x	0	T 1	A 2	C 3	T 4	A 5	A 6
0	0	0	0	0	0	0	0
T 1	0	1	0	0	1	0	0
A 2	0	0	2	0	0	2	1
A 3	0	0	1	1	0	1	3
T 4	0	0	0	0	2	0	1
A 5	0	0	1	0	0	3	1

Computing Alignment Score with Linear Memory

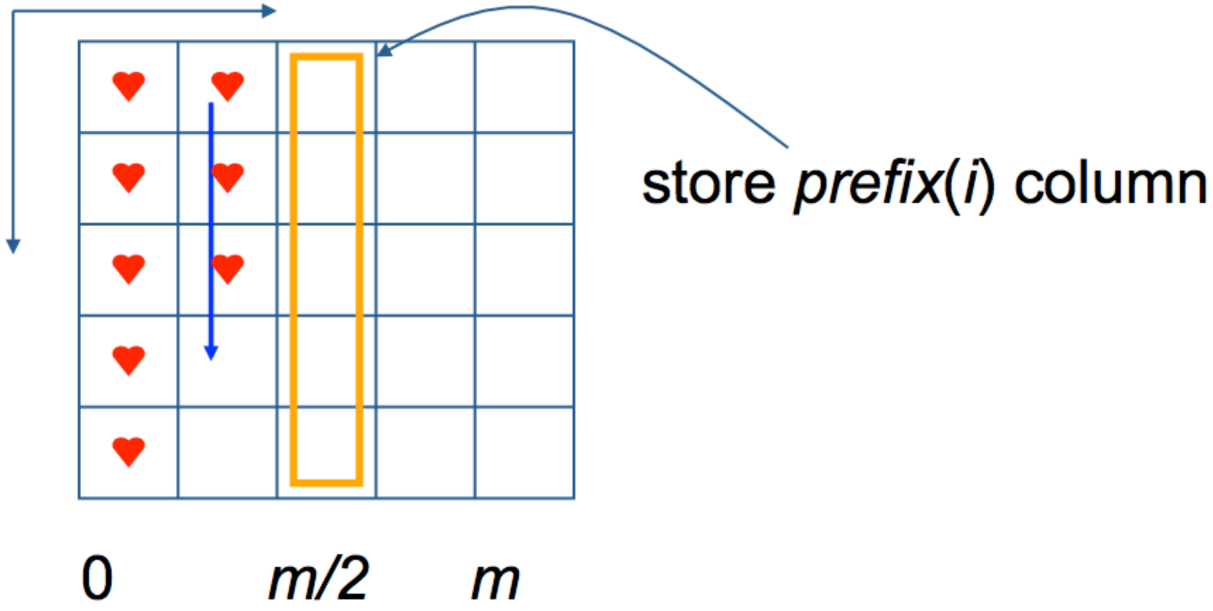
Alignment Score

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



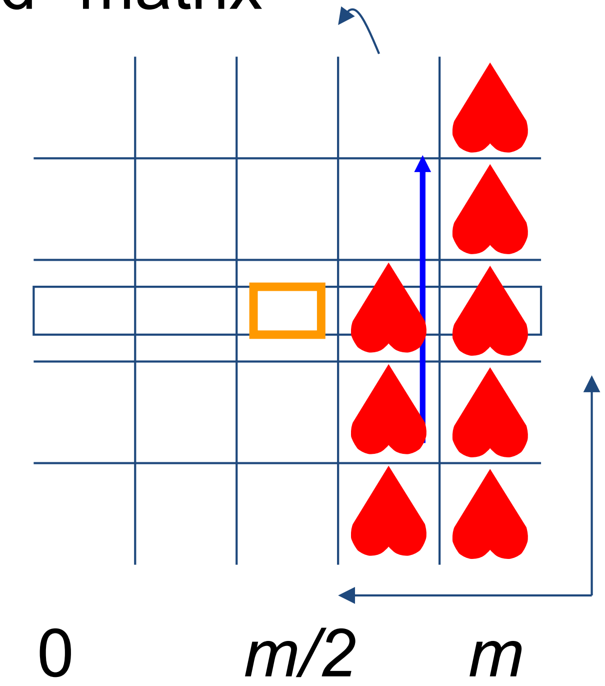
Computing Prefix(*i*)

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



Computing Suffix(*i*)

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

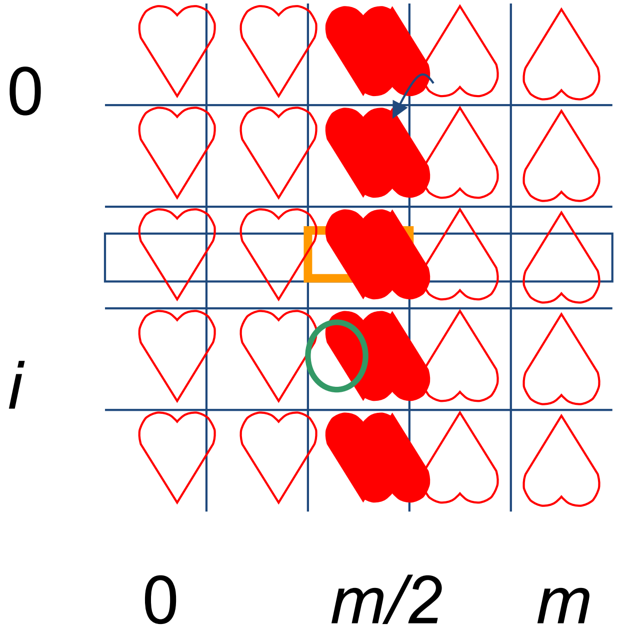


store *suffix*(*i*) column

Computing Length(*i*)

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

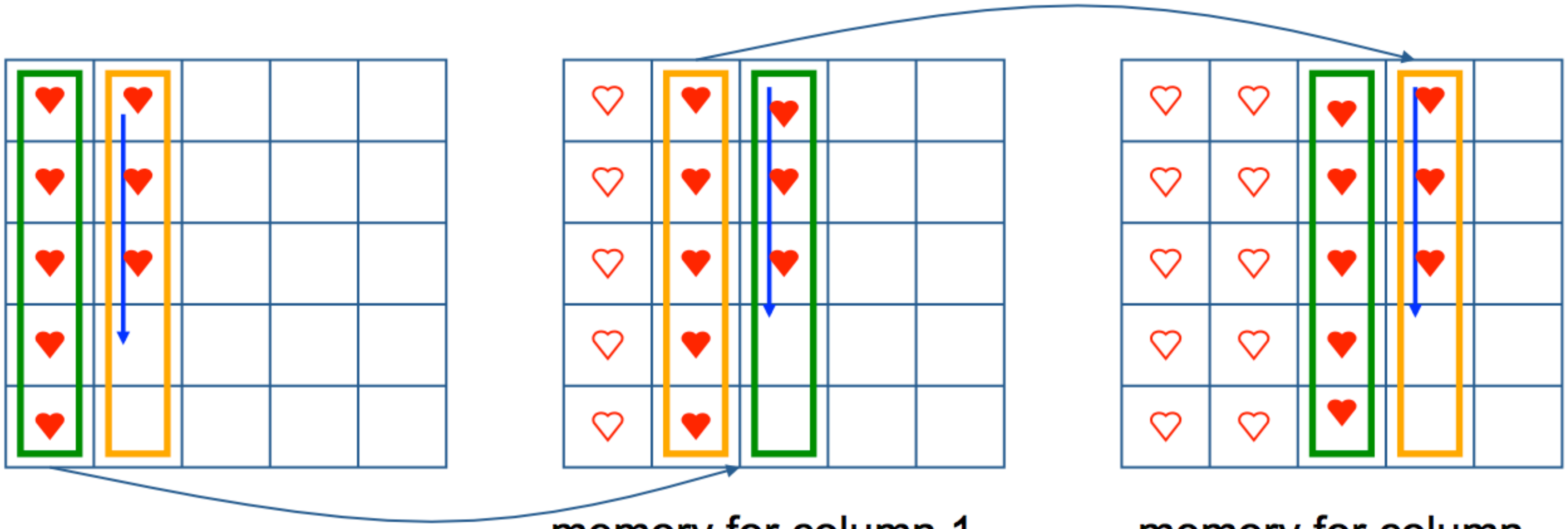
- Add *prefix*(*i*) and *suffix*(*i*) to compute *length(i):*
  - *length*(*i*)=*prefix*(*i*) + *suffix*(*i*)
- You now have a middle vertex of the maximum path (*i*,*m*/2) as maximum of *length(i)*



middle point found

Computing Alignment Score: Recycling Columns

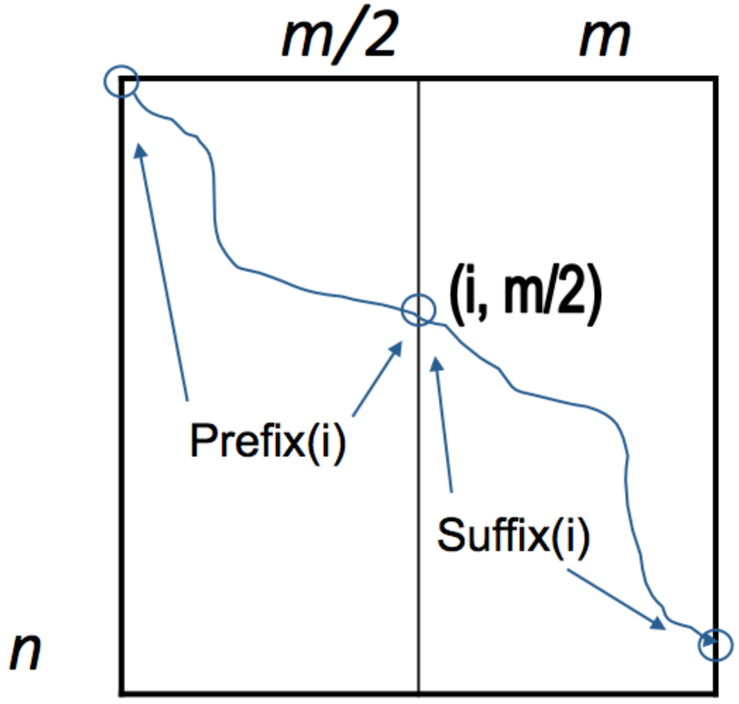
Only two columns of scores are saved at any given time



memory for column 1 is used to calculate column 3

memory for column 2 is used to calculate column 4

Crossing the Middle Line



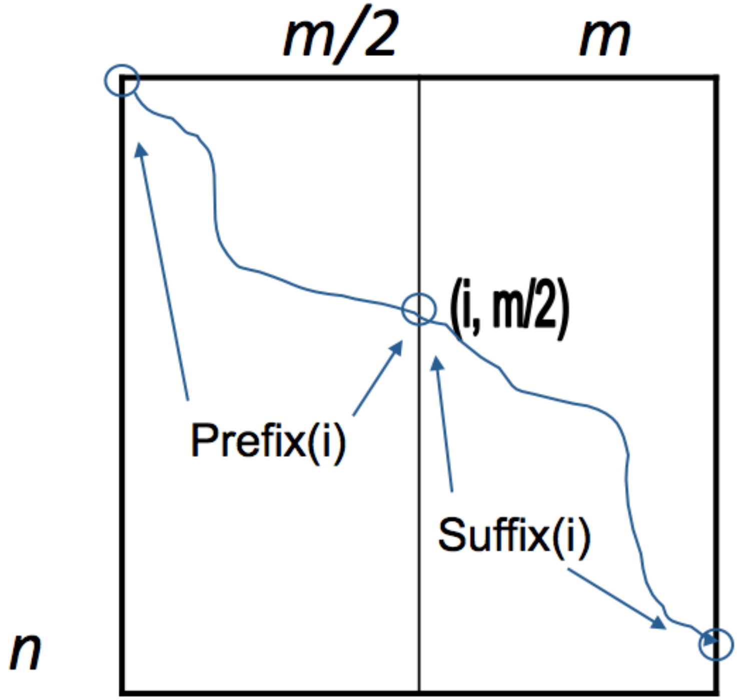
We want to calculate the longest path from (0,0) to (n,m) that passes through (i,m/2) where i ranges from 0 to n and represents the i-th row

Define

length(i)

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

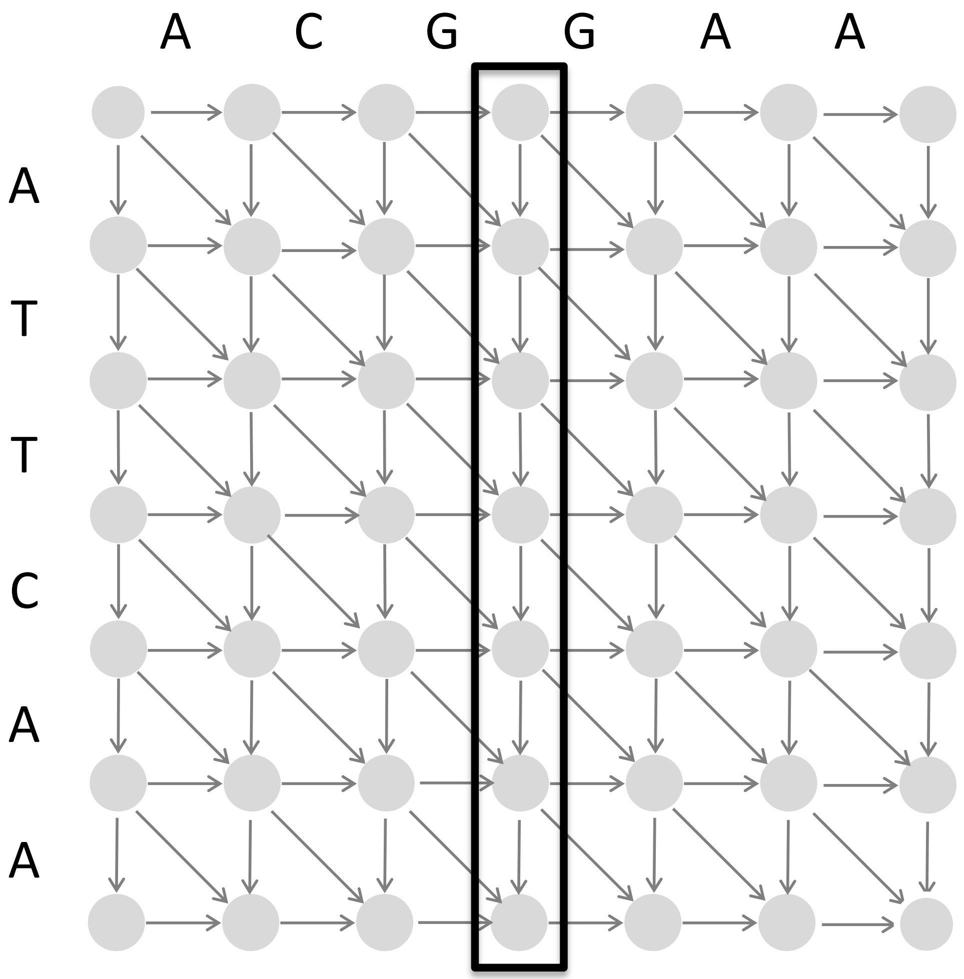
Crossing the Middle Line



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

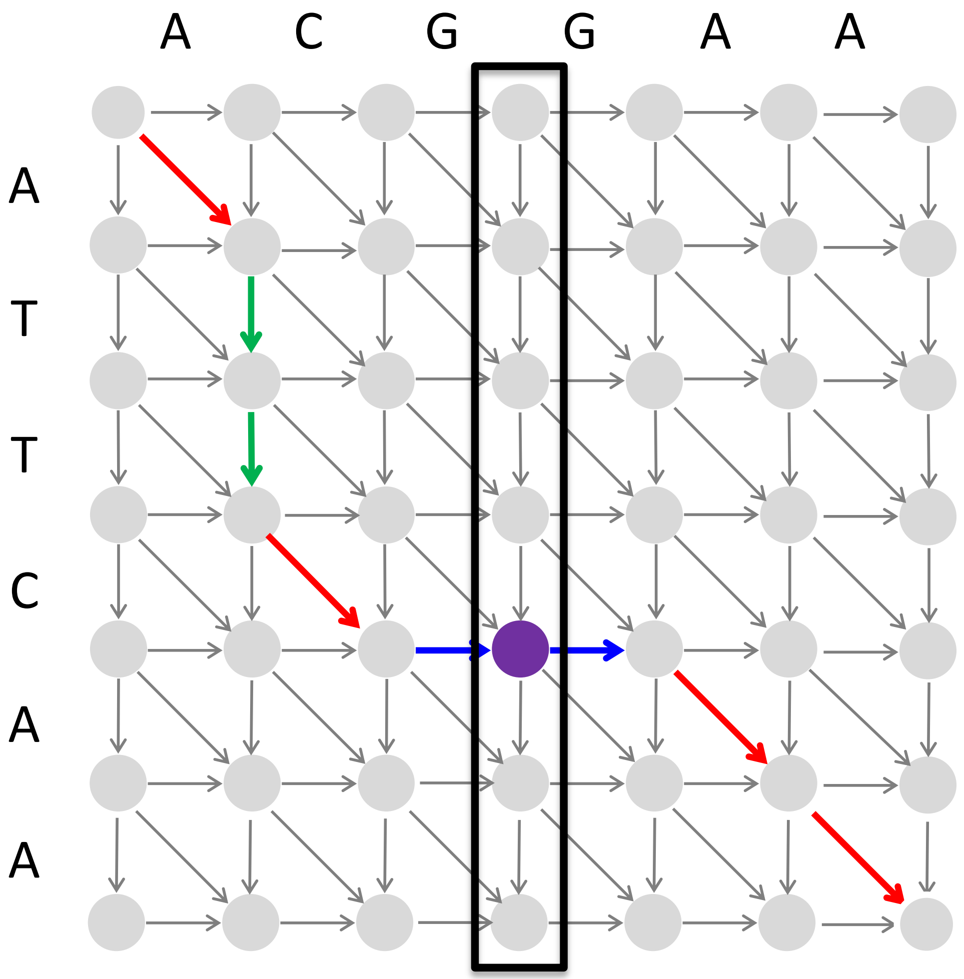
$$length(mid) = \text{optimal length} = \max_{0 \le i \le n} length(i)$$

Middle Column of the Alignment



middle column
(*middle*=#columns/2)

Middle Node of the Alignment



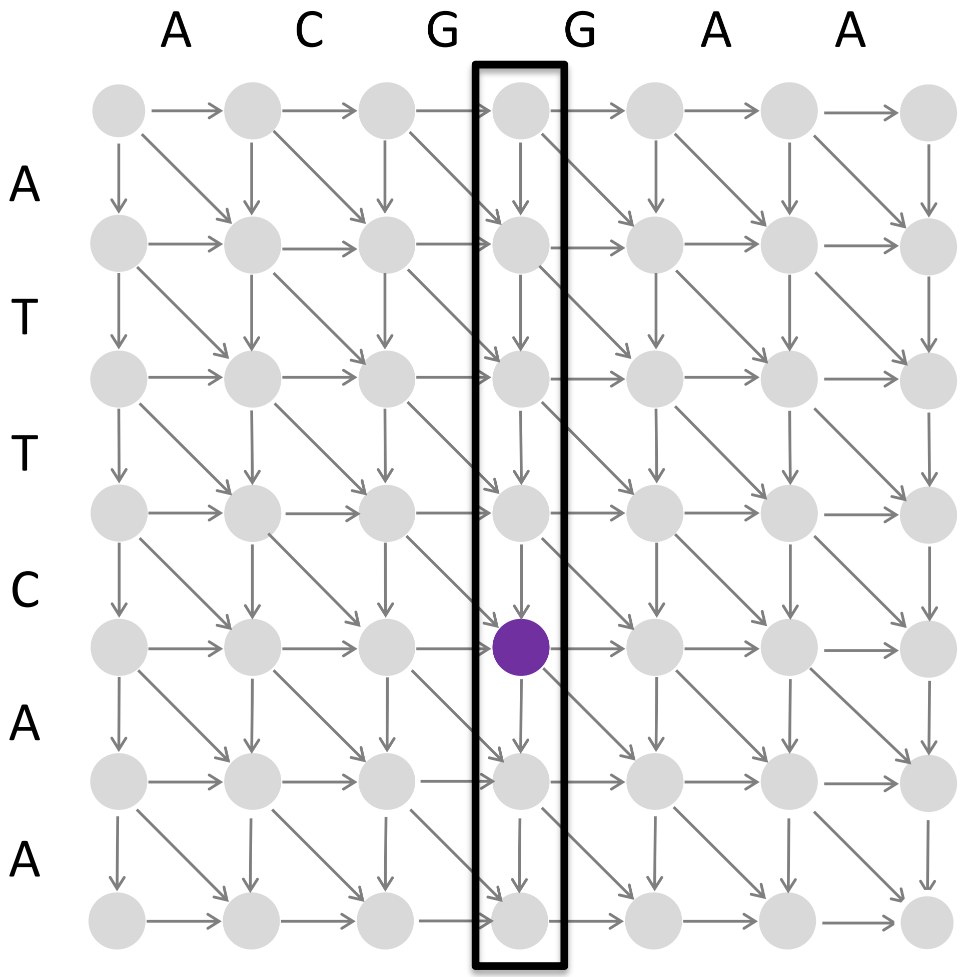
middle node

(a node where an optimal alignment path crosses the middle column)

Divide and Conquer Approach to Sequence Alignment

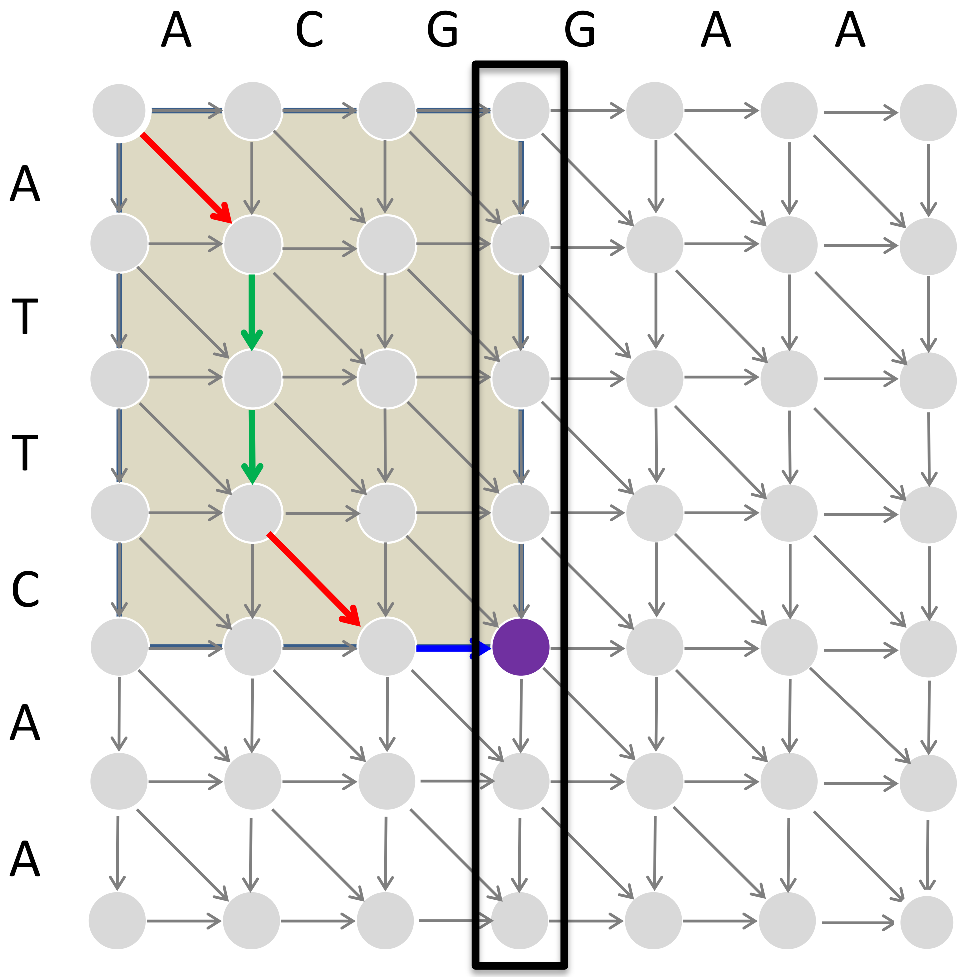
AlignmentPath(*source, sink*)

find *MiddleNode*



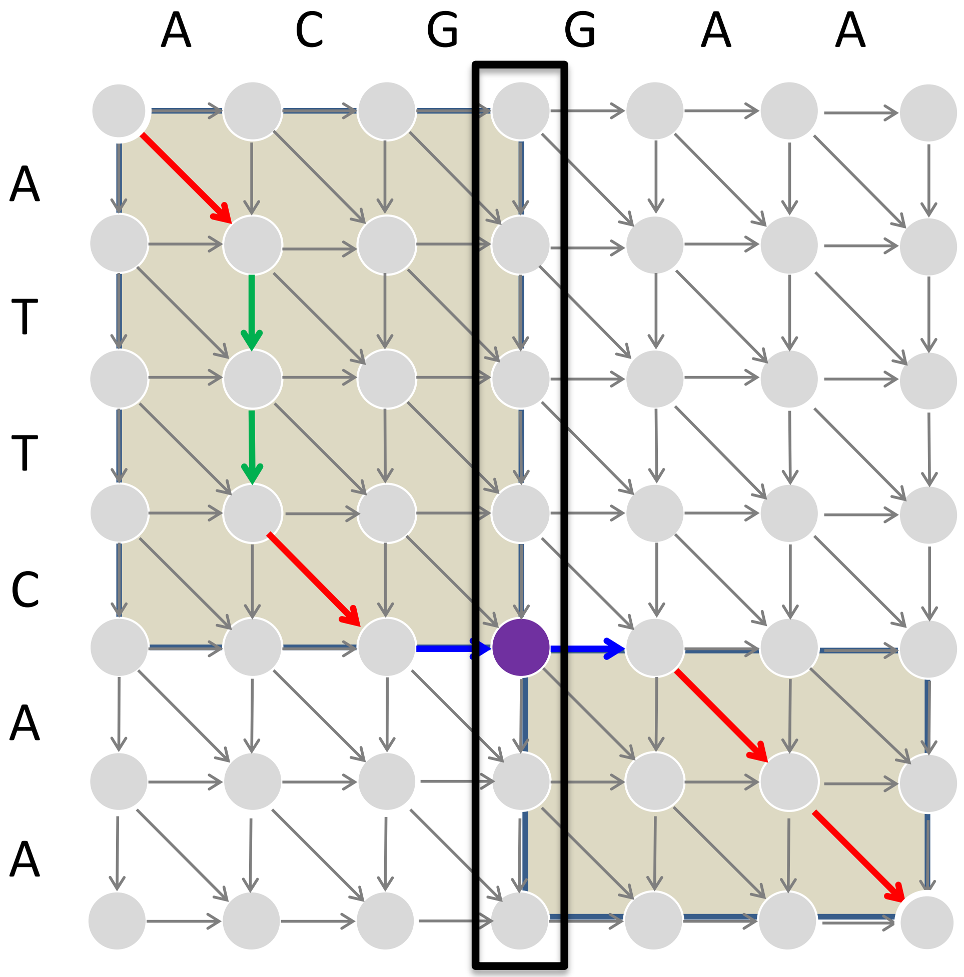
Divide and Conquer Approach to Sequence Alignment

```
AlignmentPath(source, sink)
    find MiddleNode
    AlignmentPath(source, MiddleNode)
```



Divide and Conquer Approach to Sequence Alignment

```
**AlignmentPath**(*source, sink*)
    find *MiddleNode*
    **AlignmentPath**(*source, MiddleNode*)
    **AlignmentPath**(*MiddleNode, sink*)
```



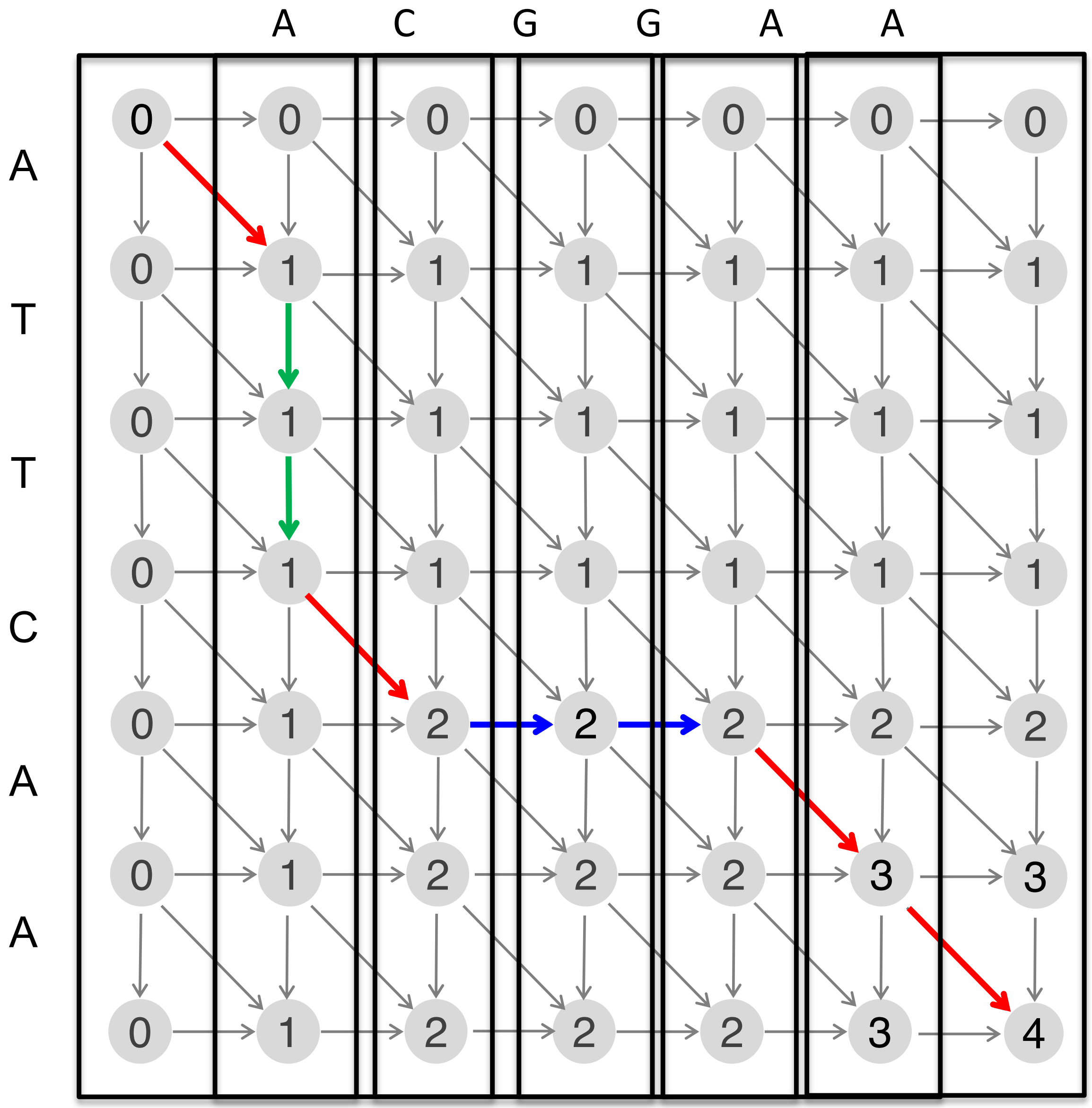
The only problem left is how to find this middle node in **linear space**!

Computing Alignment Score in Linear Space

Finding the **longest path** in the alignment graph **requires** storing all backtracking pointers – O(*nm*) memory.

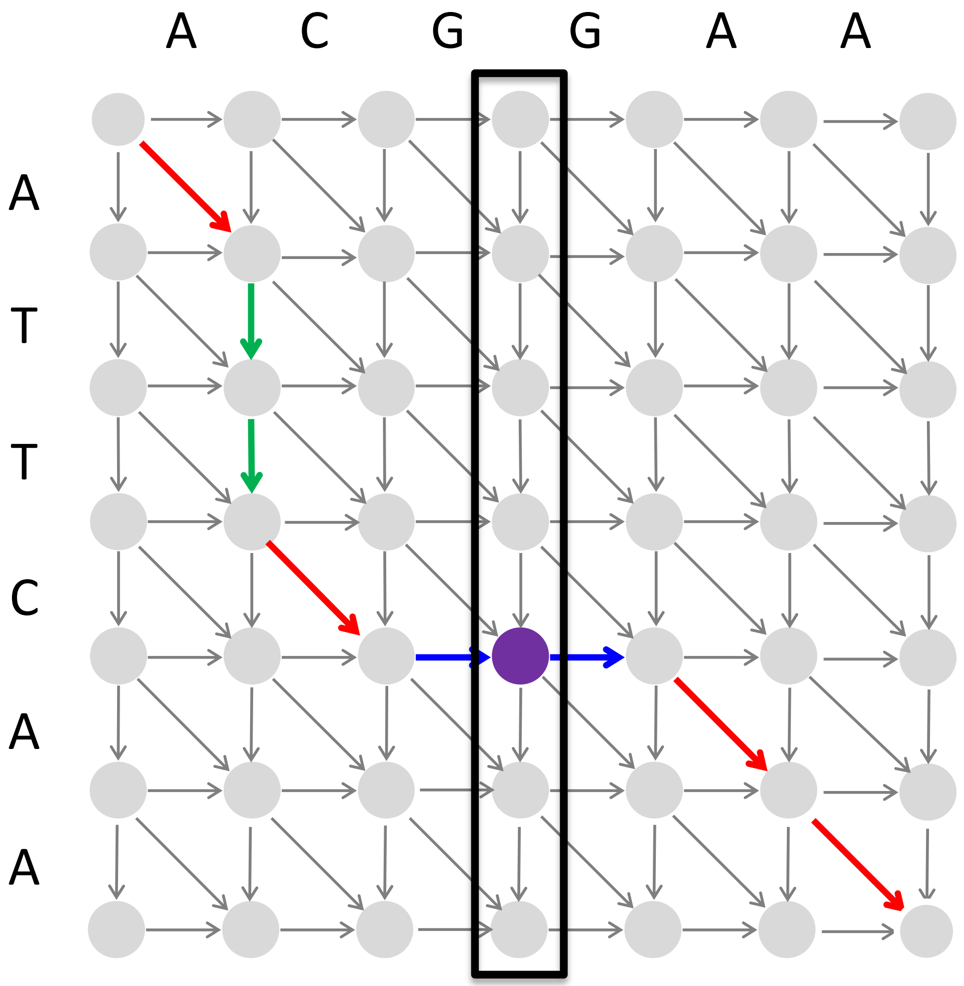
Finding the **length of the longest path** in the alignment graph **does not require** storing any backtracking pointers – O(*n*) memory.

Recycling the Columns in the Alignment Graph



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

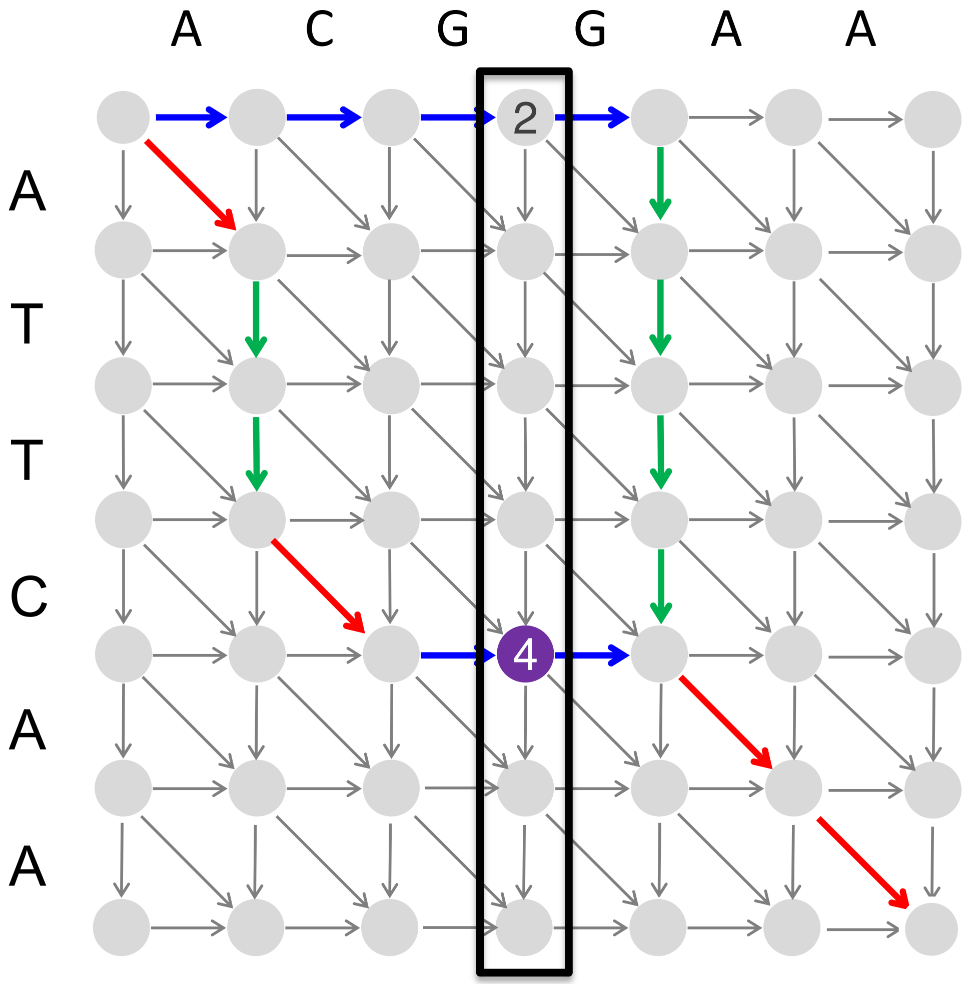
Can We Find the Middle Node without Constructing the Longest Path?



4-path that visits the node (4,middle)
In the middle column

***i*-path** – a longest path among paths that visit the *i*-th node in the middle column

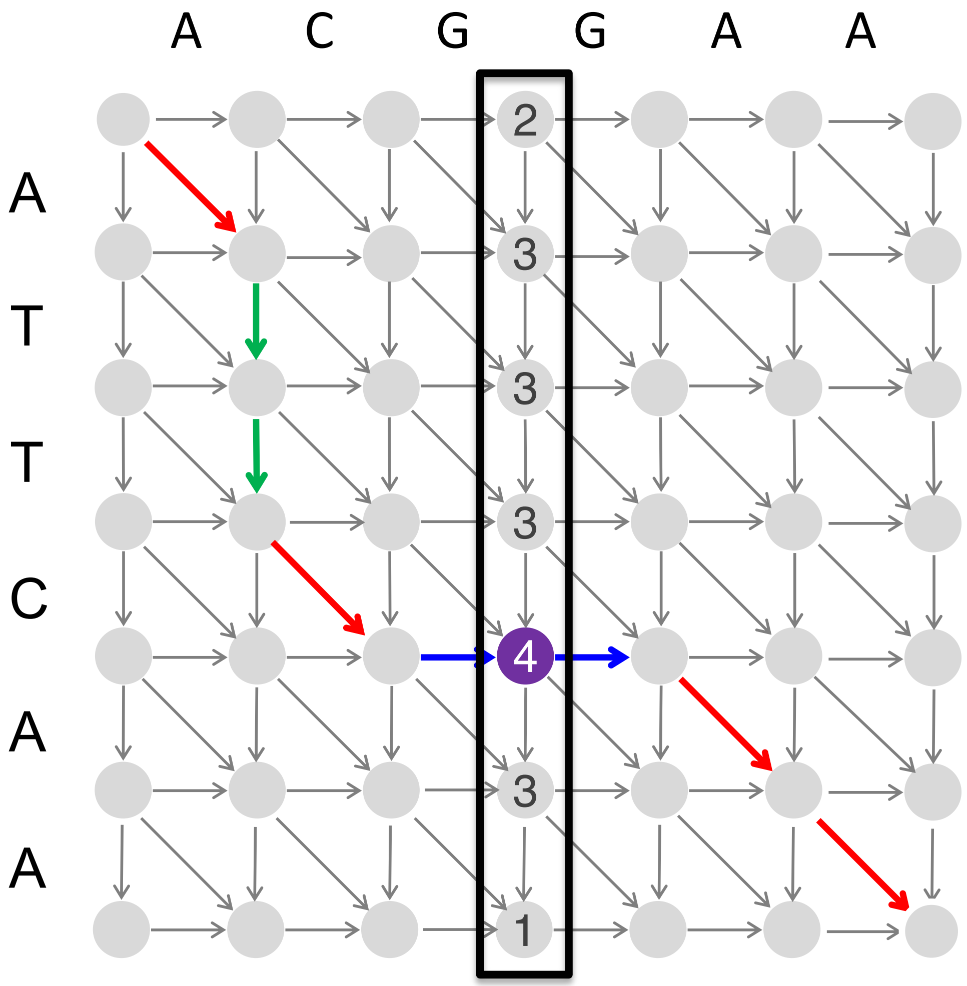
Can We Find The Lengths of All *i*-paths?



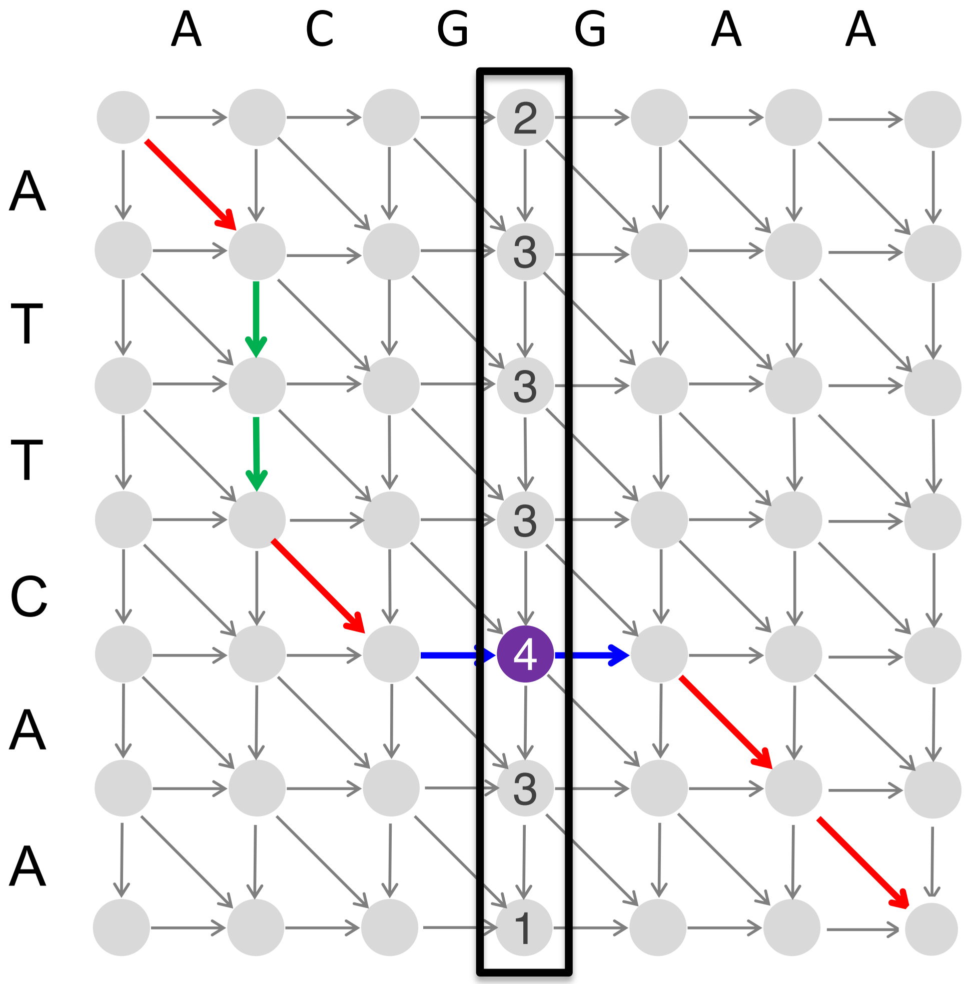
length(i):
length of an *i*-path:

length(0)=2
length(4)=4

Can We Find The Lengths of All *i*-paths?



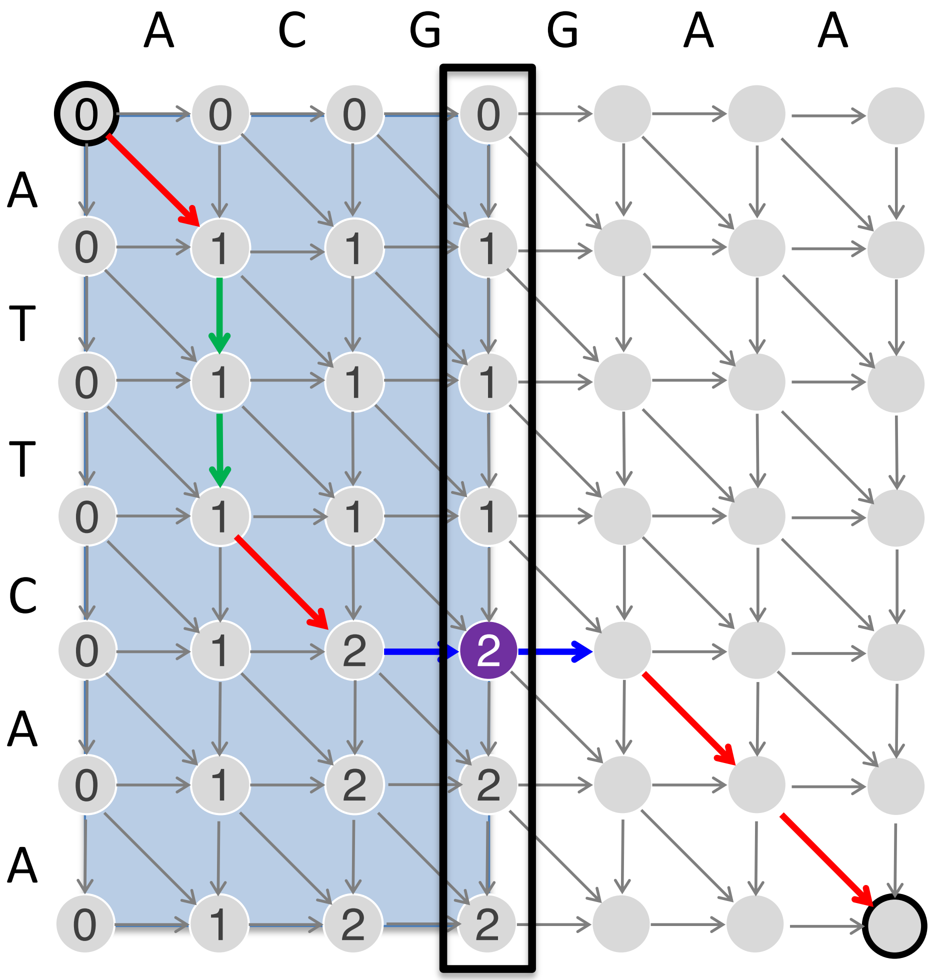
Can We Find The Lengths of *i*-paths?



length(i):
length of an *i*-path

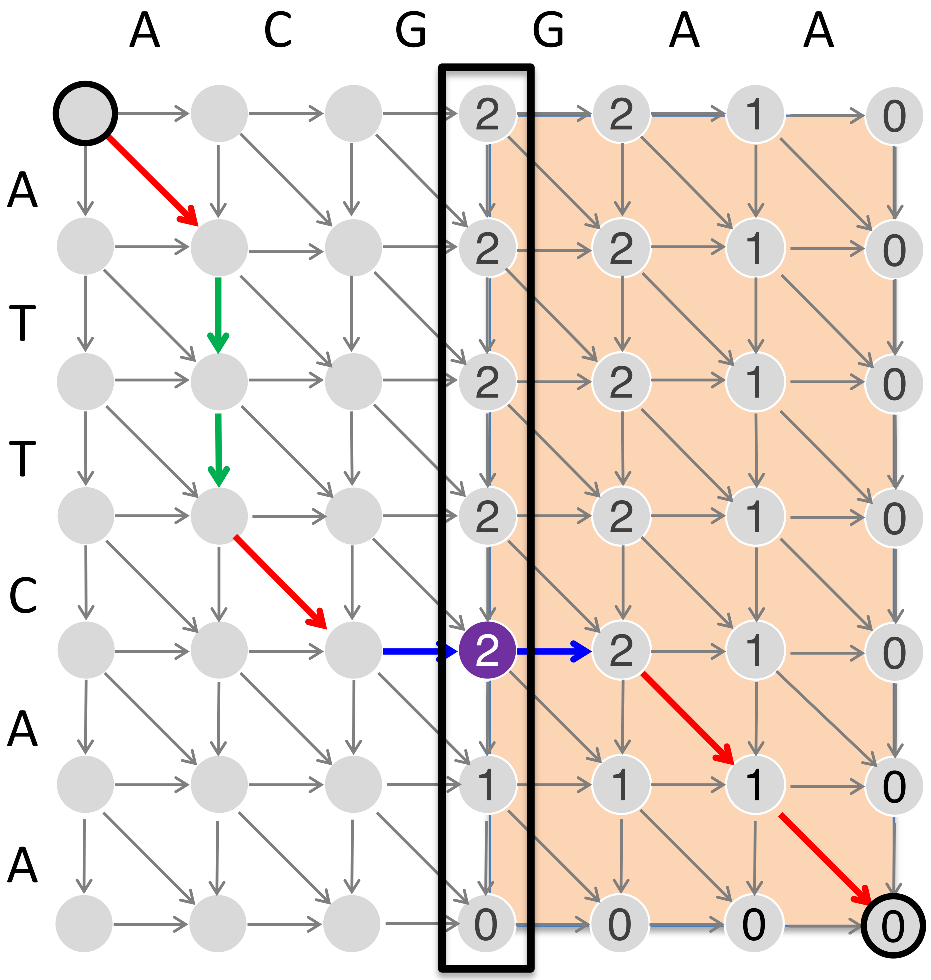
length(i)=fromSource(i)+toSink(i)

Computing *FromSource* and *toSink*



0	0	0	0			
0	1	1	1			
0	1	1	1			
0	1	1	1			
0	1	2	2			
0	1	2	2			
0	1	2	2			

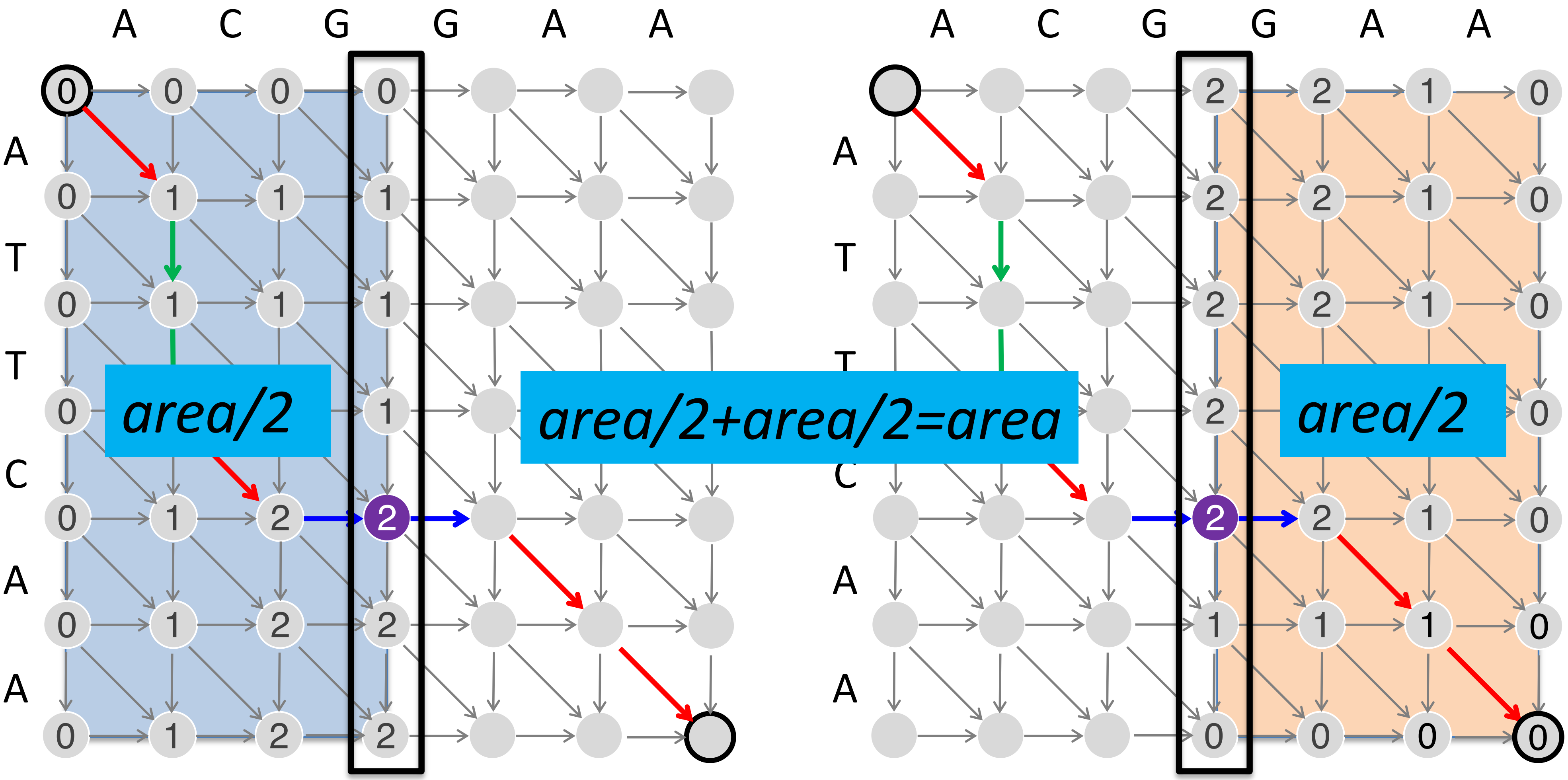
fromSource(i)



			2	2	1	0
			2	2	1	0
			2	2	1	0
			2	2	1	0
			2	2	1	0
			1	1	1	0
			0	0	0	0

toSink(i)

How Much Time Did It Take to Find the Middle Node ?



area/2

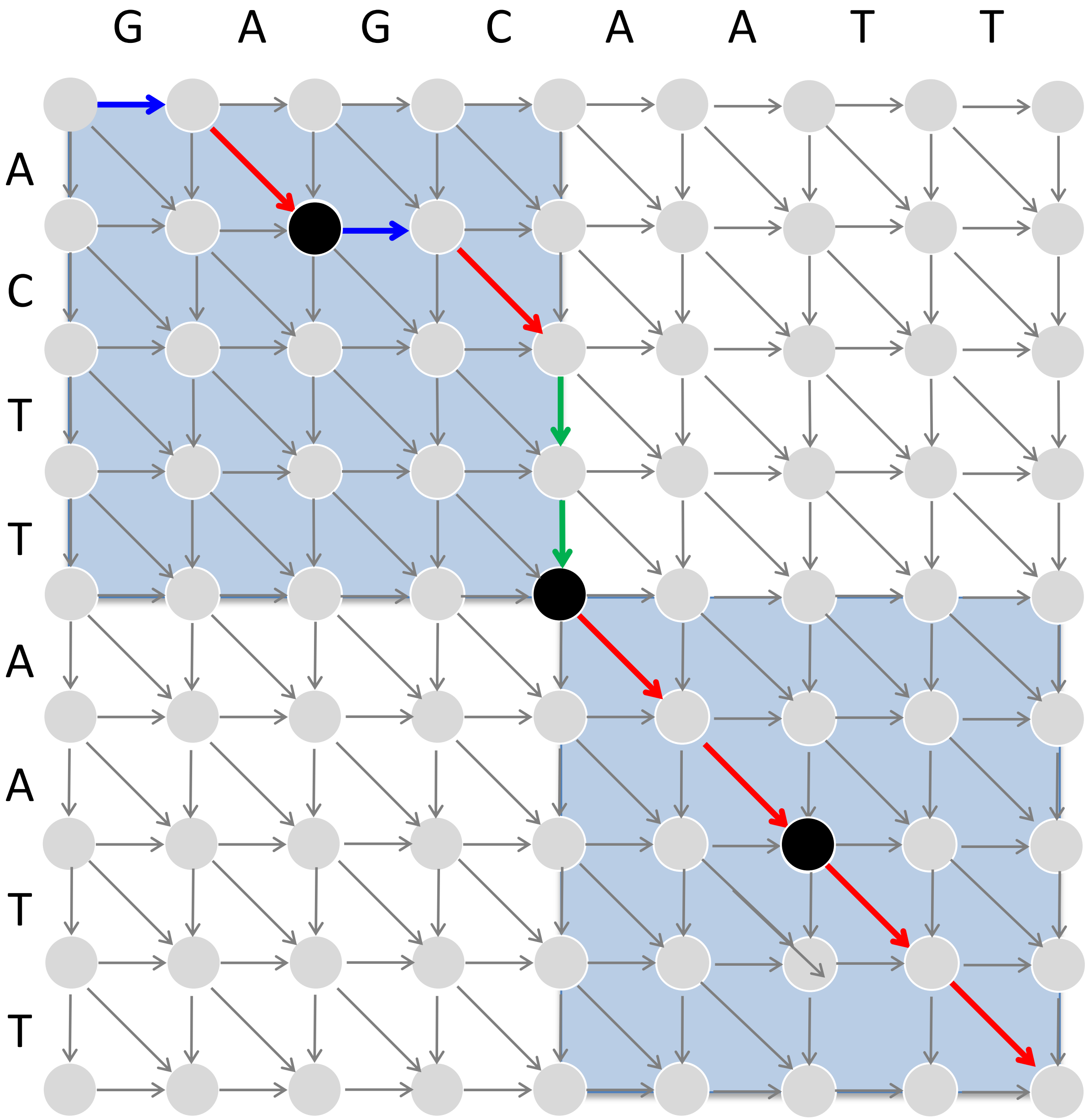
area/2+area/2=area

area/2

fromSource(i)

toSink(i)

Laughable Progress: O(*nm*) Time to Find **ONE** Node!

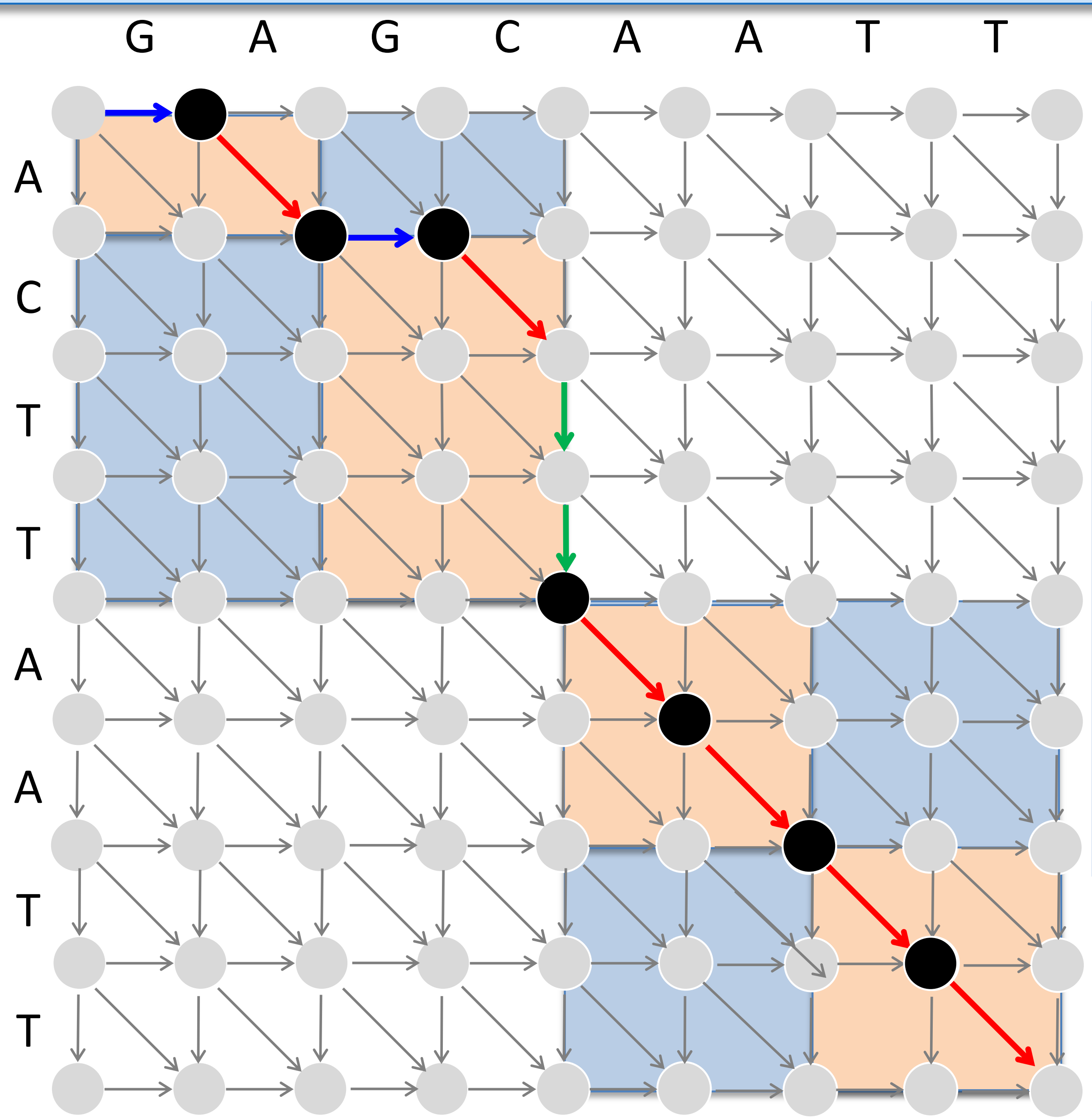


Each subproblem can be conquered in time proportional to its area:

area/4+area/4=
area/2

How much time would it take to conquer 2 subproblems?

Laughable Progress: O(*nm+nm/2*) Time to Find **THREE** Nodes!

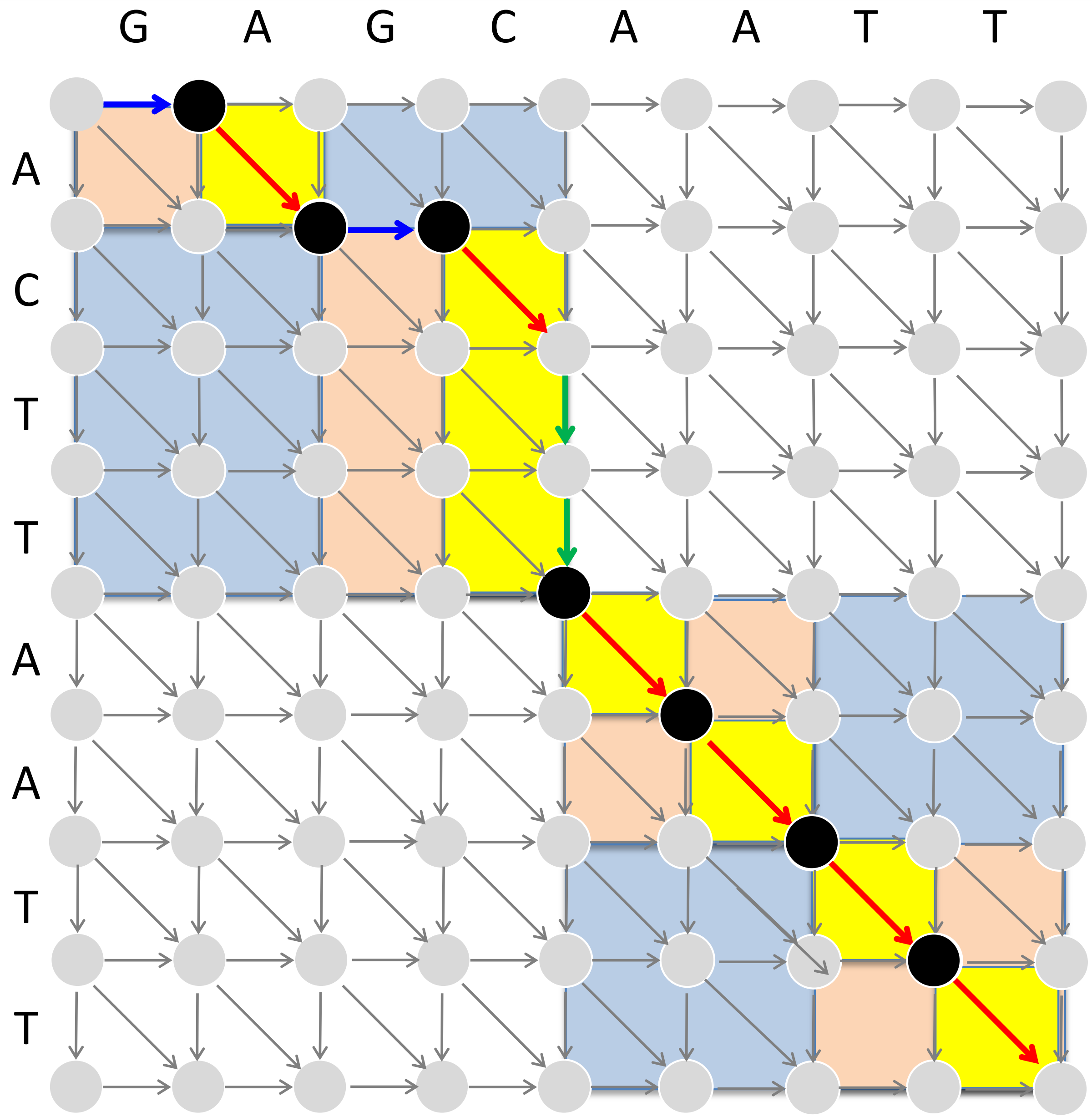


Each subproblem can be conquered in time proportional to its area:

area/8+area/8+ area/8+area/8= ***area/4***

How much time would it take to conquer 4 subproblems?

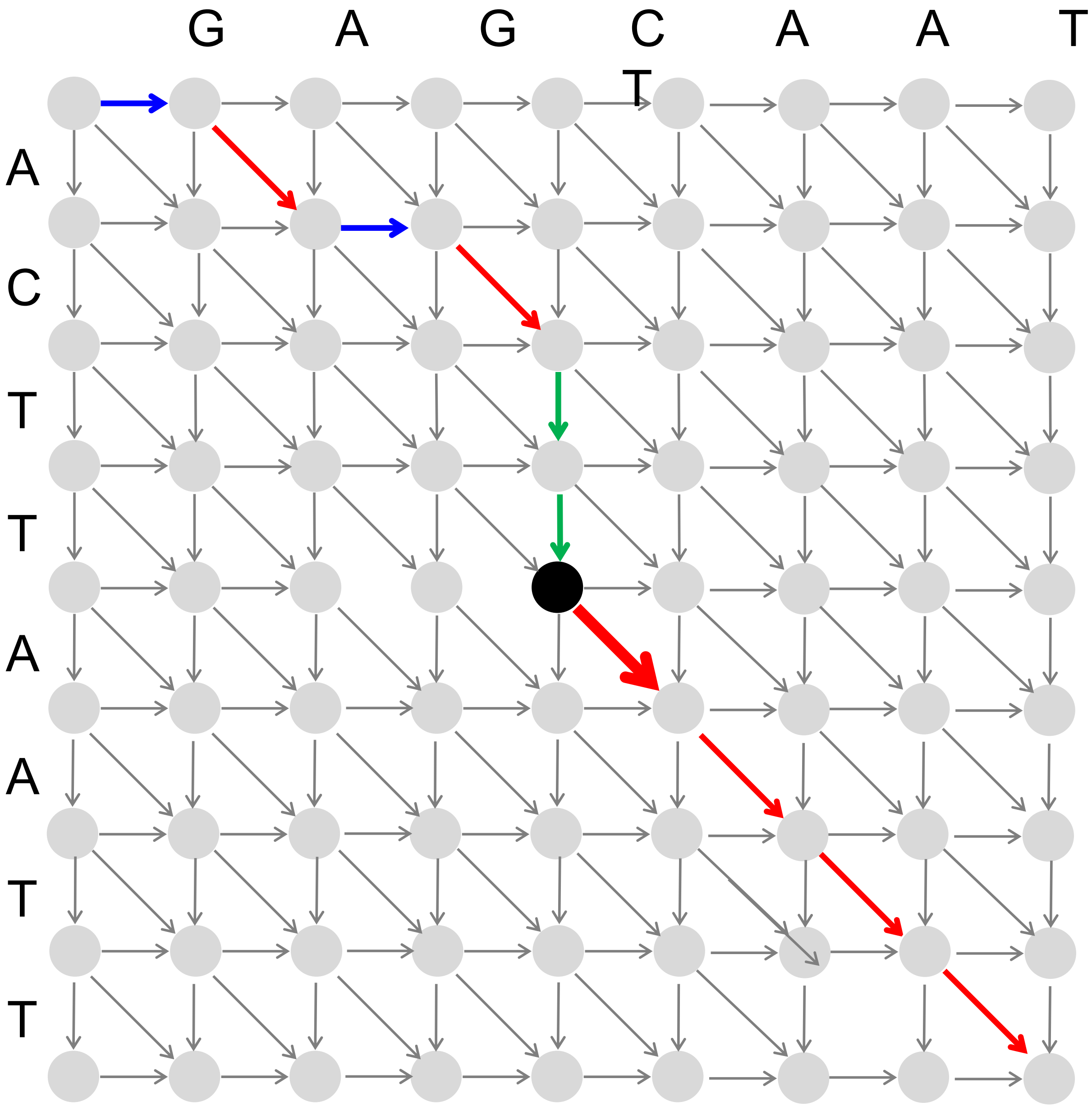
- O(*nm*+*nm/2*+*nm/4*) Time to Find **NEARLY ALL** Nodes!



area+
area/2
+area/4
+area/8
+area/16
+....+
<
2·area

How much time would it take to conquer ALL subproblems?

The Middle Edge

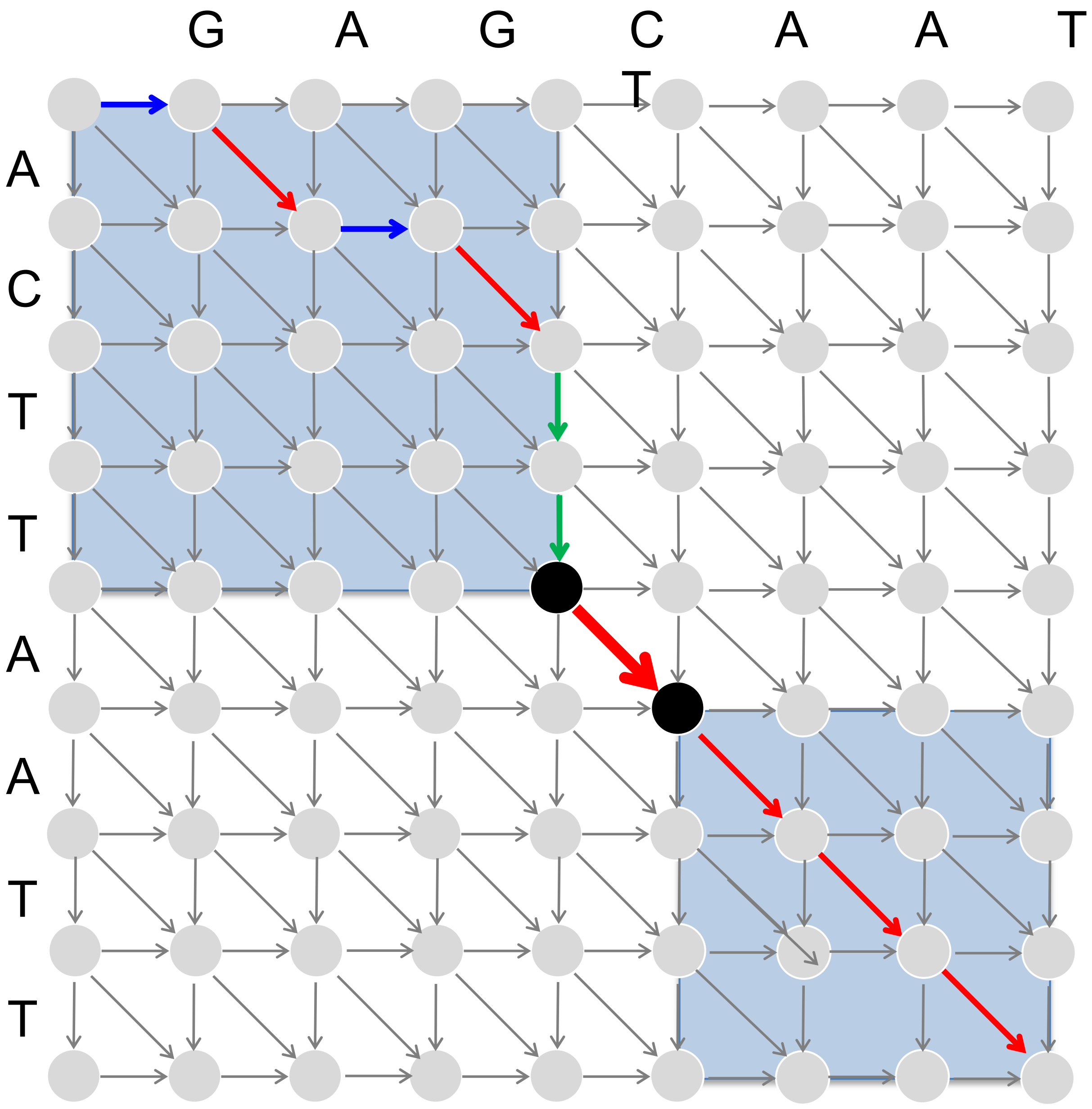


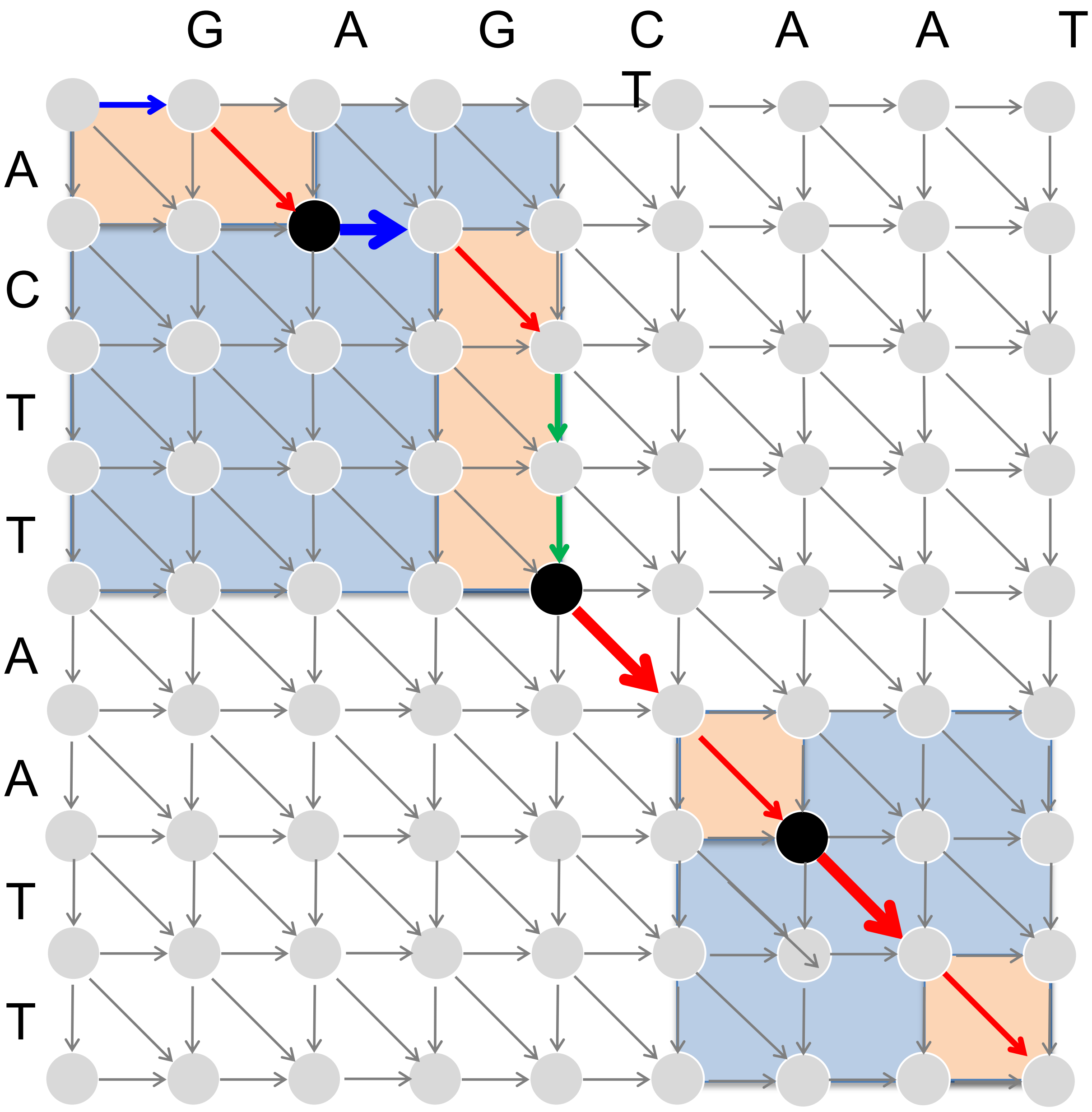
Middle Edge: an edge in an optimal alignment path starting at the middle node

The Middle Edge Problem

Middle Edge in Linear Space Problem. Find a middle edge in the alignment graph in linear space.

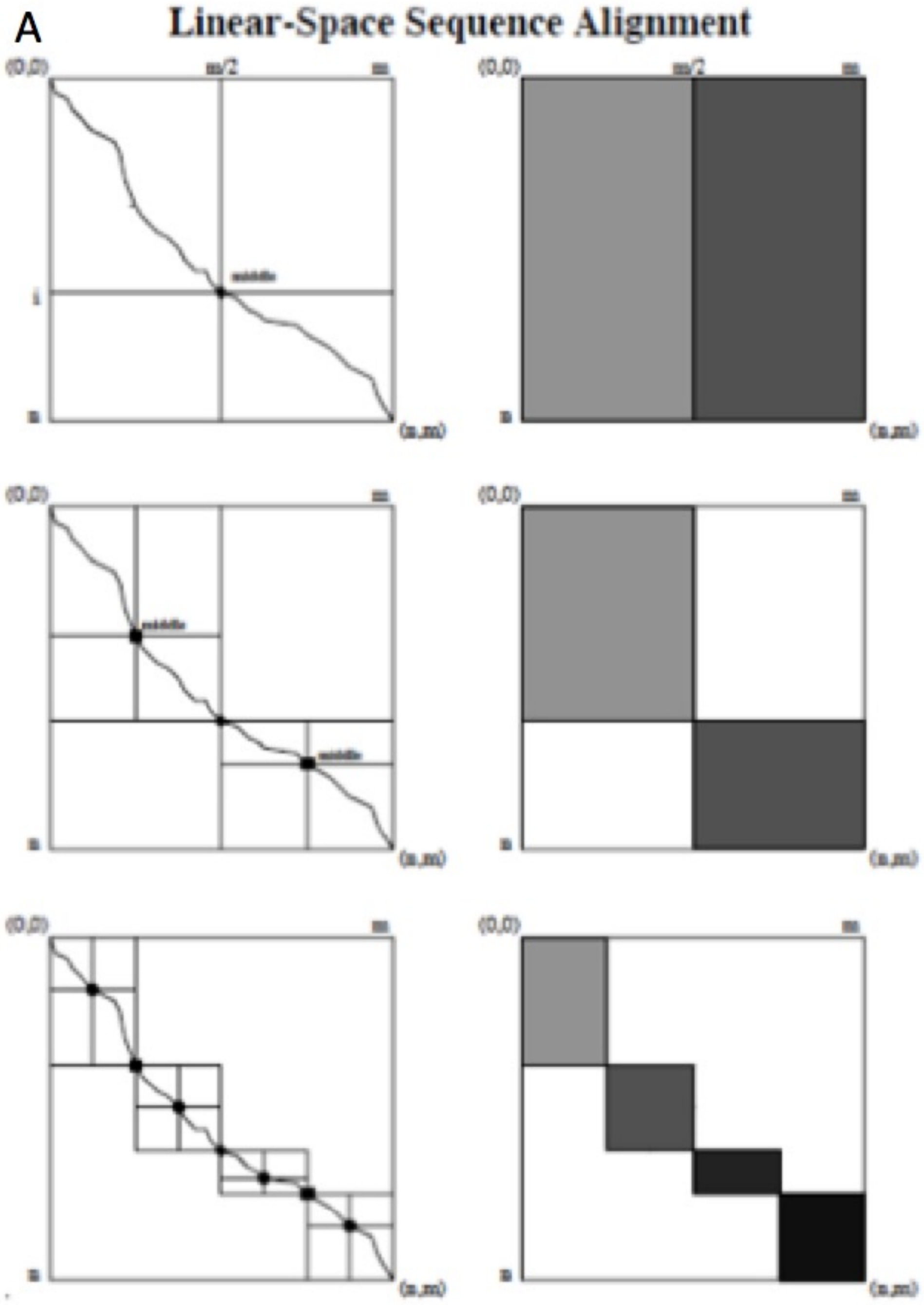
- **Input:** Two strings and matrix ***score***.
- **Output:** A middle edge in the alignment graph of these strings (as defined by the matrix ***score***).


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


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

Recursive **LinearSpaceAlignment**

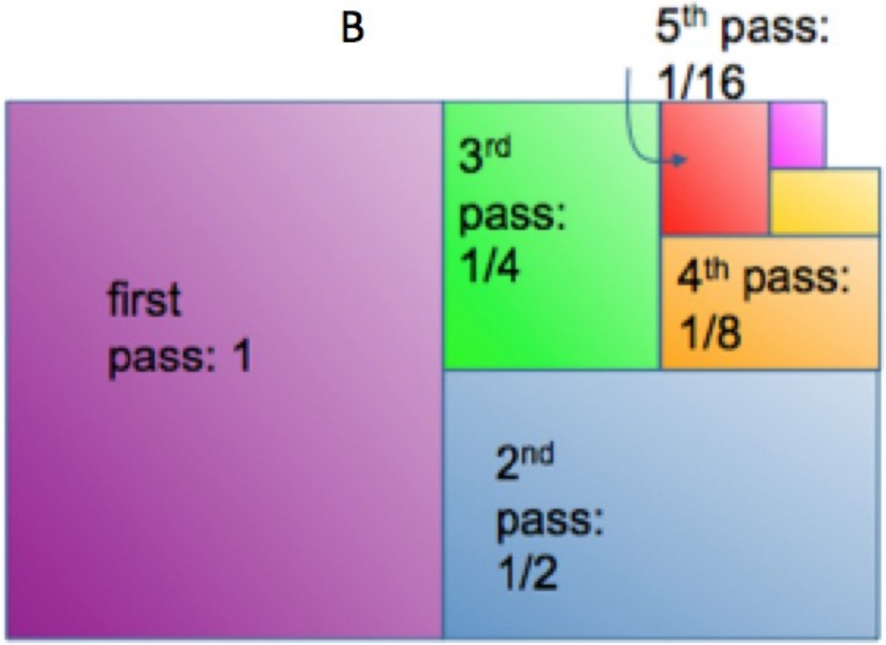
```
LinearSpaceAlignment(top,bottom,left,right)
  if left = right
    return alignment formed by bottom-top edges “↓”
  middle ← $\lfloor(left+right)/2\rfloor$
  midNode ← MiddleNode(top,bottom,left,right)
  midEdge ← MiddleEdge(top,bottom,left,right)
  LinearSpaceAlignment(top,midNode,left,middle)
  output midEdge
  if midEdge = “→“ or midEdge = “↘”
    middle ← middle+1
  if midEdge = “↓“ or midEdge = “↘”
    midNode ← midNode+1
  LinearSpaceAlignment(midNode,bottom,middle,right)
```



A: space complexity

B: time complexity

Total Time: ***area+area/2+area/4+area/8+area/16+…***



Can we compute the edit distance faster than O(nm)? yes: The Four Russians Technique (Arlazarov, V., Dinic, E., Kronrod, M., Faradžev, I.)

Key concept: Divide the input into very small parts, pre-compute the values using Dynamic Programming for all possible small parts and store them in a table. Then, speed up the dynamic programming via Table Lookup.

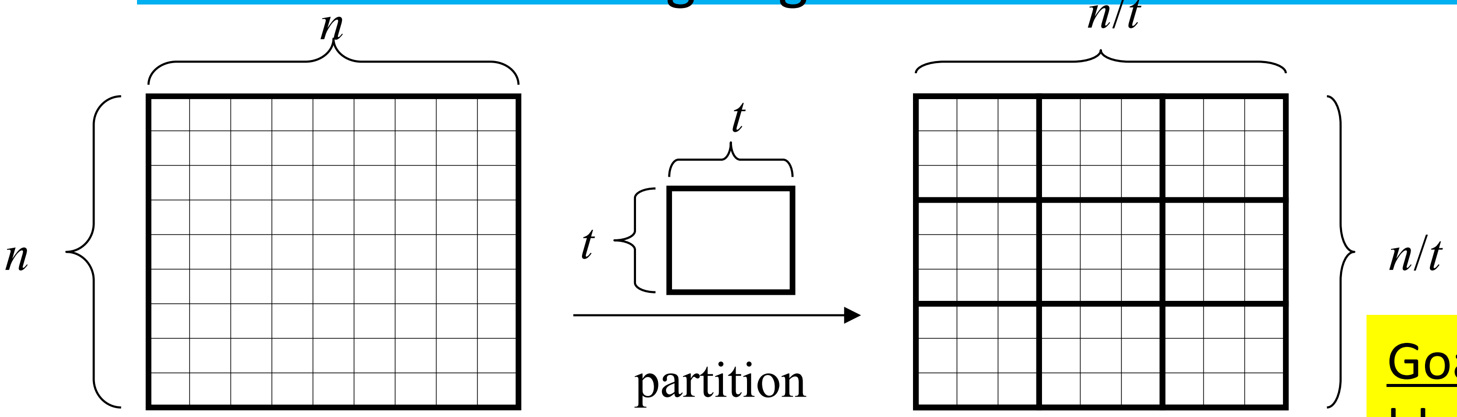
- Partition the *n* x *n* grid into blocks of size *t* x *t*
- We are comparing two sequences, each of size *n*, and each sequence is sectioned off into chunks, each of length *t*
- Sequence $\boldsymbol{u} = u_1 \ldots u_n$ becomes

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

and sequence $\boldsymbol{v} = v_1 \ldots v_\mathrm{n}$ becomes

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

Partitioning Alignment Grid into Blocks

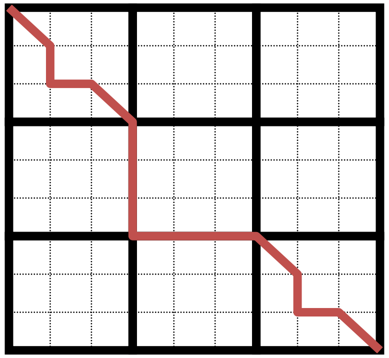


Goal: Find the longest block path through an edit graph

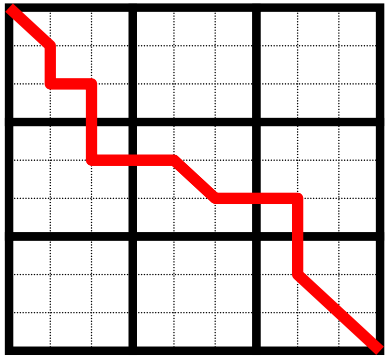
Input: Two sequences, ***u*** and ***v*** partitioned into blocks of size *t*. This is equivalent to an *n* x *n* edit graph partitioned into *t* x *t* subgrids

Output: The block alignment of ***u*** and ***v*** with the maximum score (longest block path through the edit graph

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



valid



invalid

Constructing Alignments within Blocks

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

$$s_{i,j} = \max \begin{Bmatrix} s_{i-1,j} - \sigma_{\text{block}} \\ s_{i,j-1} - \sigma_{\text{block}} \\ s_{i-1,j-1} - \beta_{i,j} \end{Bmatrix}$$

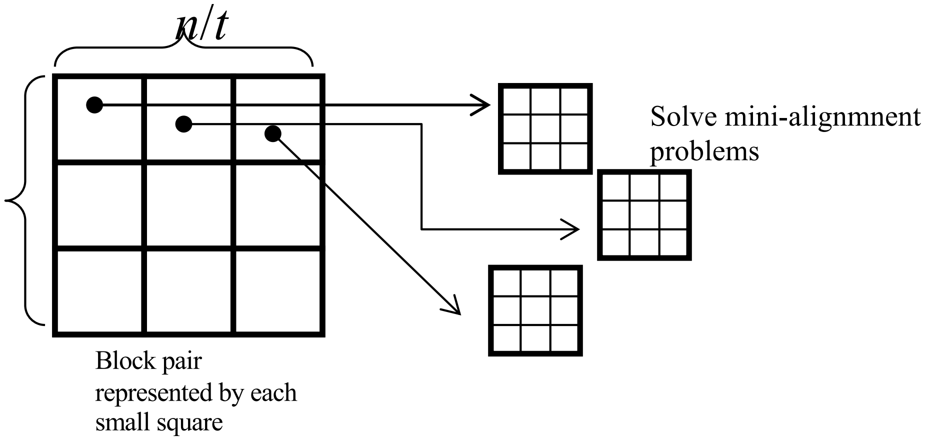
σ_{block} is the penalty for inserting or deleting an entire block

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

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

  (*n*/*t*) blocks of size *t*
- How many pairs of blocks for aligning the two sequences?
  (*n*/*t*) x (*n*/*t*)
- For each block pair, solve a mini-alignment problem of size *t* x *t*

Block Alignment Runtime


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

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

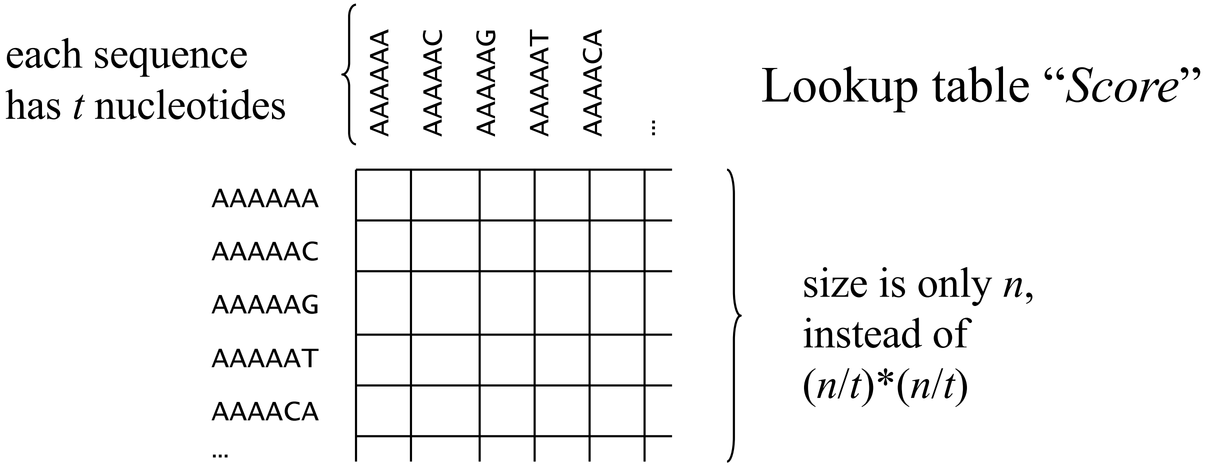
$$O(\ [n/t]*[n/t]) = O(n^2/t^2)$$

if we don't count the time to compute each $\beta_{i,j}$

- Computing all $\beta_{i,j}$ requires solving $(n/t)*(n/t)$ mini block alignments, each of size $(t*t)$
- Computing all $\beta_{i,j}$ takes time $O([n/t]*[n/t]*t*t) = O(n^2)$
- How do we speed this up?

Four Russians Technique

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



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

The new lookup table Score is indexed by a pair of t-nucleotide strings

Four Russians Speedup Runtime

We can divide up the grid into blocks and run dynamic programming only on the corners of these blocks. In order to speed up the mini-alignment calculations to under n^2, we create a lookup table of size n, which consists of all scores for all t-nucleotide pairs.

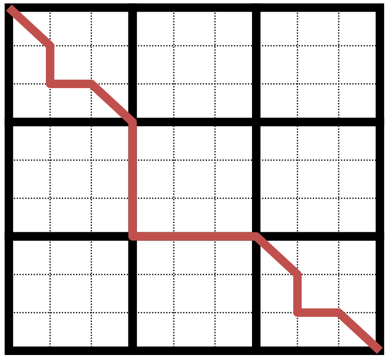
Since computing the lookup table *Score* of size n takes O(n) time, the running time is mainly limited by the $(n/t)*(n/t)$ accesses to the lookup table;

Each access takes O(log$n$) time. Overall running time:

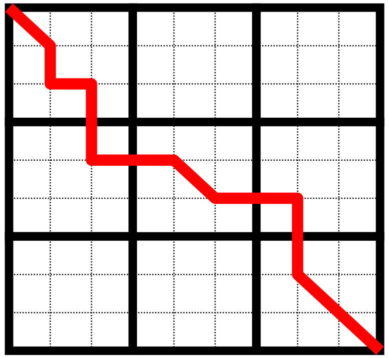
O($[n^2/t^2]*\log n$); Since $t = \log n$, substitute in: O($[n^2/\{\log n\}^2]*\log n$) $\geq$ O($n^2/\log n$).

More explanations: Four Russians Speedup for LCS

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

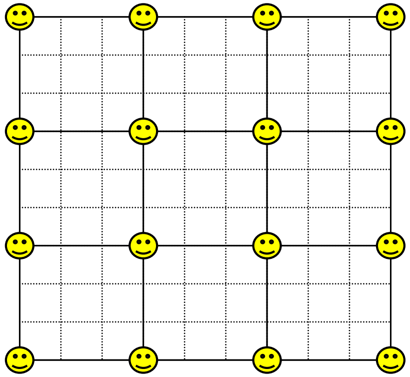


block alignment

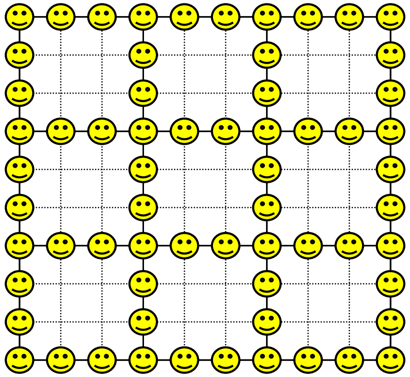


longest common subsequence

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



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



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

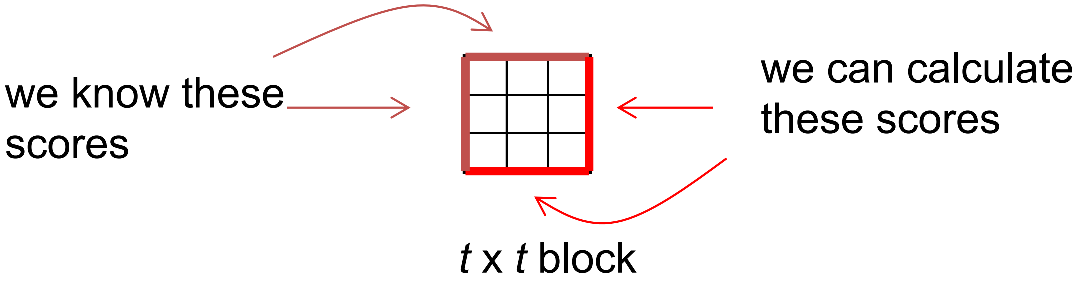
Traversing Blocks for LCS

Given alignment scores $s_{i,*}$ in the first row and scores $s_{*,j}$ in the first column of a *t* x *t* mini square, compute alignment scores in the last row and column of the minisquare.

To compute the last row and the last column score, we use these 4 variables:

- alignment scores $s_{i,*}$ in the first row
- alignment scores $s_{*,j}$ in the first column
- substring of sequence u in this block (4^t possibilities)
- substring of sequence v in this block (4^t possibilities

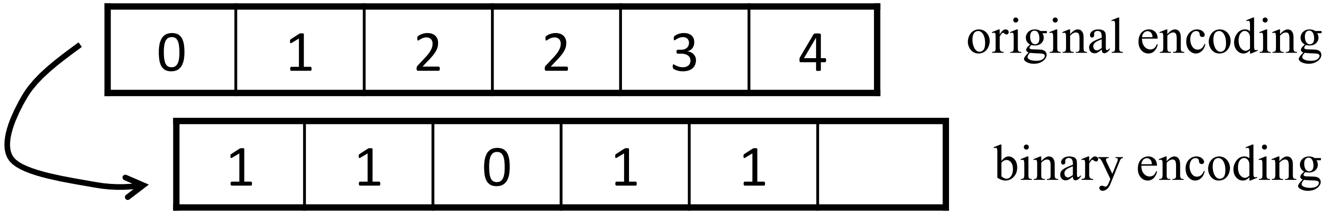
If we used this to compute the grid, it would take quadratic, O(n^2) time, but we want to do better.



Four Russians Speedup

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

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



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

Reducing Lookup Table Size

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

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

The Four-Russian Algorithm

Example

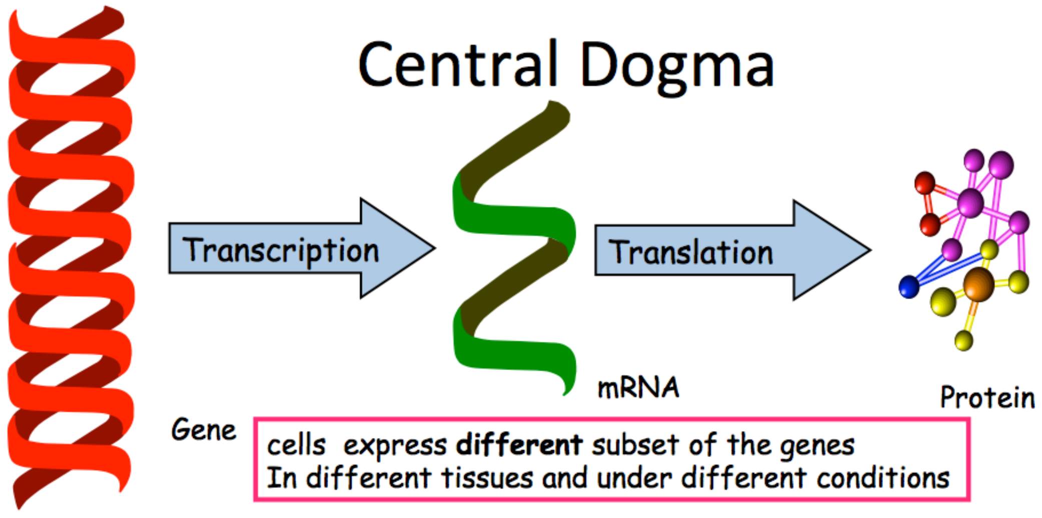
x = AACT

y = CACT

		A	A	C	T	
		0	1	-1	-1	
C	0	5	6	5	4	0
A	0	5	5	6	5	1
C	-1	4	5	5	6	1
T	1	5	5	6	5	-1
		0	0	1	-1	

		A	A	C	T	
		0	1	-1	-1	
C	0	1	2	1	0	0
A	0	1	1	2	1	1
C	-1	0	1	1	2	1
T	1	1	1	2	1	-1
		0	0	1	-1	

Central Dogma and Genetic Code



cells express **different** subset of the genes
In different tissues and under different conditions

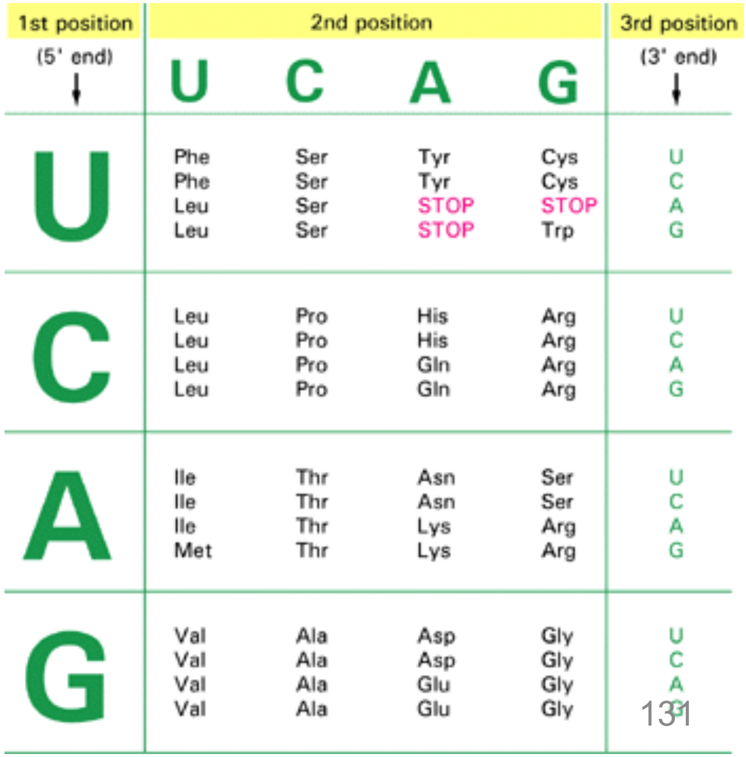
The central dogma of molecular biology explains the flow of genetic information, from DNA to RNA, to make a functional product, a protein.

The central dogma suggests that DNA contains the information needed to make all of our proteins, and that RNA is a messenger that carries this information to the ribosome

In transcription, the information in the DNA of every cell is converted into small, portable RNA messages.

During translation, these messages travel from where the DNA is in the cell nucleus to the ribosomes where they are ‘read’ to make specific proteins using a genetic code (right).

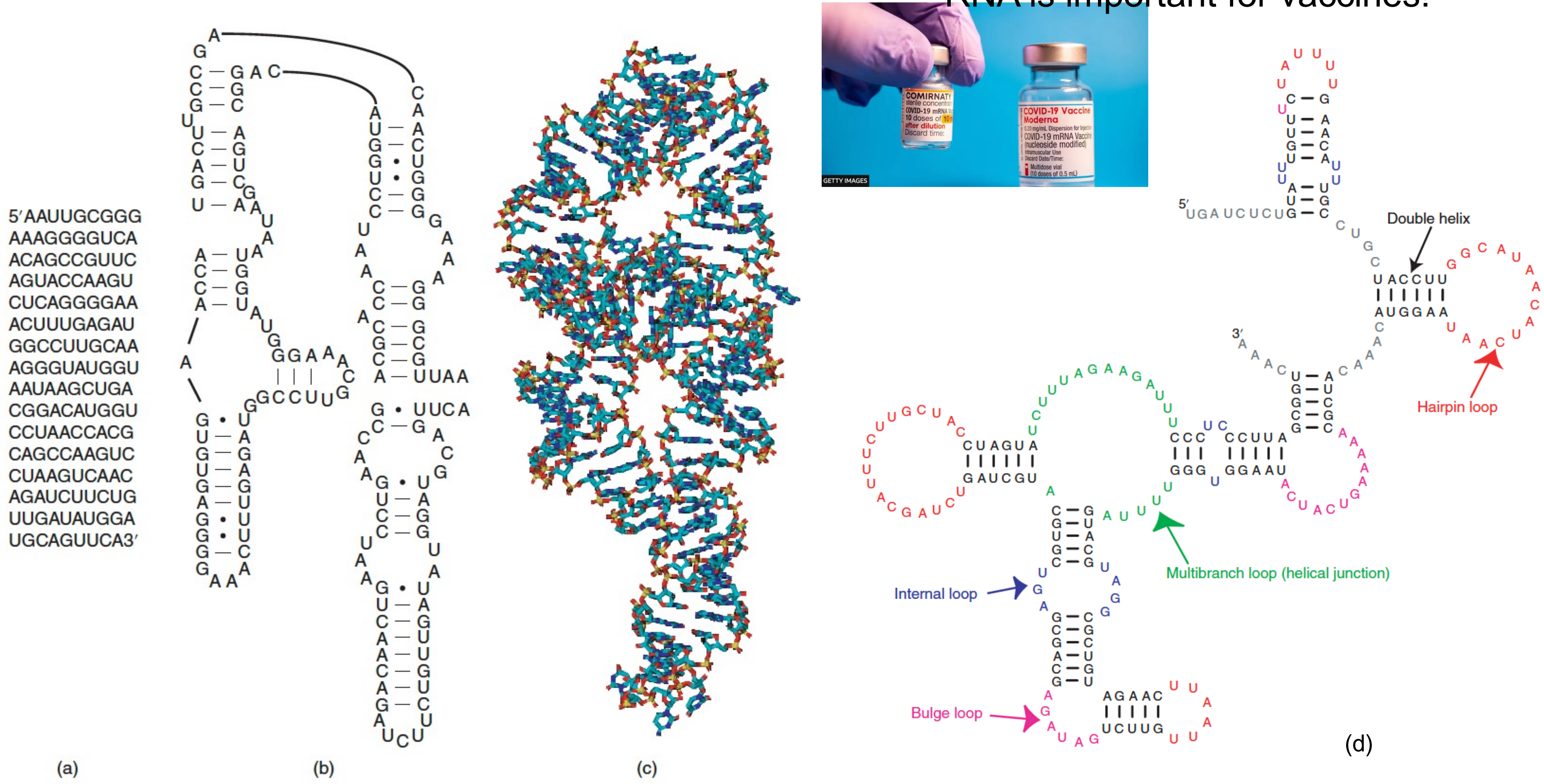
Gene expression is a tightly regulated process that increases or decreases the amount of proteins made.



1st position (5' end) ↓	2nd position U	C	A	G	3rd position (3' end) ↓
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	STOP	STOP	A
	Leu	Ser	STOP	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

Can we predict the RNA secondary structure from sequence? Sort of self alignment of one molecule which is called folding

RNA is important for vaccines!



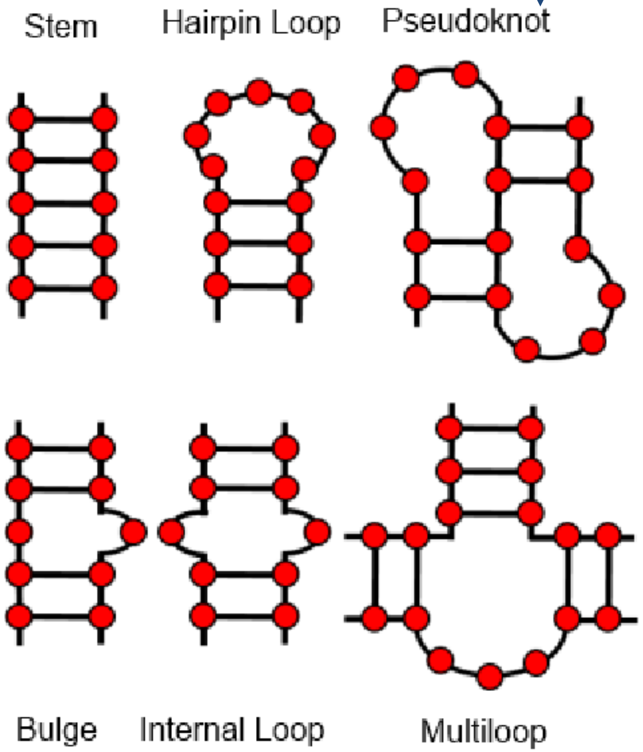
The three levels of organization of RNA structure a) sequence (1 D); b) secondary structure (2D);

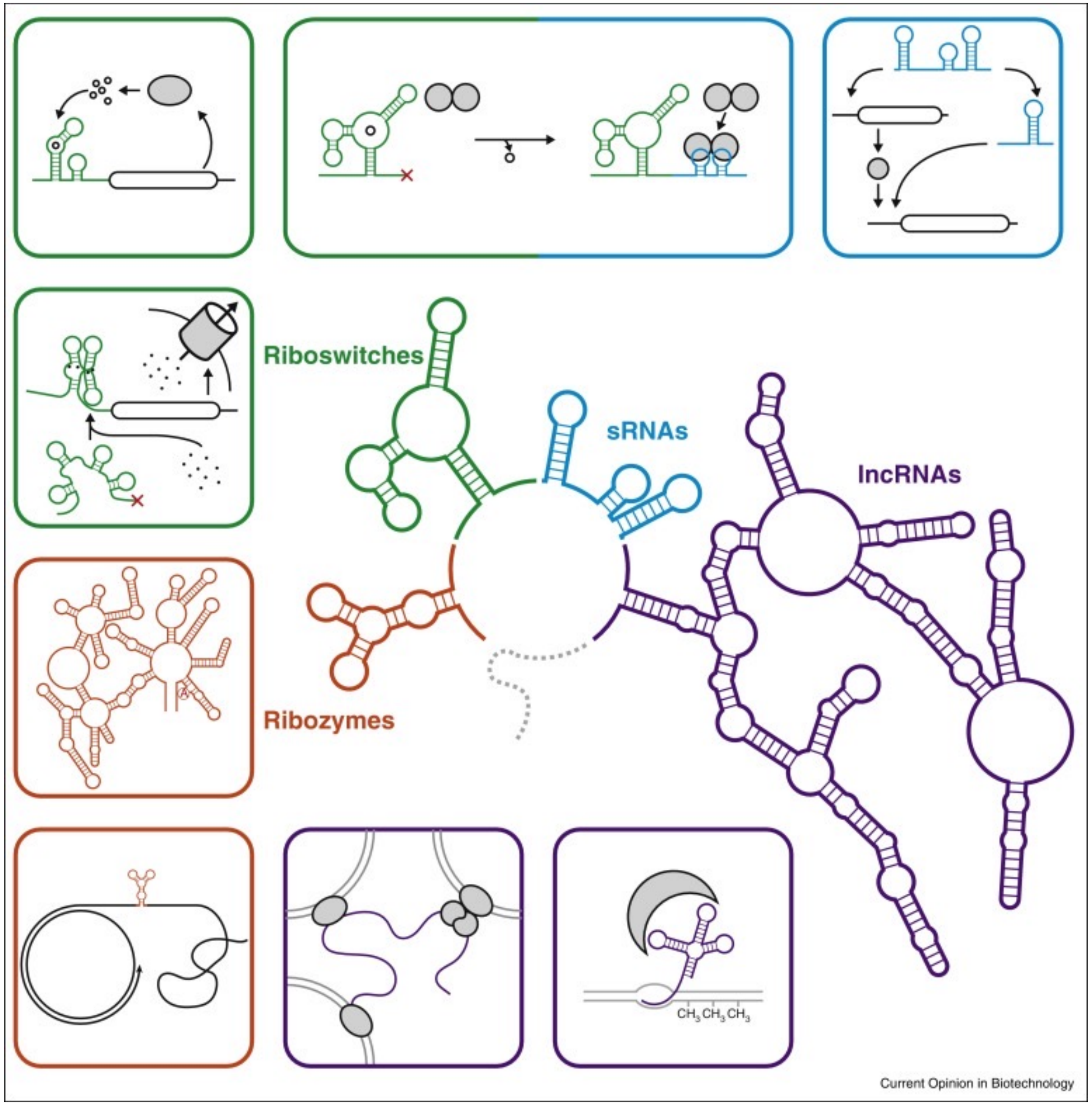
c) Tertiary structure (3D); We want to scale up 1D-> 2D -> 3D

d) figure on the right shows 2D could be very complex to decipher.

Biologists need algorithms to compute RNA local folding as this could highlight important cell functions

Right: RNA has many foldings -> many functions in the cell;
Below: The basic local foldings



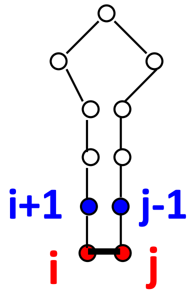


Source: https://www.sciencedirect.com/science/article/pii/S0958166916301082#fig0020

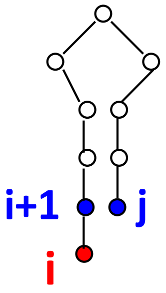
Biologists need algorithms to compute RNA local folding as this could highlight important cell functions

- Secondary Structure :
  - Set of paired positions on interval [*i,j*]
  - This tells which bases are paired in the subsequence from x_i to x_j
- Every optimal structure can be built by extending optimal substructures.
- Suppose we know all optimal substructures of length less than *j-i+1.*
  The optimal substructure for [*i,j*] must be formed in one of four ways:
  1. *i,j* paired
  2. *i* unpaired
  3. *j* unpaired
  4. combining two substructures

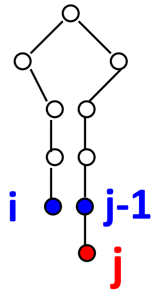
  Note that each of these consists of extending or joining substructures of length less than *j-i+1.*



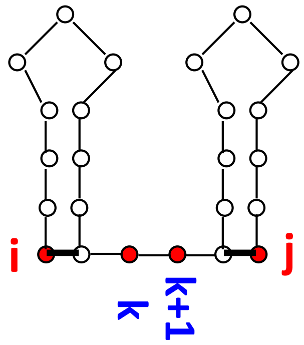
i,j pair



i unpaired



j unpaired



bifurcation

RNA Secondary Structure: The Nussinov Folding Algorithm

Nussinov, R., Pieczenik, G., Griggs, J. R. and Kleitman, D. J. (1978). Algorithms for loop matchings, SIAM J. Appl. Math

Example: GGGAAAUCC

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

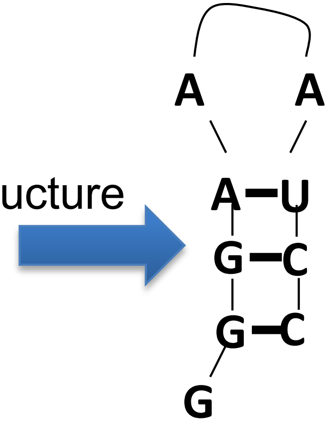
$$Initialisation\ \gamma(\mathrm{i},\mathrm{i}-1)=0\ \&\ \gamma(\mathrm{i},\mathrm{i})=0$$

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

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

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

final structure



Ruth Nussinov

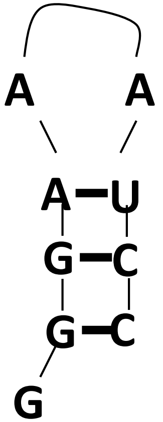
j →

i ↓

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

Nussinov Folding Algorithm: After scores for subsequences of length 2

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



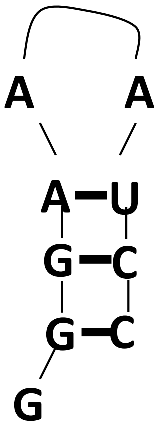
j →

i ↓

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

Nussinov Folding Algorithm: After scores for subsequences of length 3

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



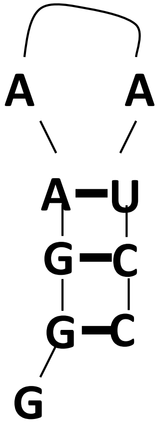
j →

i ↓

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

Nussinov Folding Algorithm
After scores for subsequences of length 4

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



j →

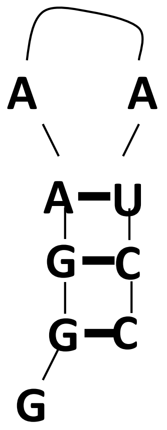
i ↓

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

Two optimal substructures for same subsequence

Nussinov Folding Algorithm
After scores for subsequences of length 5

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



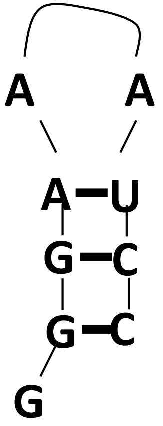
j →

i ↓

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

Nussinov Folding Algorithm
After scores for subsequences of length 6

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

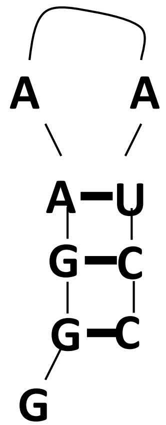


j →

i ↓	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	**0**			
G	0	0	0	0	0	0	**1**		
G		0	0	0	0	0	1	**2**	
A			0	0	0	0	1	1	**1**
A				0	0	0	1	1	1
A					0	0	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

Nussinov Folding Algorithm
After scores for subsequences of length 7

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



j →

i ↓

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

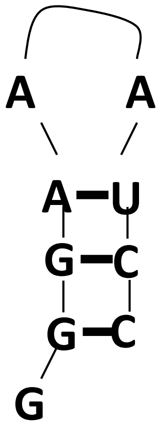
Nussinov Folding Algorithm
After scores for subsequences of length 8

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

j →

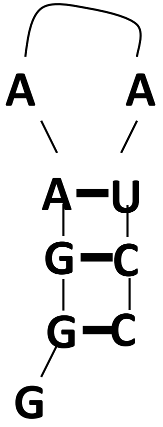
i ↓

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



Nussinov Folding Algorithm
After scores for subsequences of length 9

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

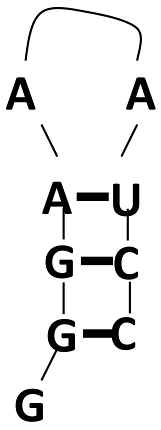


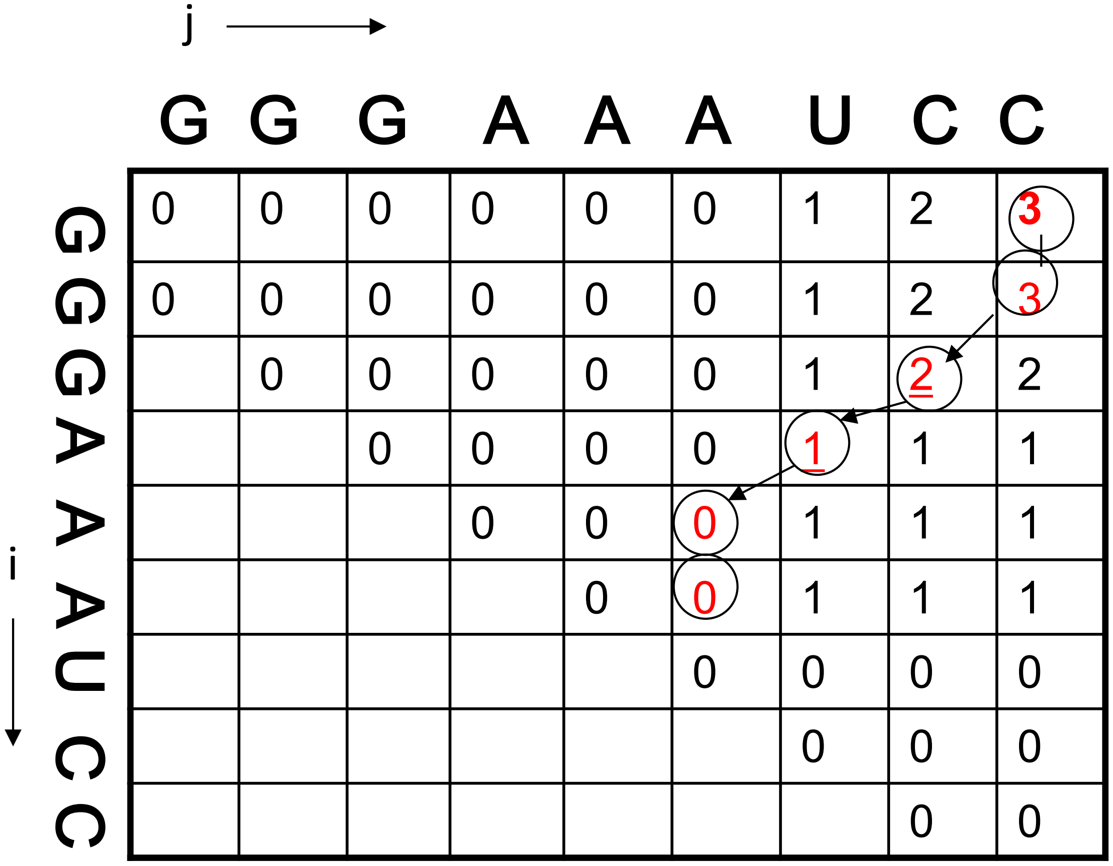
j →

i ↓

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

Nussinov Folding Algorithm Traceback

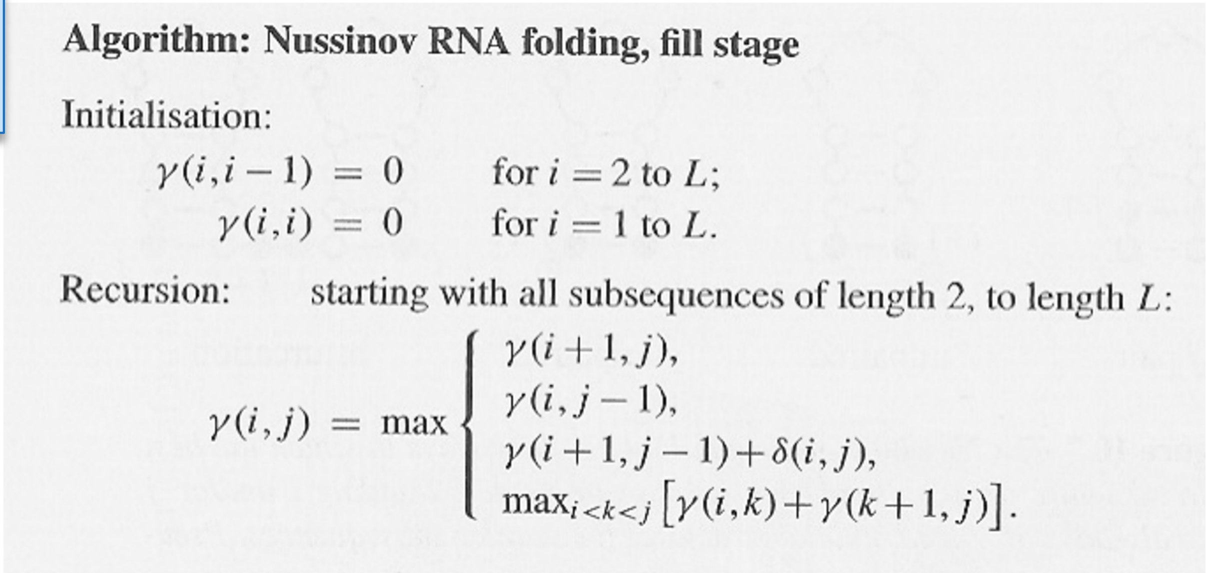




	G	G	G	A	A	A	U	C	C
G	0	0	0	0	0	0	1	2	3
G	0	0	0	0	0	0	1	2	3
G		0	0	0	0	0	1	2	2
A			0	0	0	0	1	1	1
A				0	0	0	1	1	1
A					0	0	1	1	1
U						0	0	0	0
C							0	0	0
C								0	0

Example and RECAP

Nussinov algorithm: fill-stage



Algorithm: Nussinov RNA folding, fill stage

Initialisation:

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

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

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

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

Scoring system:

δ(i,j) = 1 for all RNA Watson-Crick base-pairs including G-U else *δ(i,j)* = 0.

Blue: addition of unpaired base 3 or 7

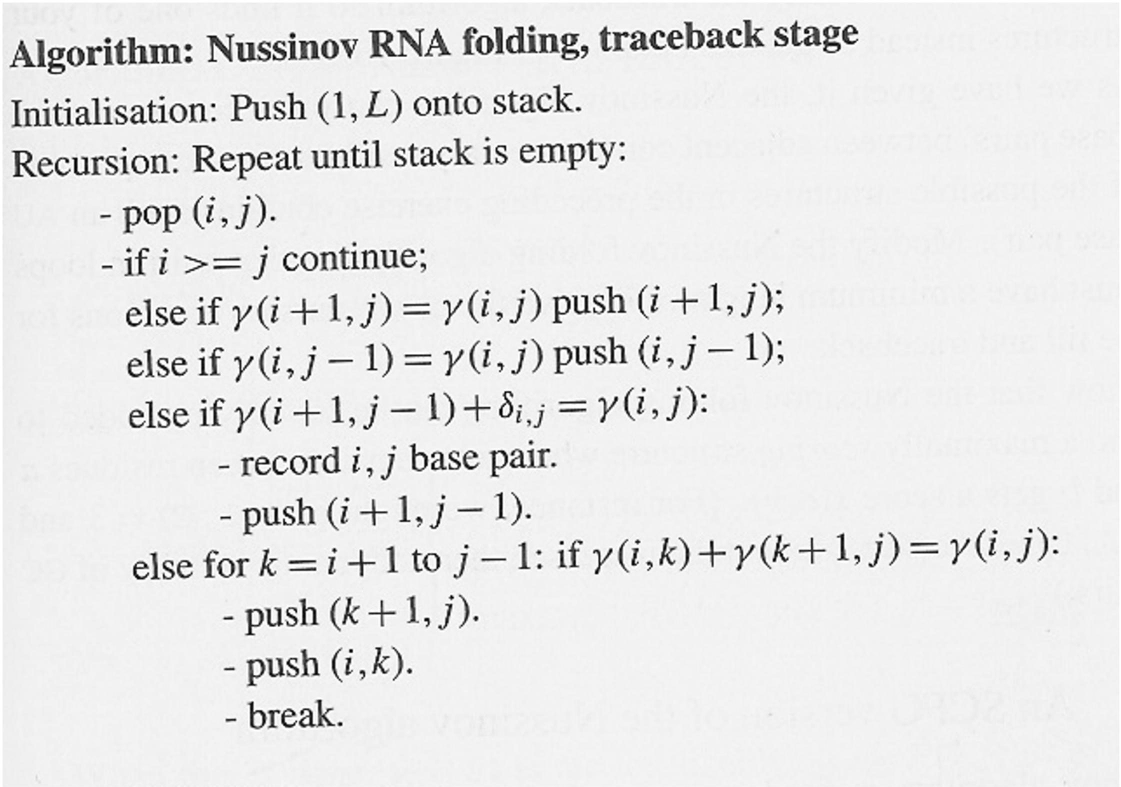
Green: addition of paired bases 1,7

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

Example and RECAP

Nussinov algorithm: trace-back

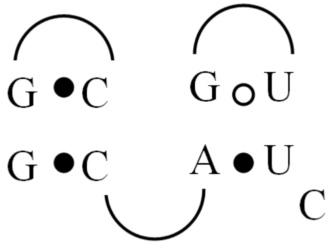
		G	G	C	C	A	G	U	U	C
		1	2	3	4	5	6	7	8	9
G	1	0	0	1	2	2	2	3	4	4
G	2	0	0	1	1	1	2	2	3	3
C	3		0	0	0	0	1	1	2	2
C	4			0	0	0	1	1	2	2
A	5				0	0	0	1	2	2
G	6					0	0	1	1	1
U	7						0	0	0	0
U	8							0	0	0
C	9								0	0



Algorithm: Nussinov RNA folding, traceback stage

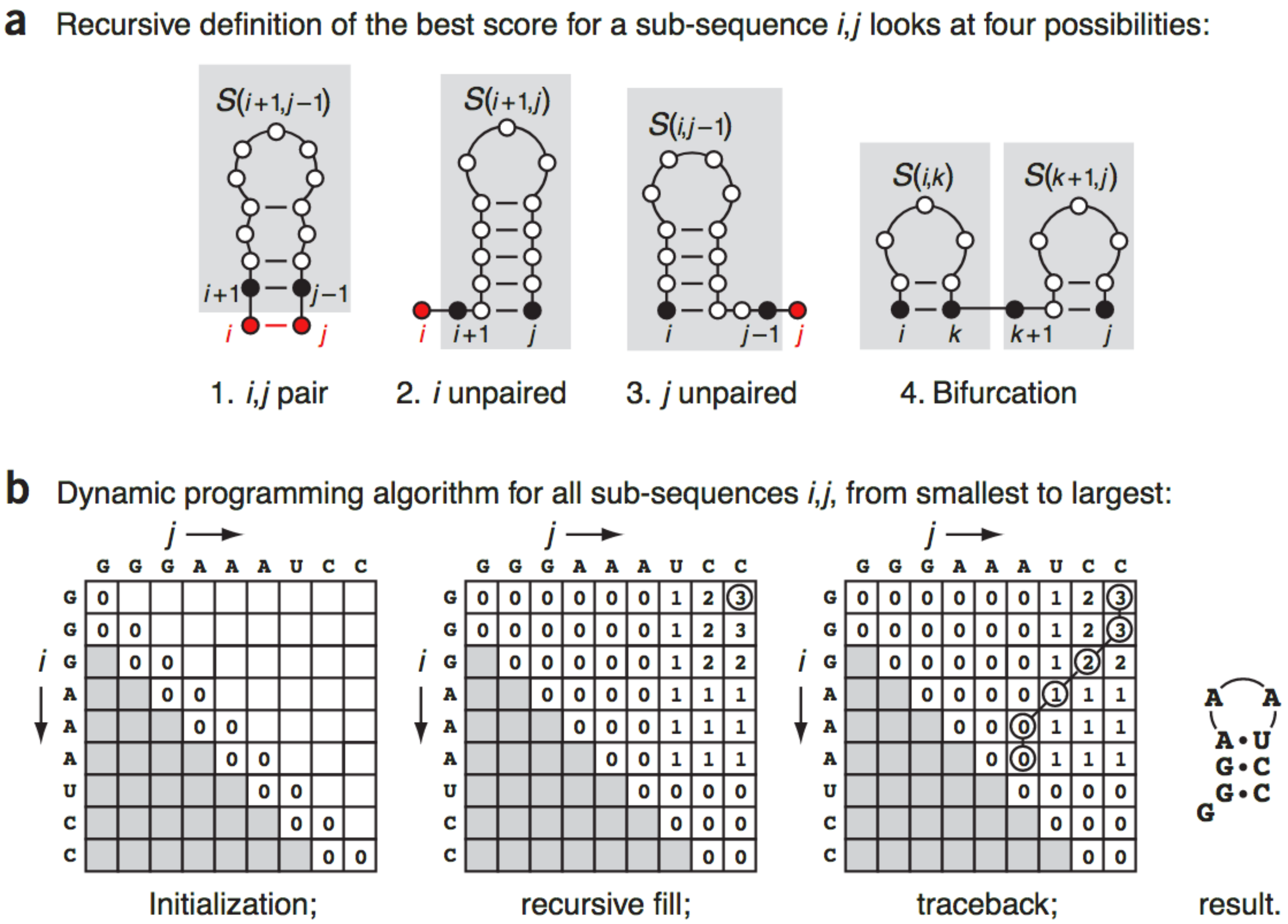
```
Initialisation: Push $(1, L)$ onto stack.
Recursion: Repeat until stack is empty:
    - pop $(i, j)$.
    - if $i >= j$ continue;
      else if $\gamma(i+1, j) = \gamma(i, j)$ push $(i+1, j)$;
      else if $\gamma(i, j-1) = \gamma(i, j)$ push $(i, j-1)$;
      else if $\gamma(i+1, j-1) + \delta_{i,j}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

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



Final structure

RECAP



a Recursive definition of the best score for a sub-sequence *i,j* looks at four possibilities:

b Dynamic programming algorithm for all sub-sequences *i,j*, from smallest to largest:

Initialization;

	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

recursive fill;

	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

traceback;

	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

result.

Dynamic programming algorithm for RNA secondary structure prediction. (**a**) The four cases examined by the dynamic programming recursion. Red dots mark the bases being added onto previously calculated optimal sub-structures (*i,j* pair, unpaired *i* or unpaired *j*). Gray boxes are a reminder that the recursion tabulates the *score* of the smaller optimal sub-structures, not the structures themselves. Example sub-structures are shown in the gray boxes solely as examples. (b) The dynamic programming algorithm in operation, showing the matrix *S*(*i,j*) for a sequence GGGAAAUCC after initialization, after the recursive fill, and after an optimal structure with three base pairs has been traced back.

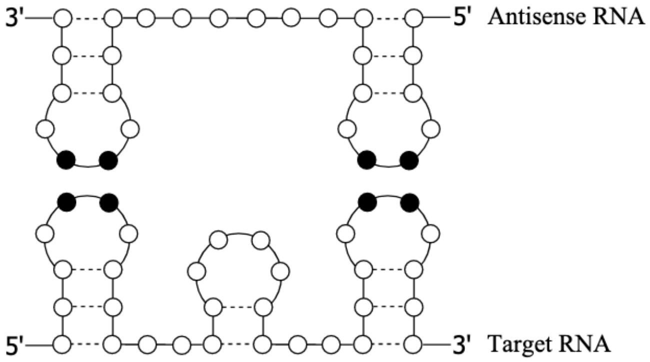
Complexity

$$\boxed{\textit{Initialisation}\ \gamma(\mathrm{i},\mathrm{i}-1)=0\ \&\ \gamma(\mathrm{i},\mathrm{i})=0}$$

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

Complexity: 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) in time and O(n^2) in space.

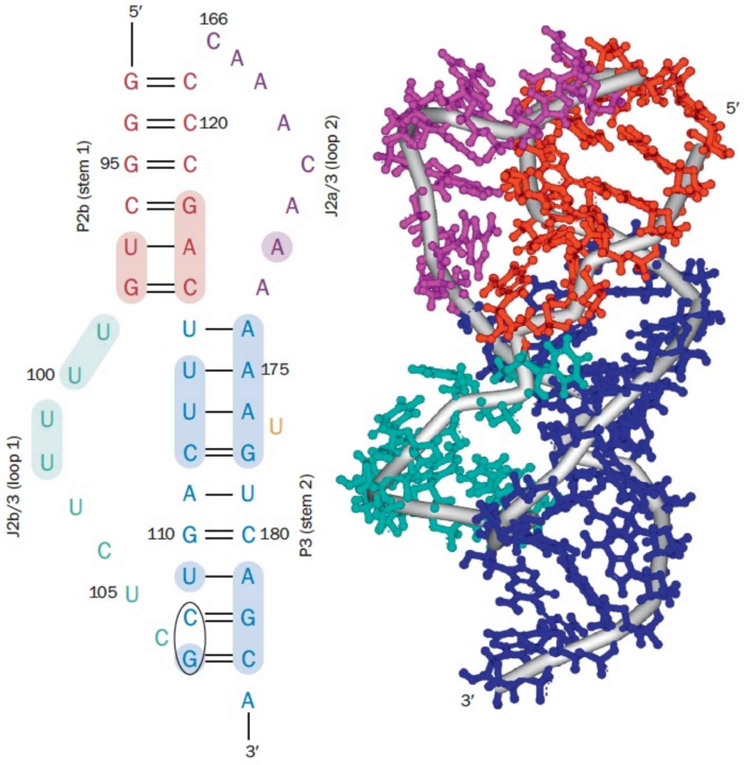
Open problems



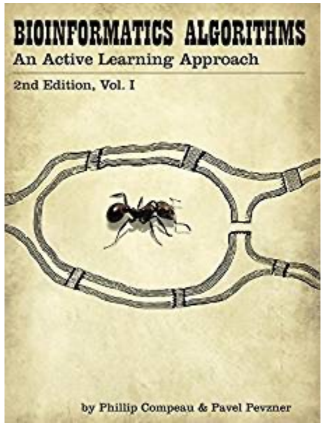
Some noncoding RNAs, called antisense RNAs, aim at inhibiting their target RNA function through base complementary binding; several kissing hairpin structures (left) caused by loop-loop interaction have been reported.

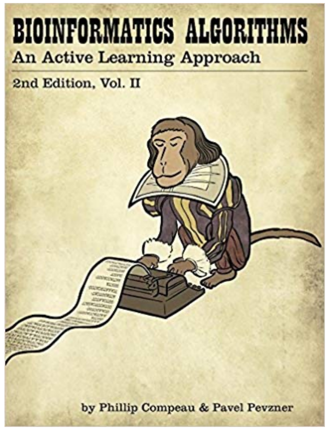
Add energy

3D structure

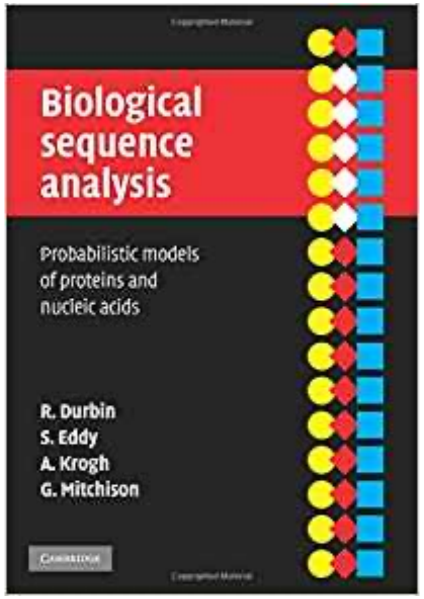


Reference for this section


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. I
by Phillip Compeau & Pavel Pevzner


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. II
by Phillip Compeau & Pavel Pevzner

- Chapter 5 Vol 1 (pag 226-291)


Biological sequence analysis
Probabilistic models of proteins and nucleic acids
R. Durbin
S. Eddy
A. Krogh
G. Mitchison
CAMBRIDGE

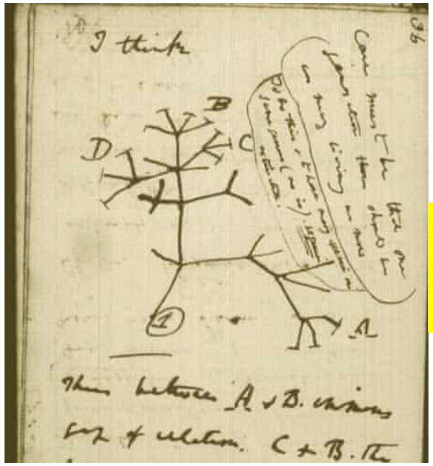
Pag 261-277

Section 3

Algorithms to build Trees

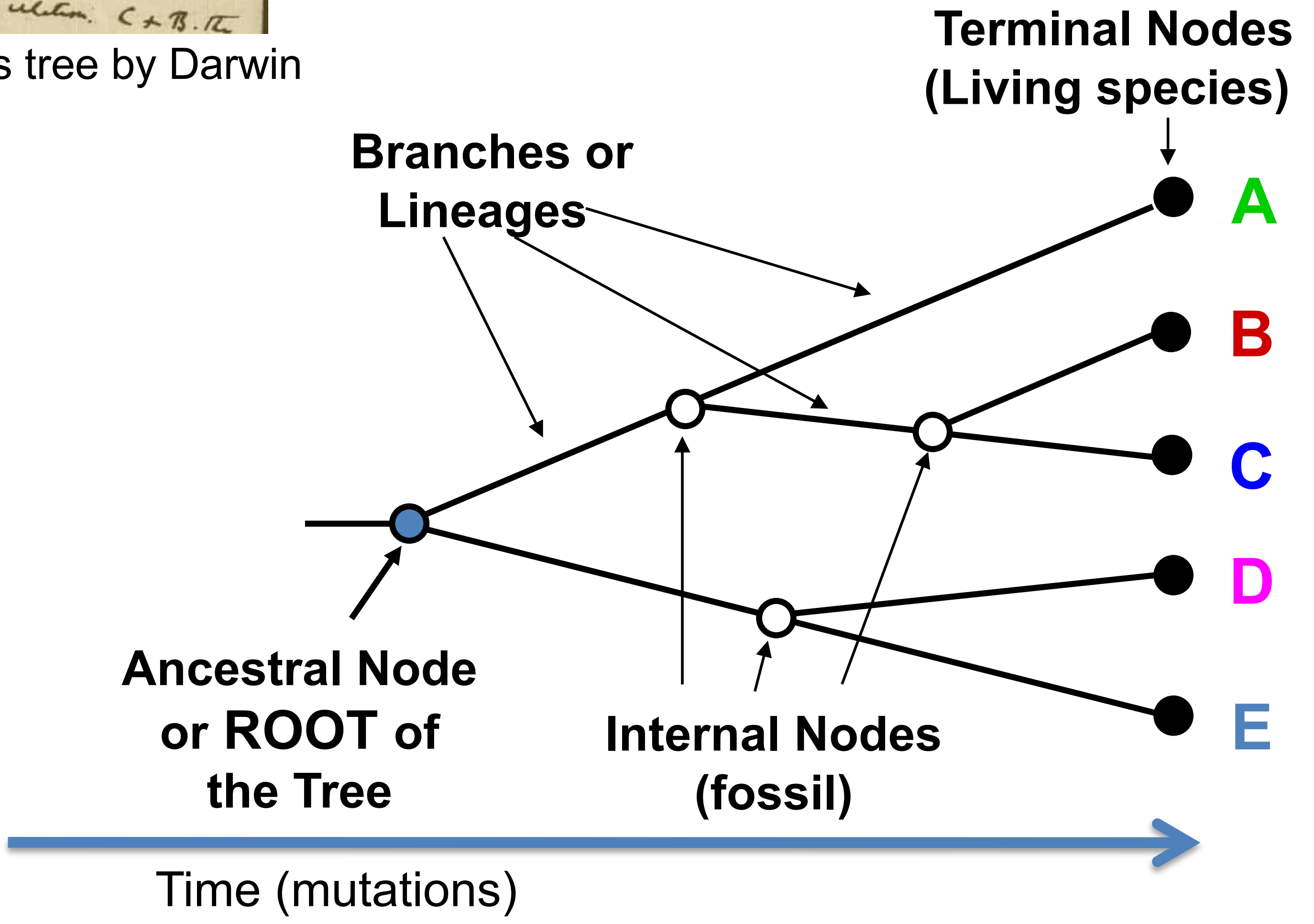
- Additive Phylogeny
- Using Least-Squares to Construct Distance-Based Phylogenies
- Ultrametric Evolutionary Trees
- The Neighbor-Joining Algorithm
- Character-Based Tree Reconstruction
- The Small Parsimony Problem
- The Large Parsimony Problem
- Back to the alignment: progressive alignment

What it is in a tree



species tree by Darwin

Why is important for biologists:Nothing in Biology Makes Sense Except in the Light of Evolution (Dobzhansky, 1964)



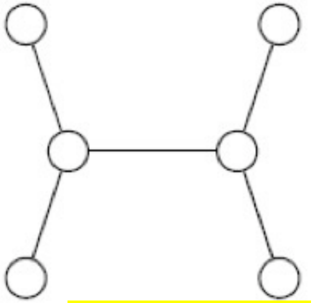
Terminal Nodes (Living species)

Branches or Lineages

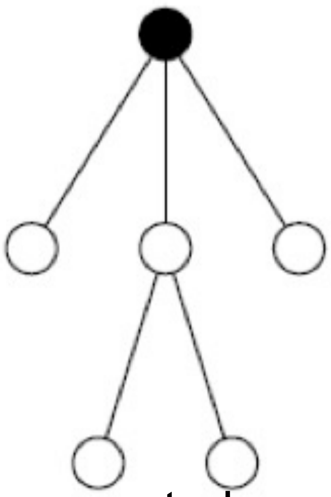
Ancestral Node or ROOT of the Tree

Internal Nodes (fossil)

Time (mutations)



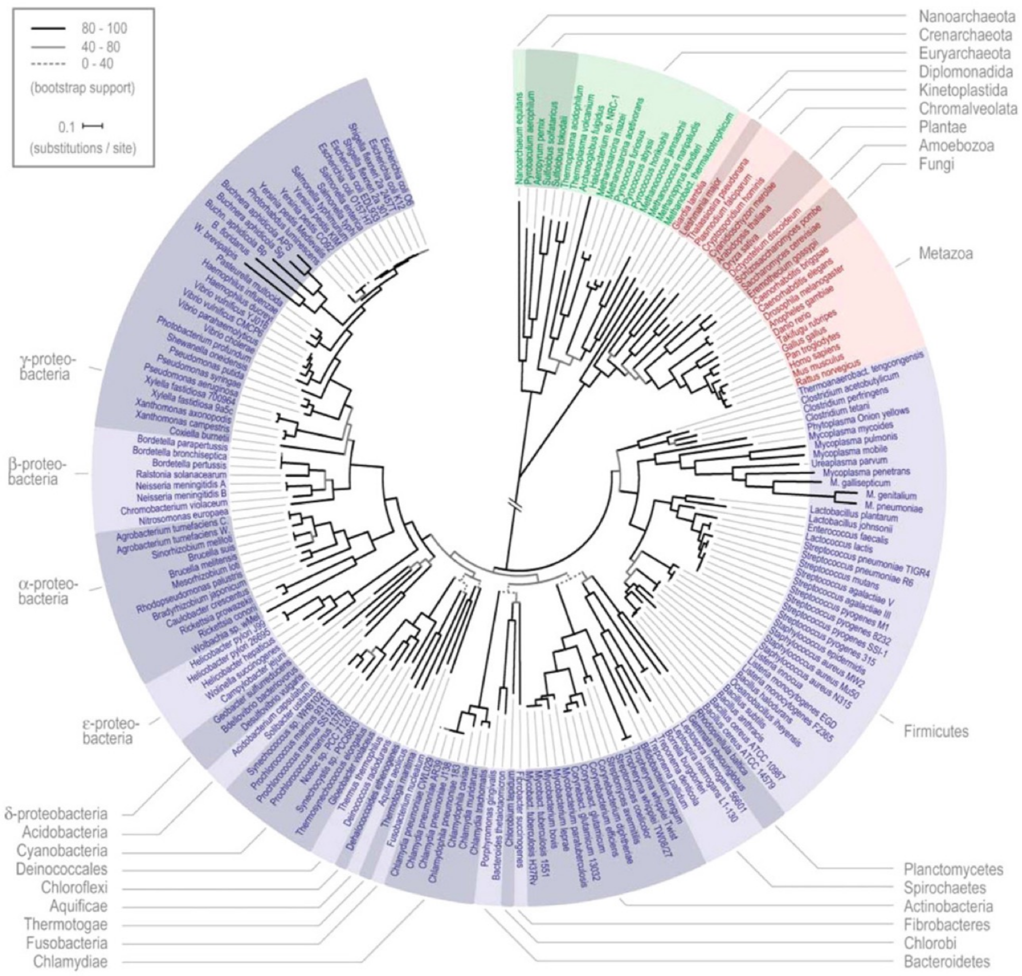
unrooted



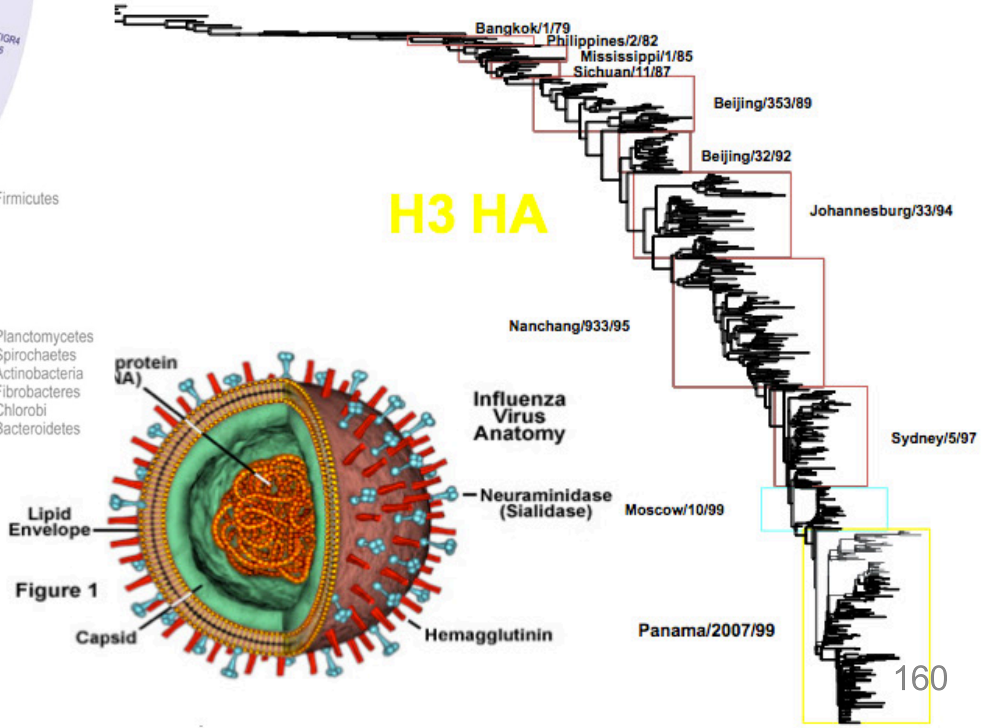
rooted

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

Why biologists need algorithms to build trees



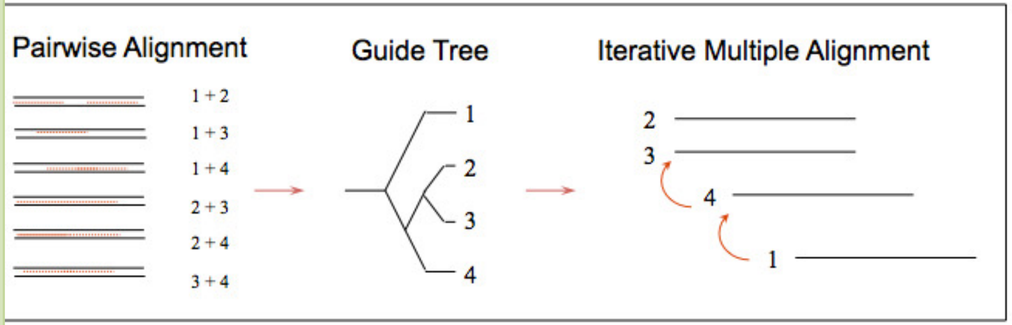
Reconstruction of evolutive patterns: tree of life based on mitochondrial sequences



tracing influenza strain variations
Based on variations in hemagglutinin sequence

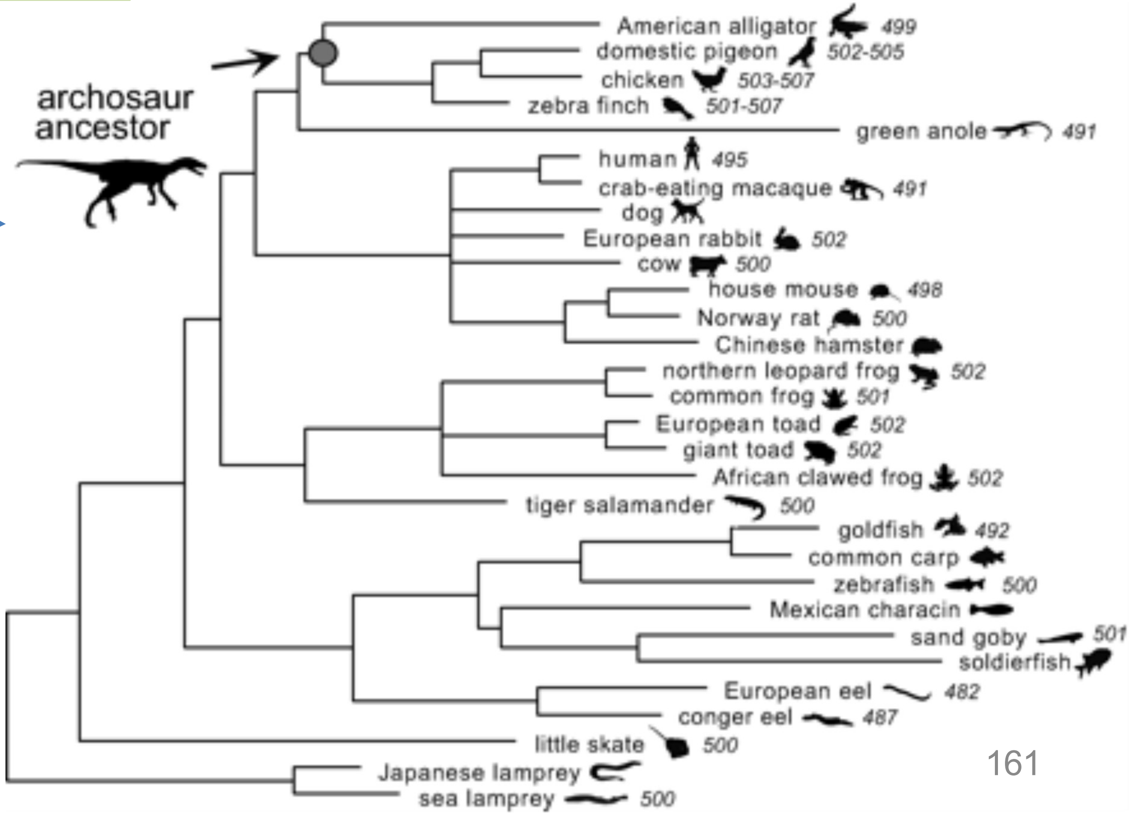
Why biologists need algorithms to build trees

We can use a tree to guide a multiple sequence alignment. The sequence of genes and proteins are conserved in nature. 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.



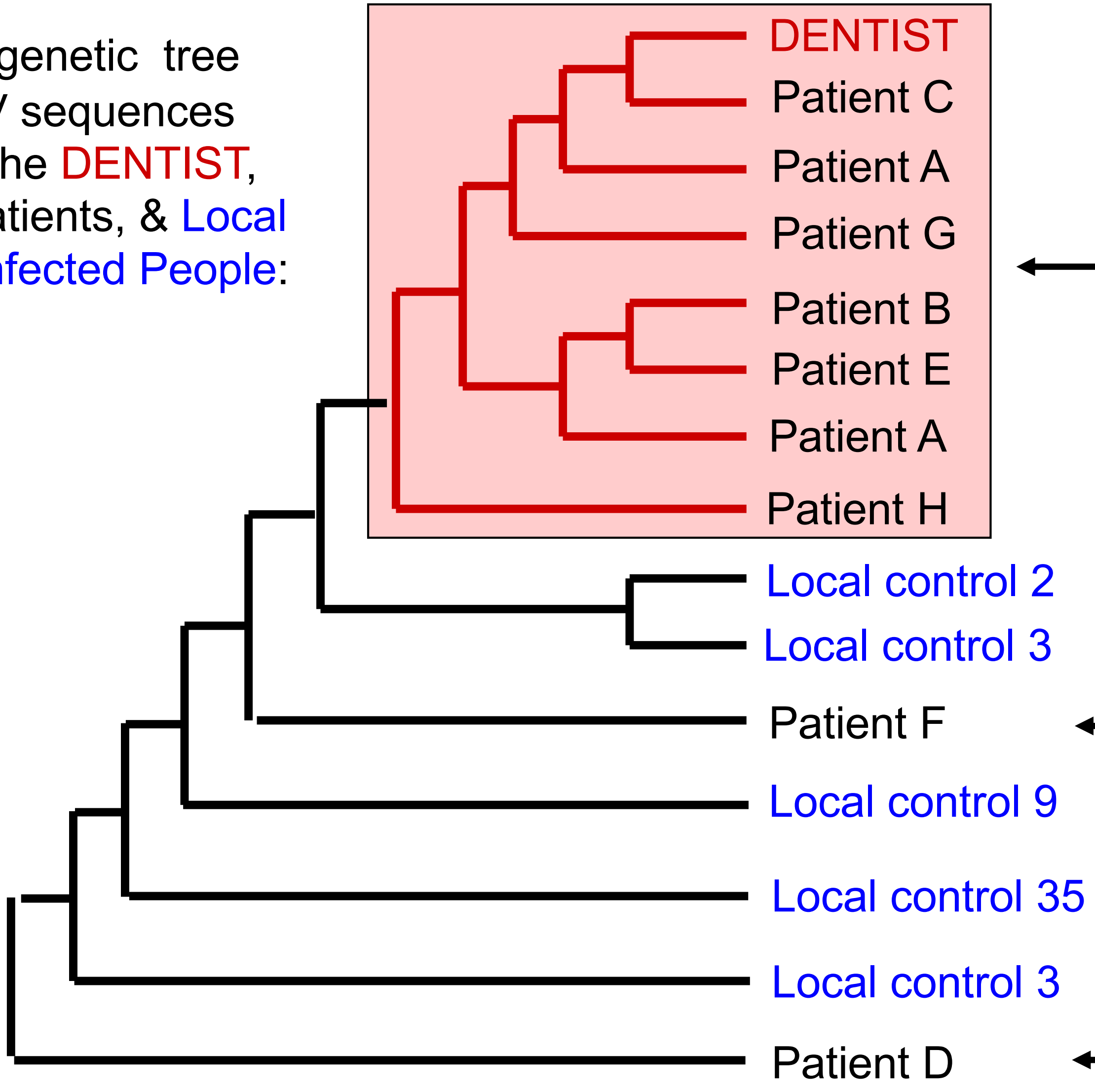
We can reconstruct the likely sequence of protein of an archosaur based on the sequence of the same protein in existant species.

We look at the changes between chicken and alligator



Did the *Florida Dentist* infect his patients with HIV?

Phylogenetic tree of HIV sequences from the DENTIST, his Patients, & Local HIV-infected People:



DENTIST

Patient C

Patient A

Patient G

Patient B

Patient E

Patient A

Patient H

Local control 2

Local control 3

Patient F

Local control 9

Local control 35

Local control 3

Patient D

Yes:
The HIV sequences from these patients fall within the clade of HIV sequences found in the dentist.

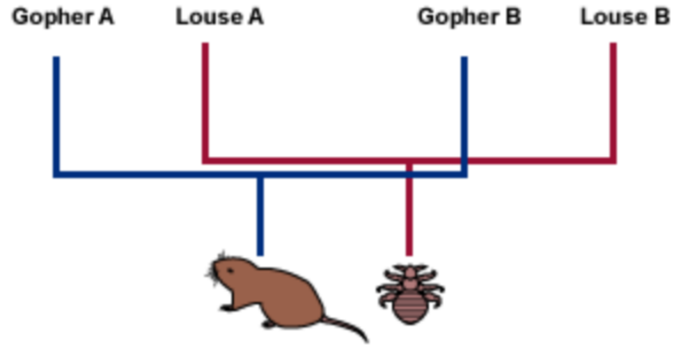
Patient F ← No

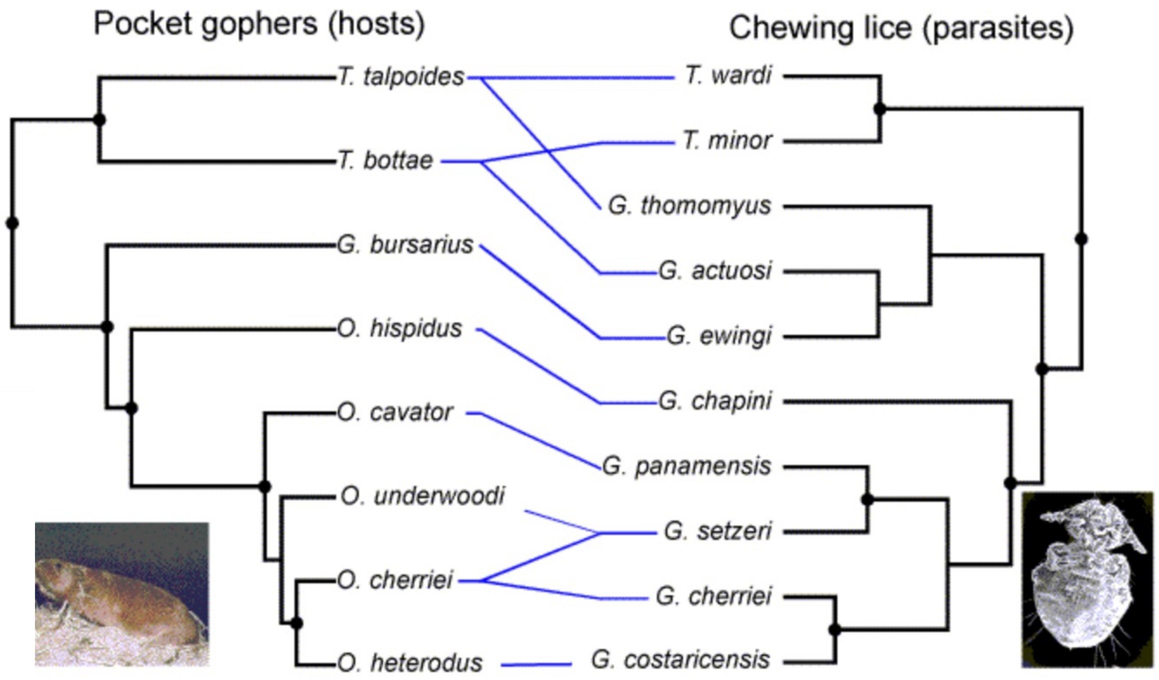
Patient D ← No

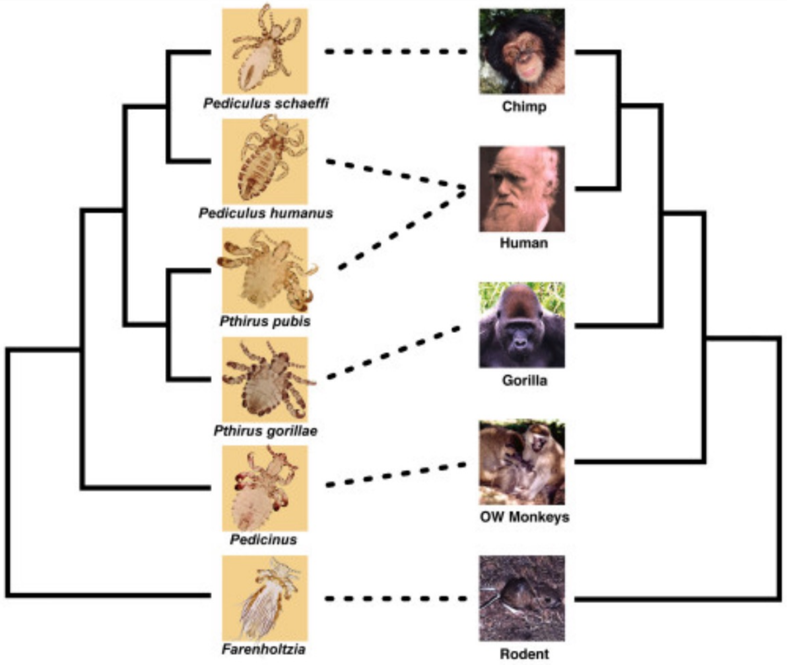
From Ou et *al*. (1992) and Page & Holmes (1998)

Why biologists need algorithms to build trees: evolution

Lice have few opportunities for gopher-switching, and lice on gopher lineage A don't mate with lice living on gopher lineage B. This "geographic" isolation of the louse lineages may cause them to become reproductively isolated as well, and hence, separate species.





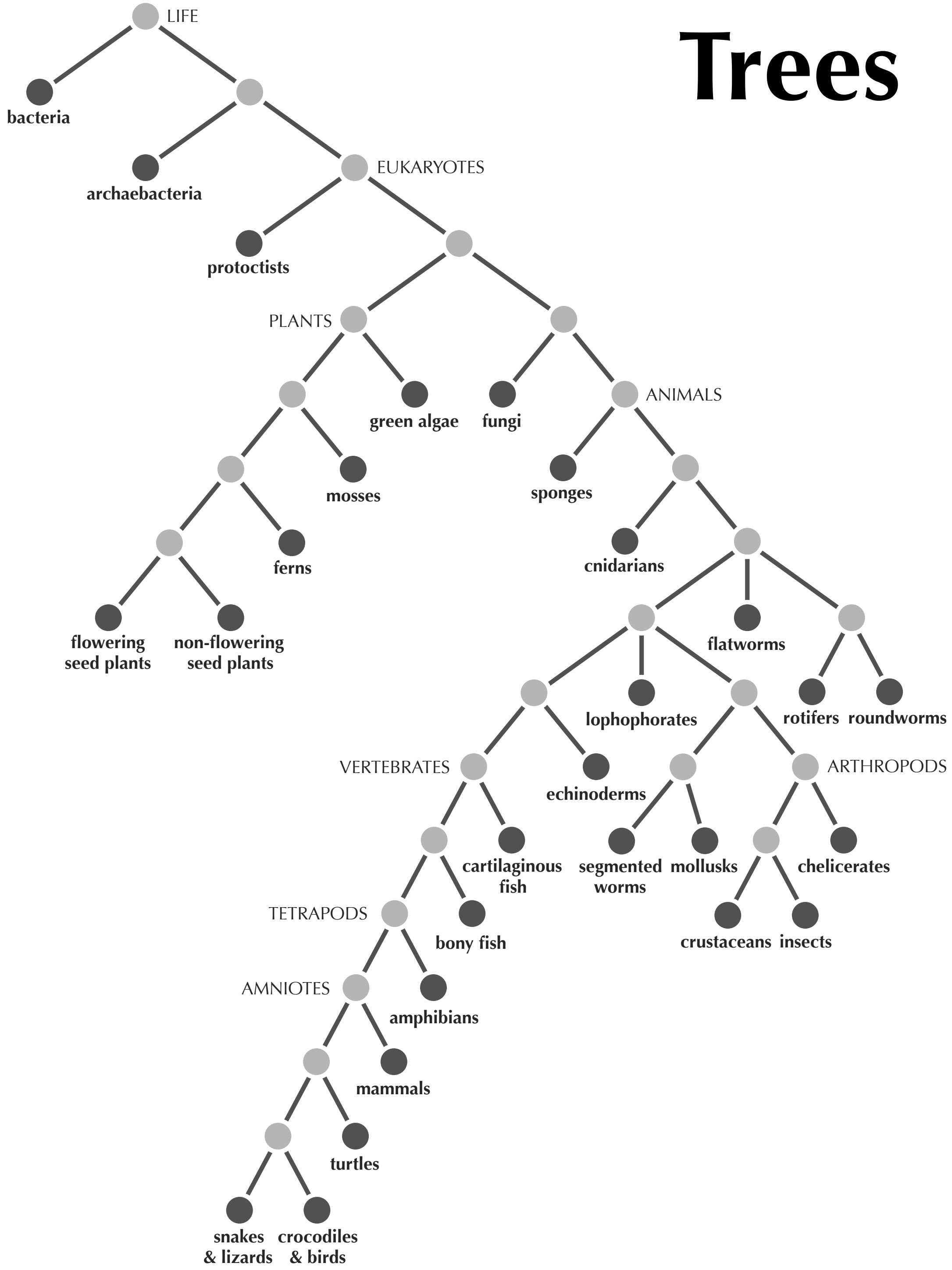


Constructing a Distance Matrix

$D_{i,j}$ = number of differing symbols between *i*-th and *j*-th rows of a "multiple alignment".

SPECIES	ALIGNMENT	DISTANCE MATRIX			
		Chimp	**Human**	**Seal**	**Whale**
Chimp	`ACGTAGGCCT`	0	3	6	4
Human	`ATGTAAGACT`	3	0	7	5
Seal	`TCGAGAGCAC`	6	7	0	2
Whale	`TCGAAAGCAT`	4	5	2	0

Trees



Tree: Connected graph containing no cycles.

Leaves (degree = 1): present-day species

Internal nodes (degree ≥ 1): ancestral species

Constructing a Distance Matrix

$D_{i,j}$ = number of differing symbols between *i*-th and *j*-th rows of a "multiple alignment".

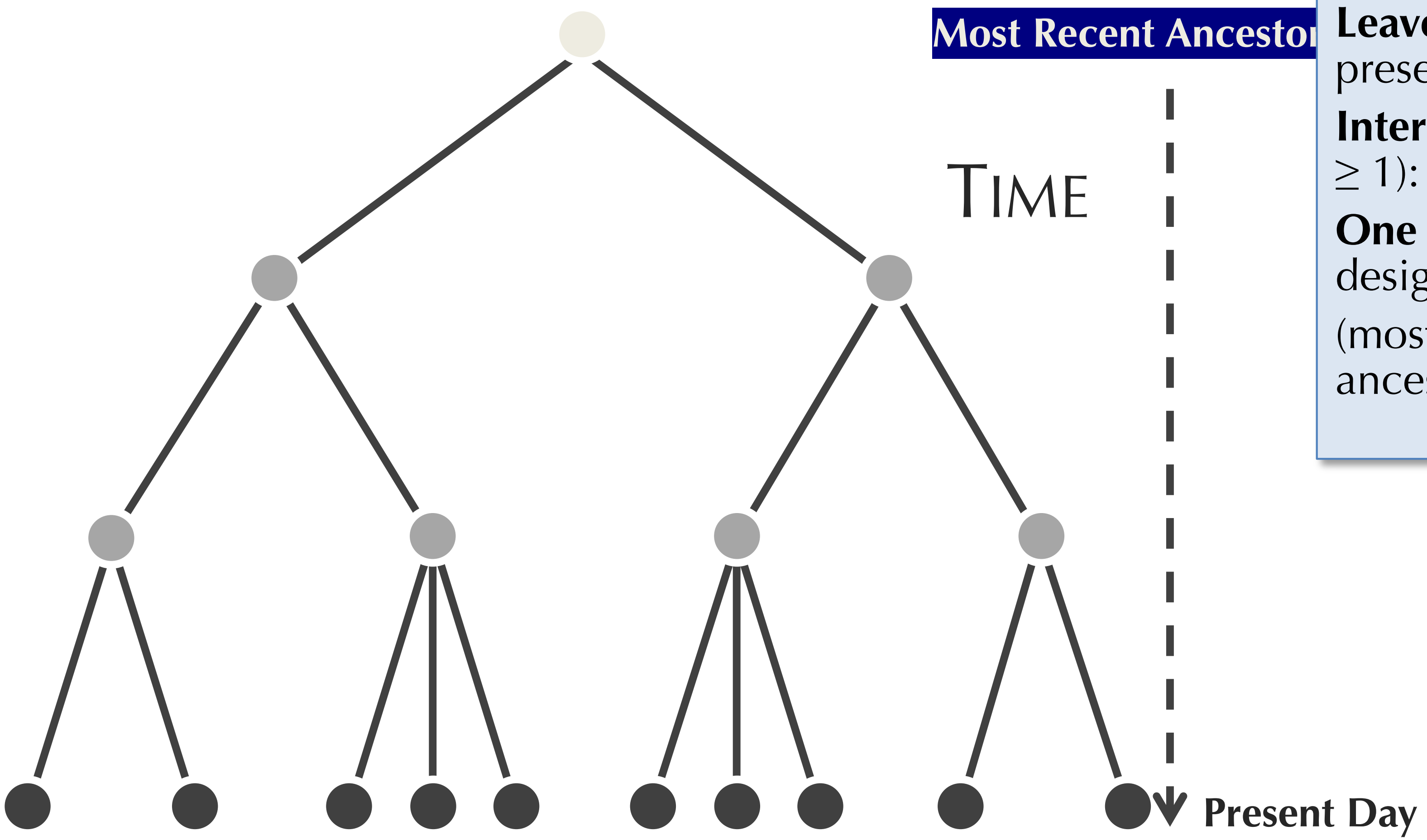
Species	Alignment	Distance Matrix			
		Chimp	**Human**	**Seal**	**Whale**
Chimp	`A`**`C`**`GTA`**`G`**`G`**`C`**`CT`	0	**3**	6	4
Human	`A`**`T`**`GTA`**`A`**`G`**`A`**`CT`	**3**	0	7	5
Seal	`TCGAGAGCAC`	6	7	0	2
Whale	`TCGAAAGCAT`	4	5	2	0

Constructing a Distance Matrix

$D_{i,j}$ = number of differing symbols between *i*-th and *j*-th rows of a multiple alignment.

Species	Alignment	Distance Matrix			
		Chimp	**Human**	**Seal**	**Whale**
Chimp	`ACGTAGGCCT`	0	3	6	4
Human	`ATGTAAGACT`	3	0	7	5
Seal	`TCGAGAGCAC`	6	7	0	2
Whale	`TCGAAAGCAT`	4	5	2	0

How else could we form a distance matrix?



Tree: Connected graph containing no cycles.

Leaves: (degree=1): present day species.

Internal nodes (degree $\geq$ 1): ancestral species.

One node could be designated as root

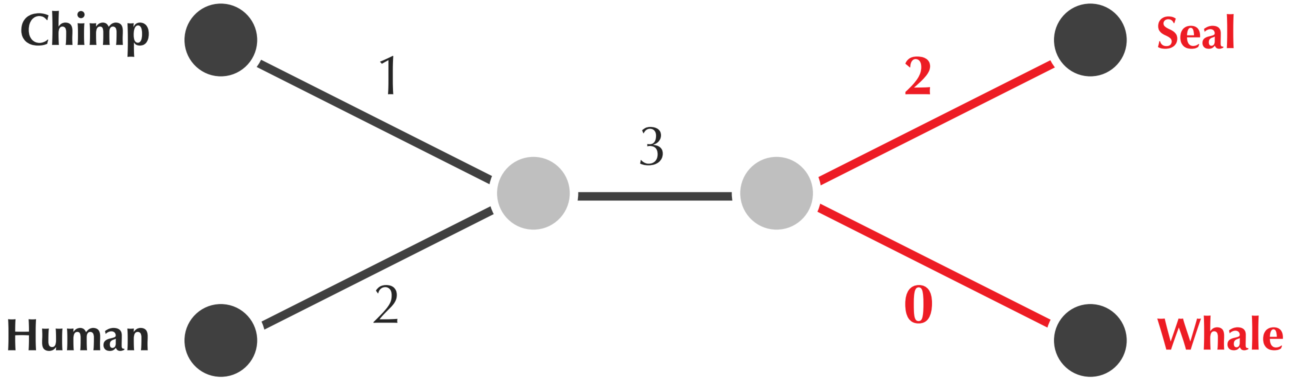
(most recent common ancestor)

Distance-Based Phylogeny Problem: *Construct an evolutionary tree from a distance matrix.*

- **Input:** A distance matrix.
- **Output:** The unrooted tree fitting this distance matrix.

Fitting a Tree to a Matrix

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	**2**
Whale	4	5	2	0



Return to Distance-Based Phylogeny

Distance-Based Phylogeny Problem: *Construct an evolutionary tree from a distance matrix.*

- **Input:** A distance matrix.
- **Output:** The unrooted tree fitting this distance matrix.

Now is this problem well-defined?

Return to Distance-Based Phylogeny

Exercise Break: Try fitting a tree to the following matrix.

	i	***j***	***k***	***l***
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0

No Tree Fits a Matrix

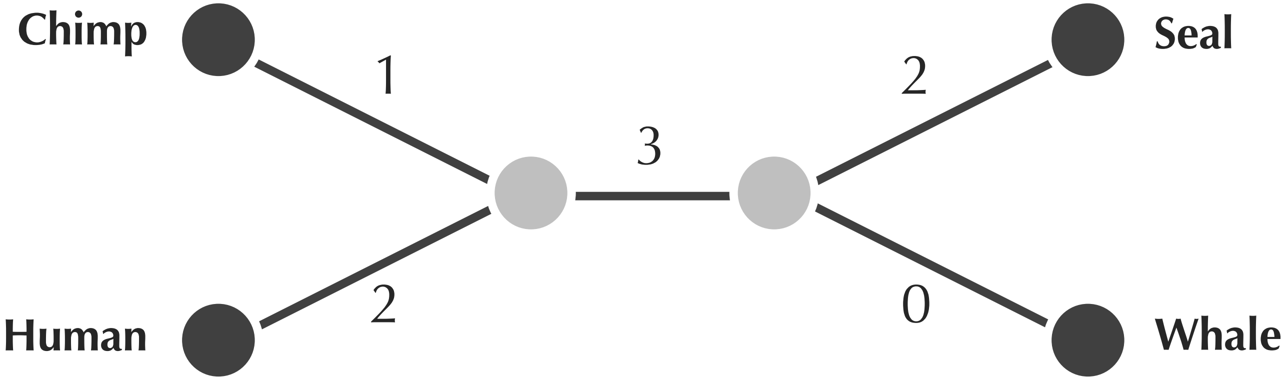
Exercise Break: Try fitting a tree to the following matrix.

$$\begin{array}{ccccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \boldsymbol{i} & 0 & 3 & 4 & 3 \\ \boldsymbol{j} & 3 & 0 & 4 & 5 \\ \boldsymbol{k} & 4 & 4 & 0 & 2 \\ \boldsymbol{l} & 3 & 5 & 2 & 0 \end{array}$$

Additive matrix: distance matrix such that there exists an unrooted tree fitting it.

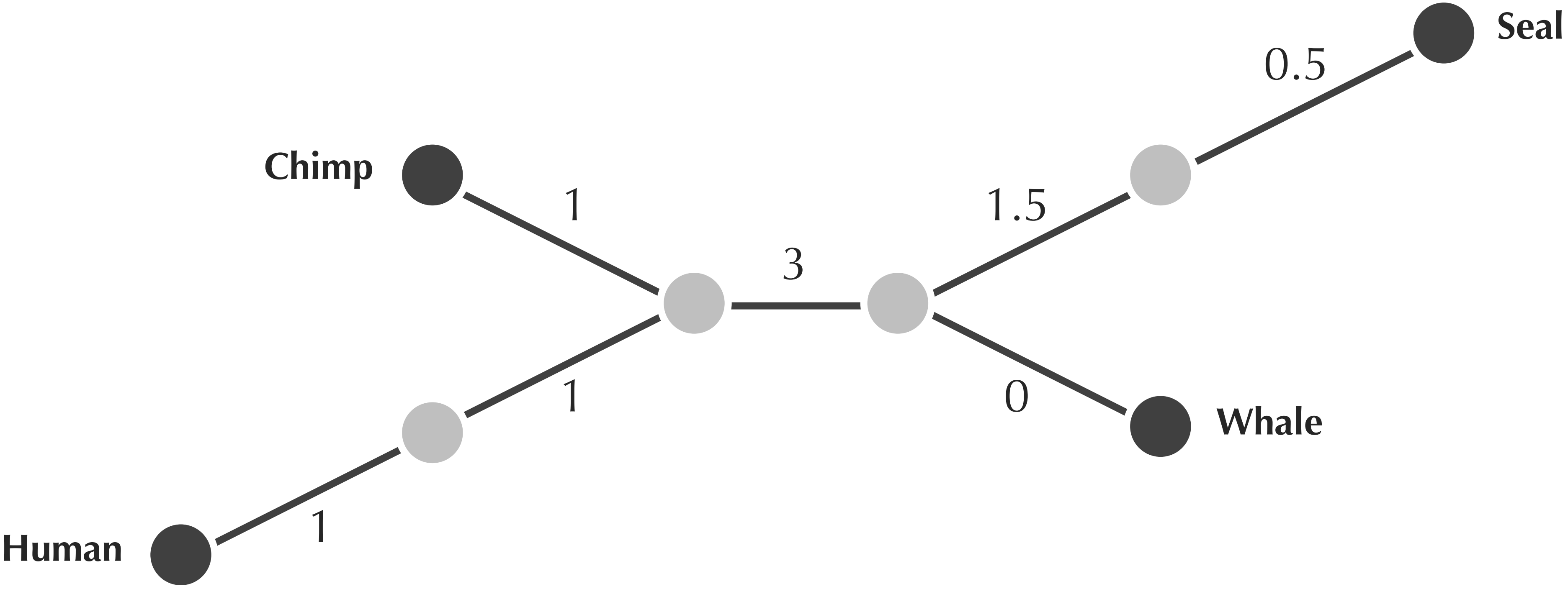
More Than One Tree Fits a Matrix

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

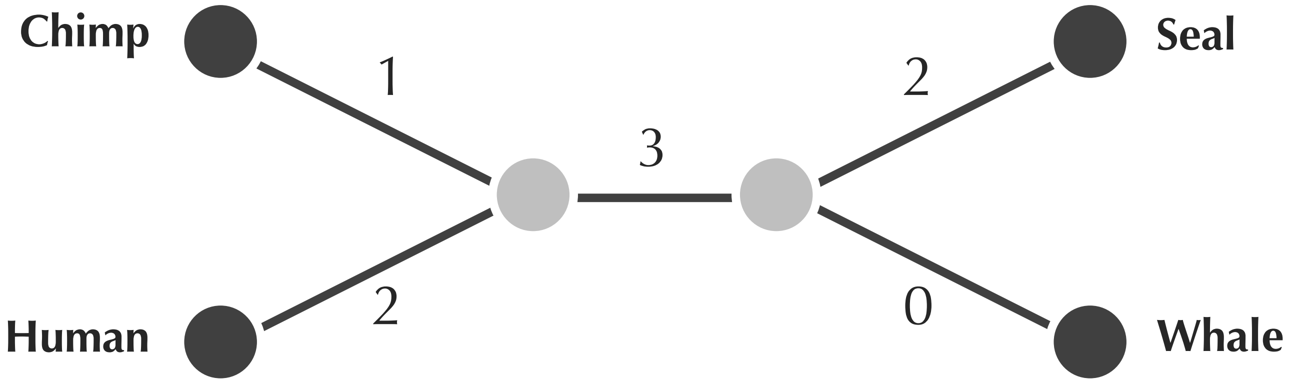


More Than One Tree Fits a Matrix

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



Which Tree is "Better"?



Simple tree: tree with no nodes of degree 2.

Theorem: There is a unique *simple* tree fitting an *additive* matrix.

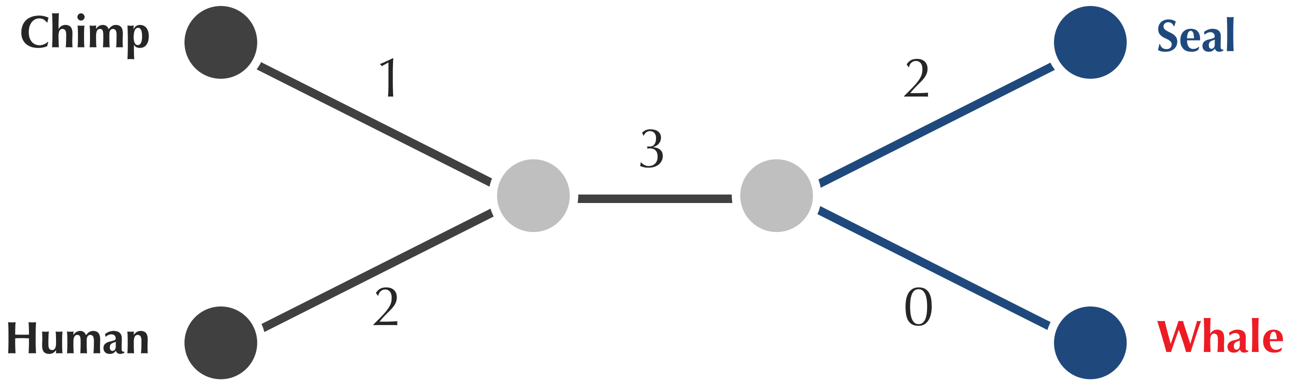
Reformulating Distance-Based Phylogeny

Distance-Based Phylogeny Problem: *Construct an evolutionary tree from a distance matrix.*

- **Input:** A distance matrix.
- **Output:** The simple tree fitting this distance matrix (if this matrix is additive).

An Idea for Distance-Based Phylogeny

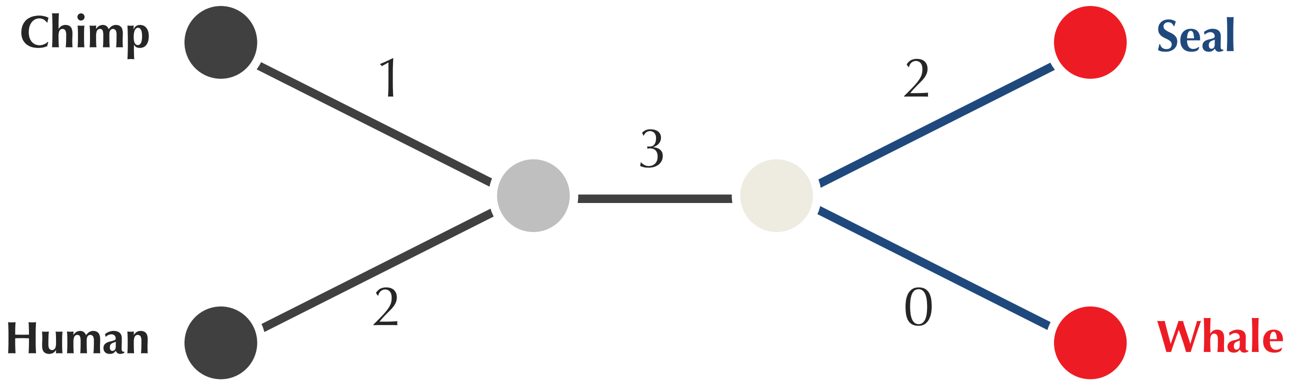
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	**2**
Whale	4	5	2	0



An Idea for Distance-Based Phylogeny

Seal and whale are **neighbors** (meaning they share the same **parent**).

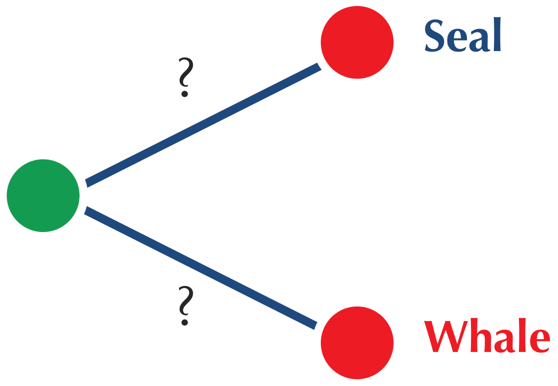
Theorem: Every simple tree with at least two nodes has at least one pair of neighboring leaves.



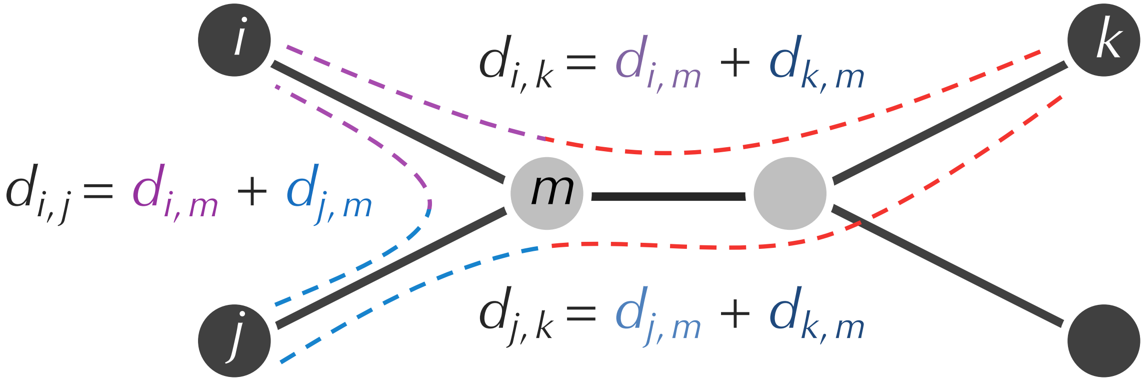
An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	**2**
Whale	4	5	2	0

How do we compute the unknown distances?



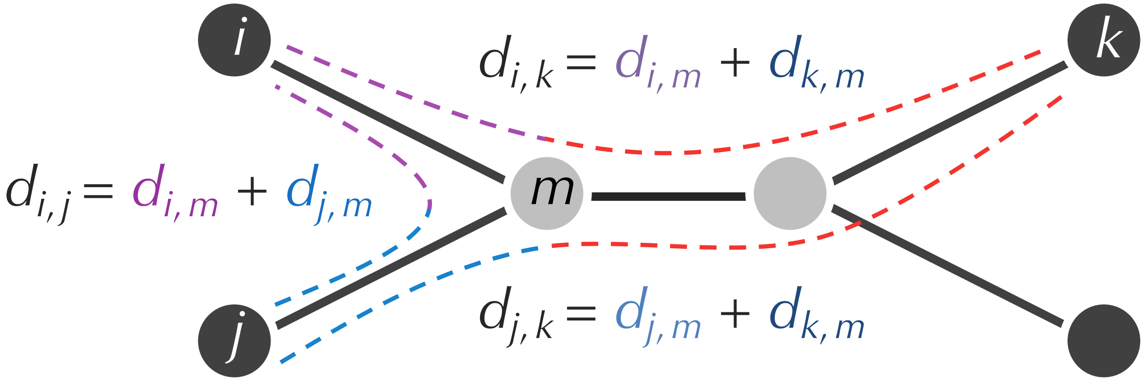
Toward a Recursive Algorithm



i
k
$d_{i,k} = d_{i,m} + d_{k,m}$
$d_{i,j} = d_{i,m} + d_{j,m}$
m
j
$d_{j,k} = d_{j,m} + d_{k,m}$

$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] \;/\; 2$$

Toward a Recursive Algorithm



i
k
m
j
$d_{i,k} = d_{i,m} + d_{k,m}$
$d_{i,j} = d_{i,m} + d_{j,m}$
$d_{j,k} = d_{j,m} + d_{k,m}$

$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

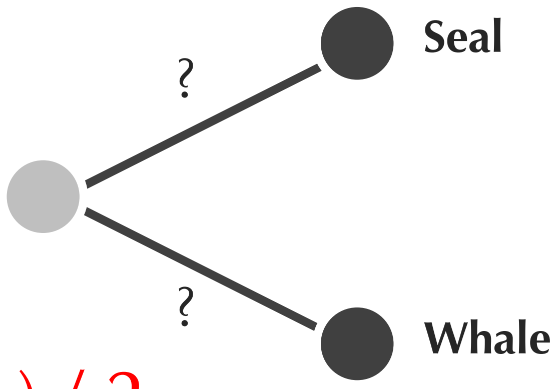
$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$\therefore \quad d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$$

An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$$

An Idea for Distance-Based Phylogeny

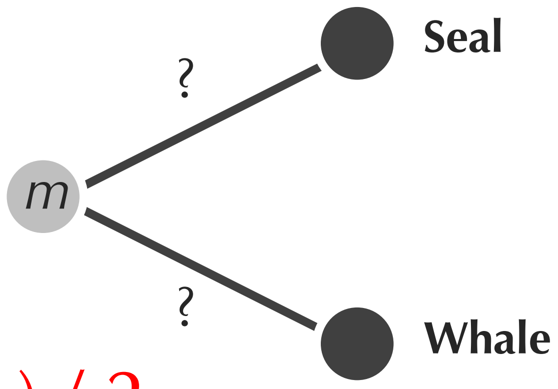
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) \,/\, 2$$

$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) \,/\, 2$$

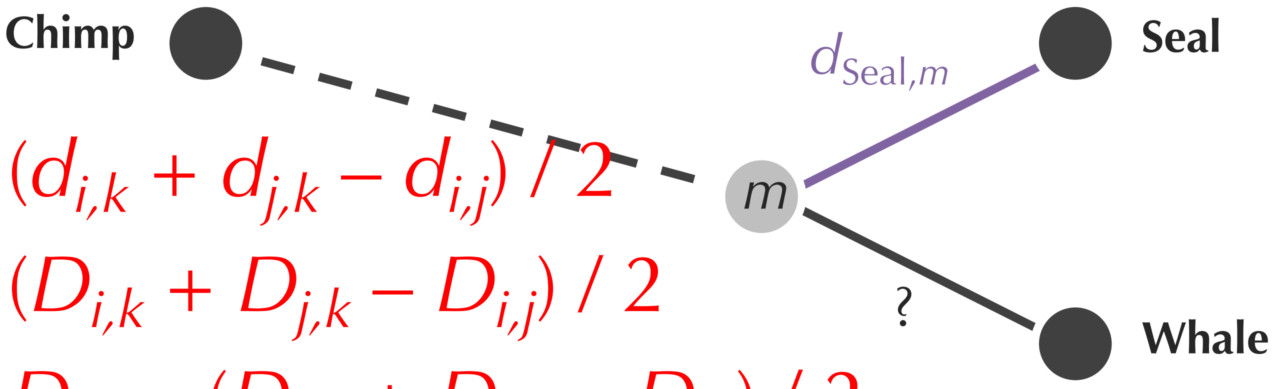
$$d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) \,/\, 2$$

$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) \,/\, 2$$



An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) \,/\, 2$$

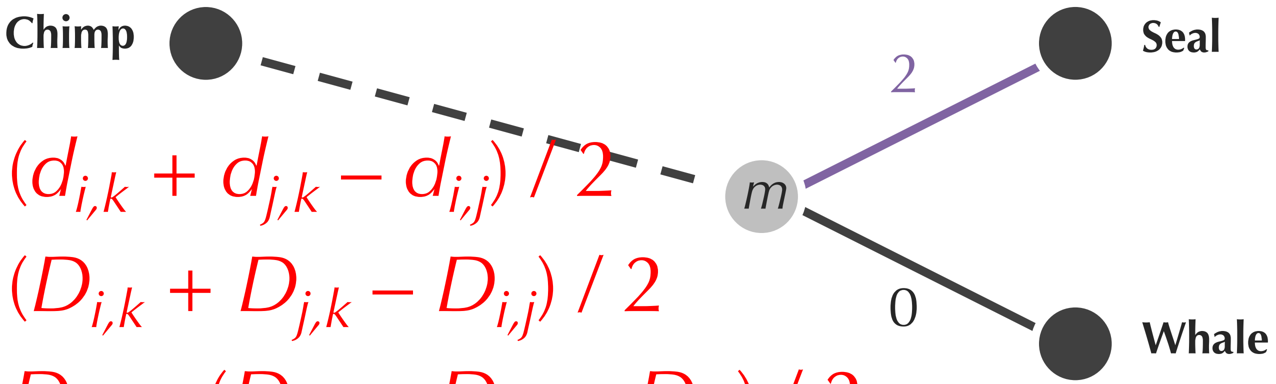
$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) \,/\, 2$$

$$d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) \,/\, 2$$

$$d_{\text{Seal},m} = (D_{\text{Seal,Chimp}} + D_{\text{Seal,Whale}} - D_{\text{Whale,Chimp}}) \,/\, 2$$

An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

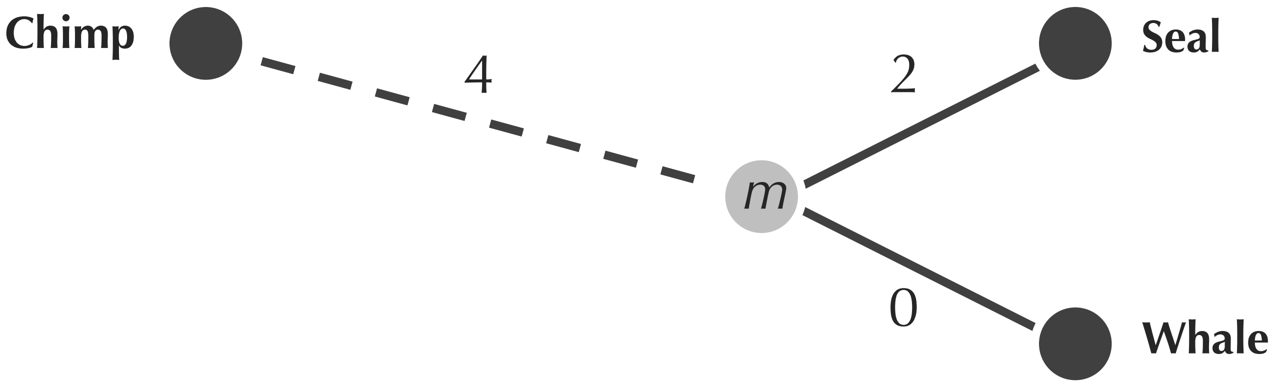
$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{\text{Seal},m} = 2$$

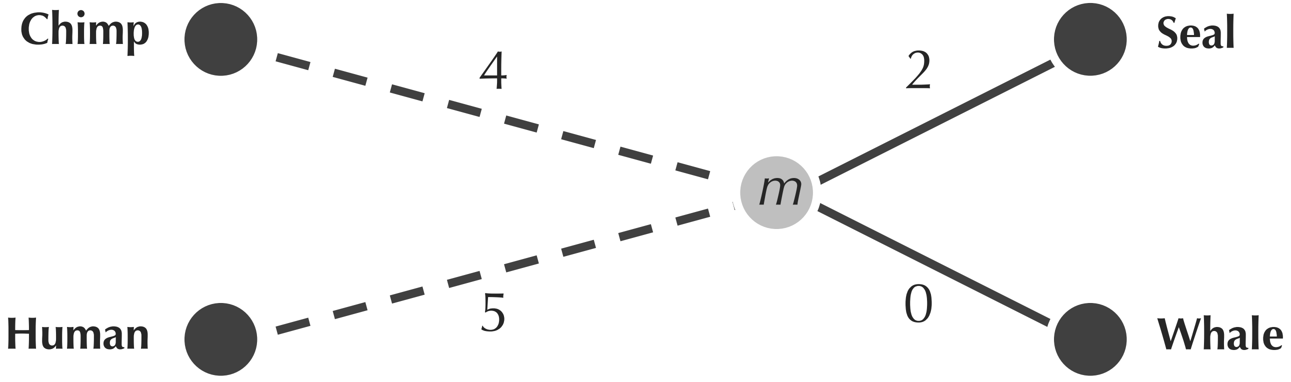
An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



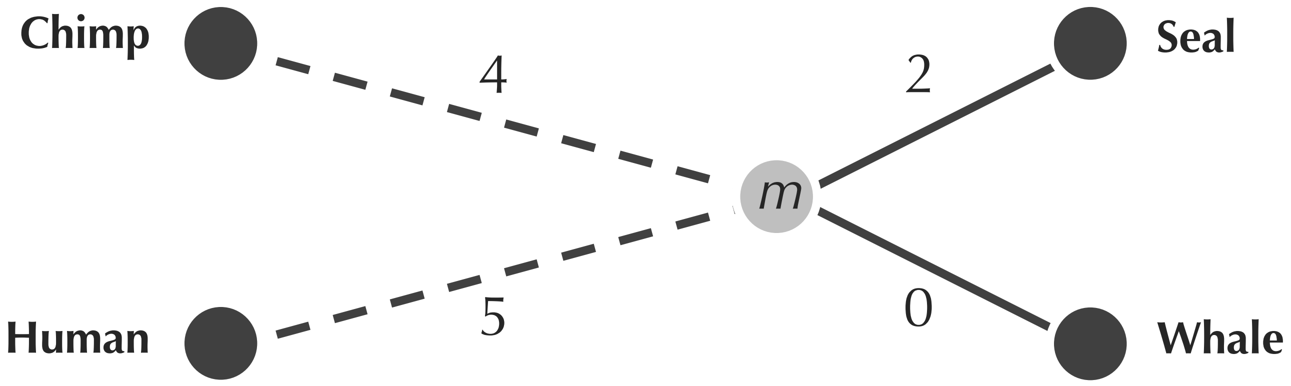
An Idea for Distance-Based Phylogeny

	Chimp	**Human**	**Seal**	**Whale**	***m***
Chimp	0	3	6	4	4
Human	3	0	7	5	5
Seal	6	7	0	2	2
Whale	4	5	2	0	0
m	4	5	2	0	0



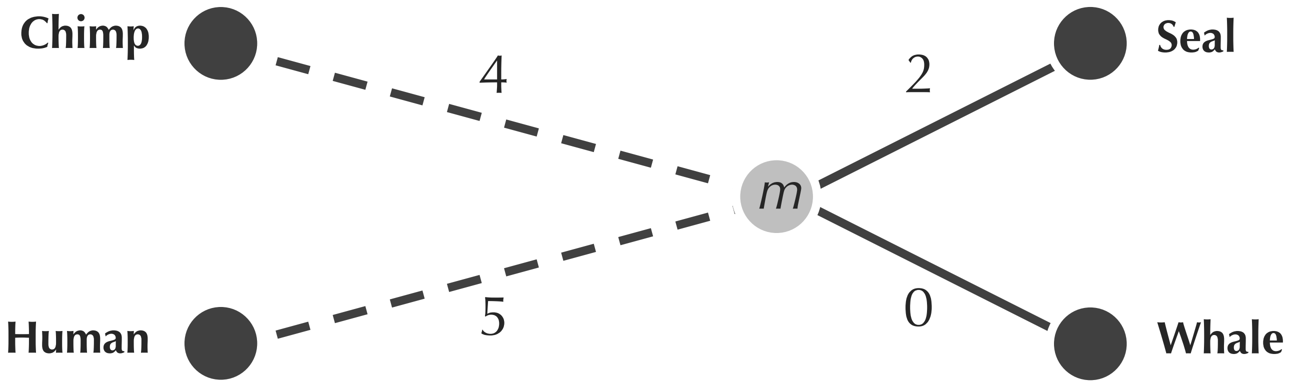
An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale	*m*
Chimp	0	3	6	4	4
Human	3	0	7	5	5
Seal	6	7	0	2	2
Whale	4	5	2	0	0
m	4	5	2	0	0



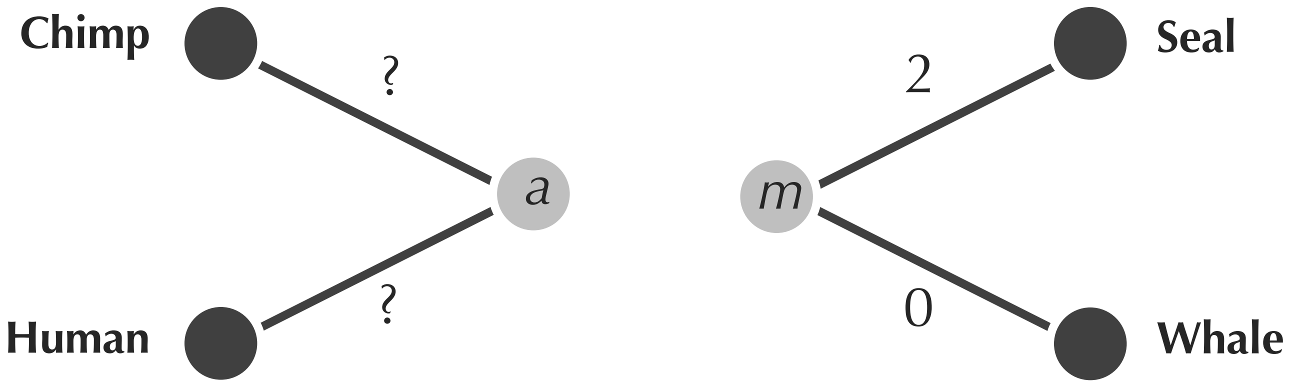
An Idea for Distance-Based Phylogeny

	Chimp	**Human**	***m***
Chimp	0	3	4
Human	3	0	5
m	4	5	0



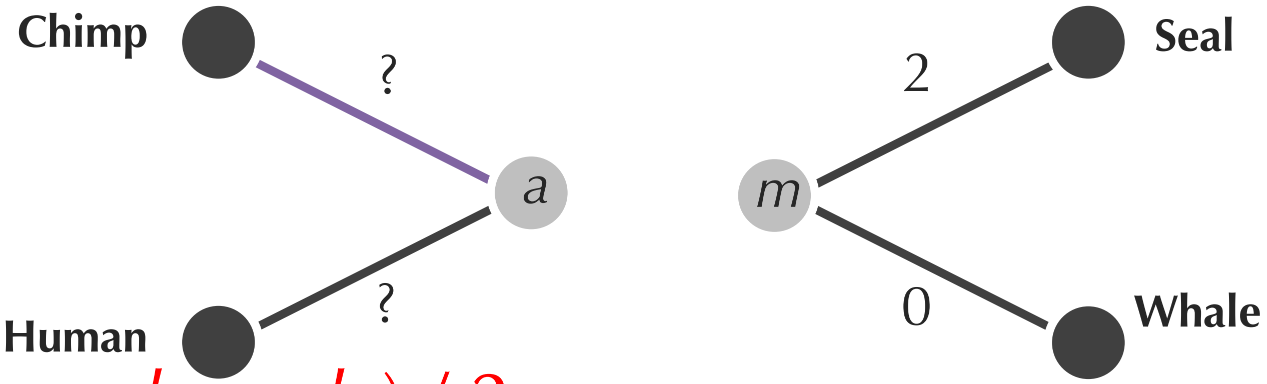
An Idea for Distance-Based Phylogeny

	Chimp	Human	*m*
Chimp	0	3	4
Human	3	0	5
m	4	5	0



An Idea for Distance-Based Phylogeny

	Chimp	Human	*m*
Chimp	0	3	4
Human	3	0	5
m	4	5	0



$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

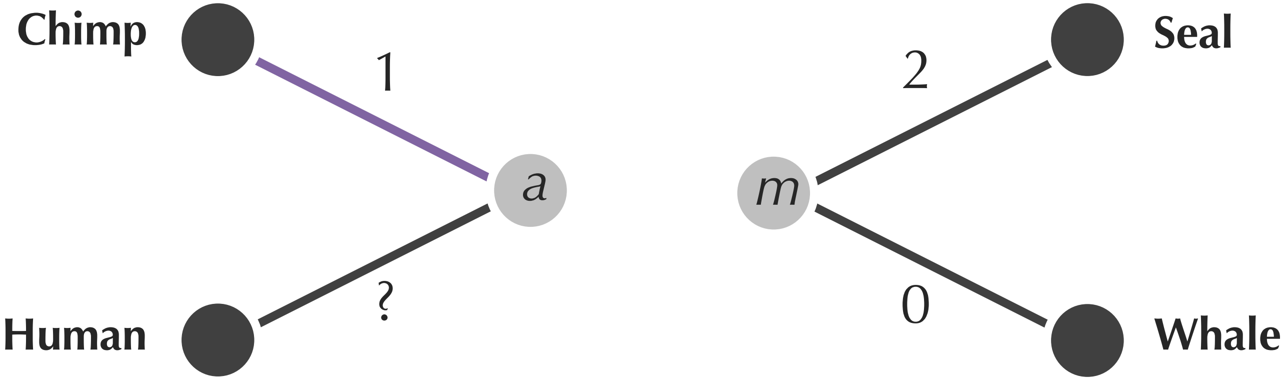
$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{\mathrm{Chimp},a} = (D_{\mathrm{Chimp},m} + D_{\mathrm{Chimp,Human}} - D_{\mathrm{Human},m}) / 2$$

An Idea for Distance-Based Phylogeny

	Chimp	**Human**	***m***
Chimp	0	3	4
Human	3	0	5
m	4	5	0



$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) \,/\, 2$$

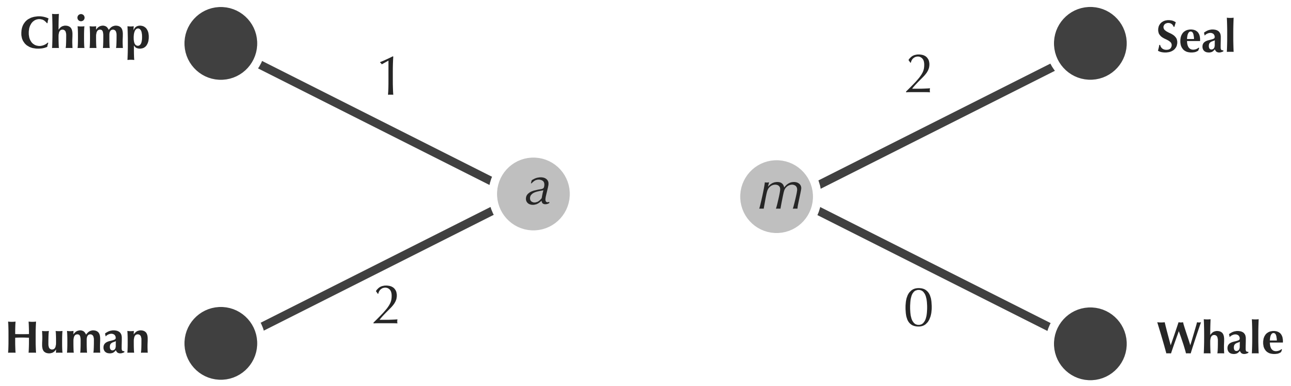
$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) \,/\, 2$$

$$d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) \,/\, 2$$

$$d_{\text{Chimp},a} = 1$$

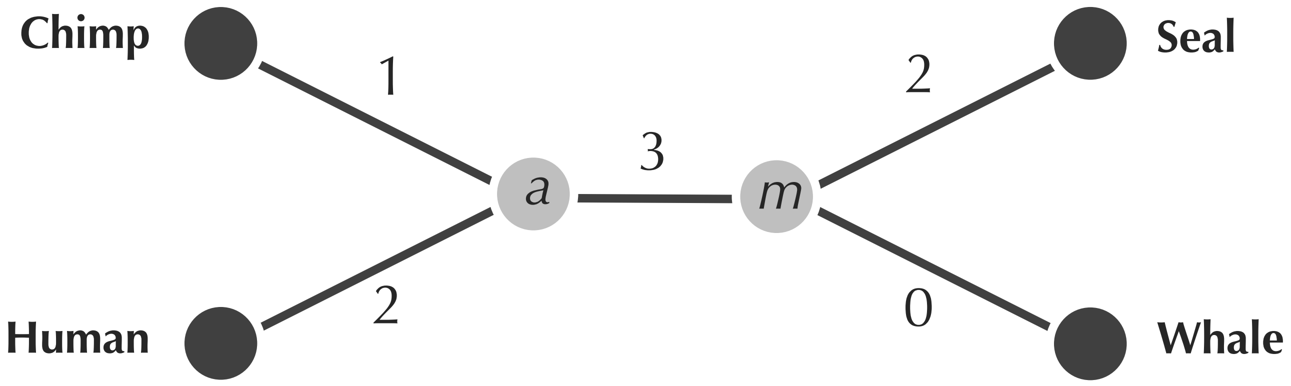
An Idea for Distance-Based Phylogeny

	Chimp	**Human**	***m***
Chimp	0	3	4
Human	3	0	5
m	4	5	0



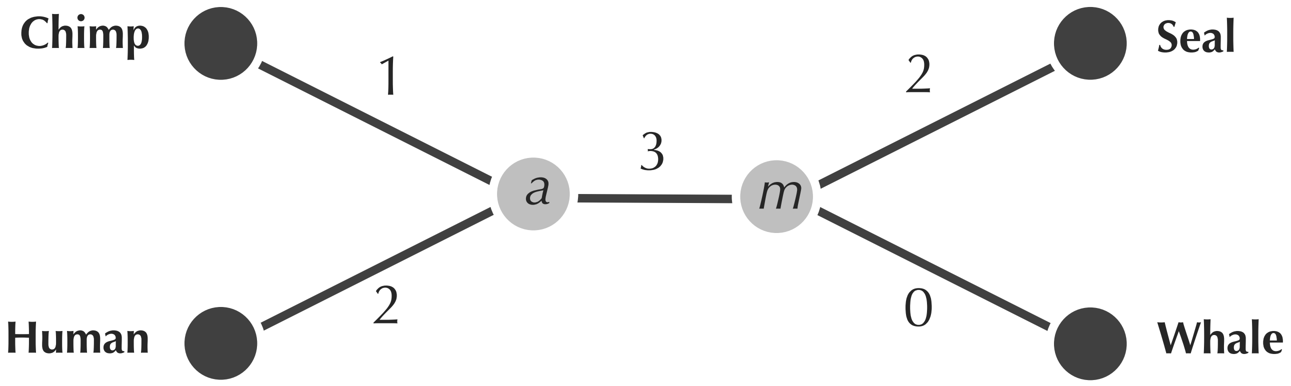
An Idea for Distance-Based Phylogeny

	Chimp	**Human**	***m***
Chimp	0	3	4
Human	3	0	5
m	4	5	0



An Idea for Distance-Based Phylogeny

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



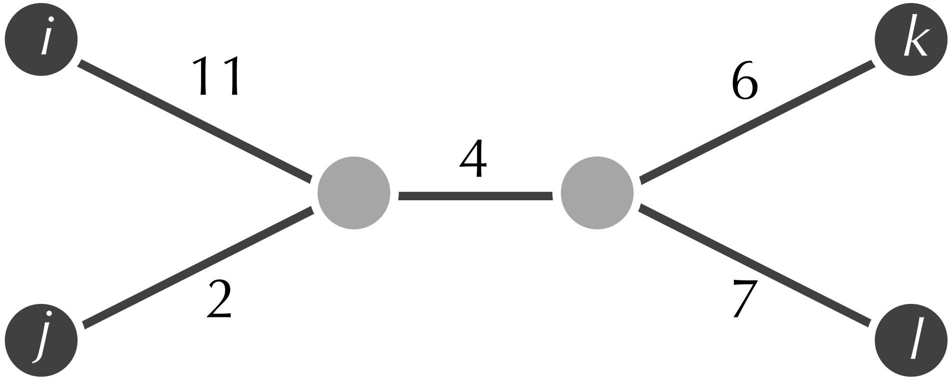
An Idea for Distance-Based Phylogeny

Exercise Break: Apply this recursive approach to the distance matrix below.

	i	***j***	***k***	***l***
i	0	13	21	22
j	13	0	12	13
k	21	12	0	13
l	22	13	13	0

What Was Wrong With Our Algorithm?

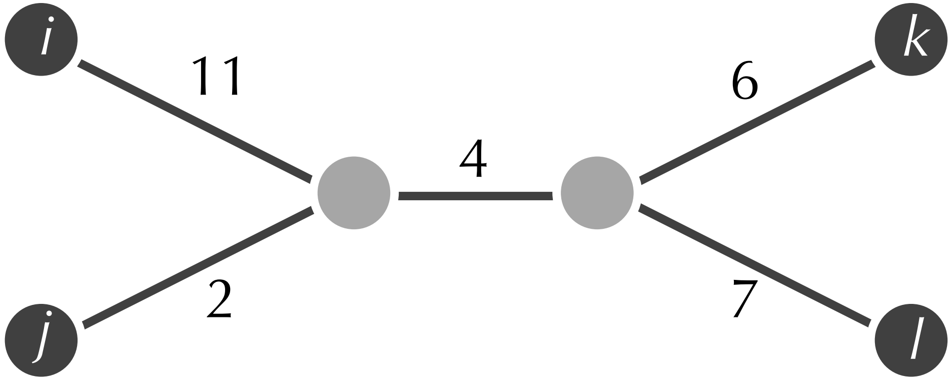
	i	***j***	***k***	***l***
i	0	13	21	22
j	13	0	12	13
k	21	12	0	13
l	22	13	13	0



What Was Wrong With Our Algorithm?

	i	***j***	***k***	***l***
i	0	13	21	22
j	13	0	**12**	13
k	21	12	0	13
l	22	13	13	0

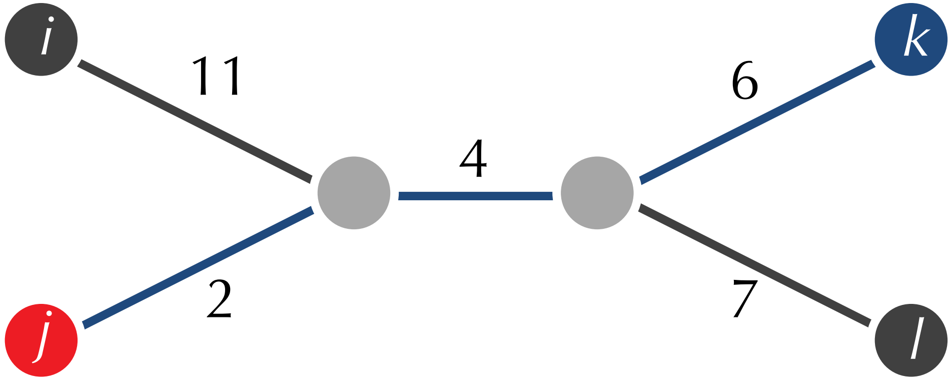
minimum element is $D_{j,k}$



What Was Wrong With Our Algorithm?

$$\begin{array}{c|cccc}
 & i & j & k & l \\
i & 0 & 13 & 21 & 22 \\
j & 13 & 0 & \mathbf{12} & 13 \\
k & 21 & 12 & 0 & 13 \\
l & 22 & 13 & 13 & 0
\end{array}$$

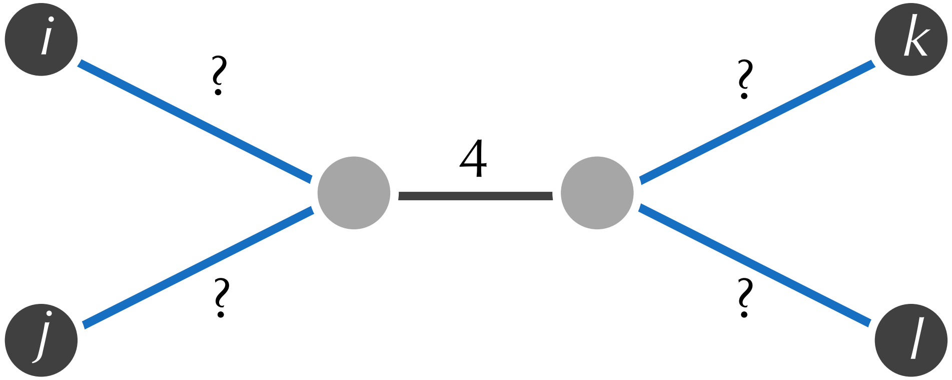
minimum element is $D_{j,k}$



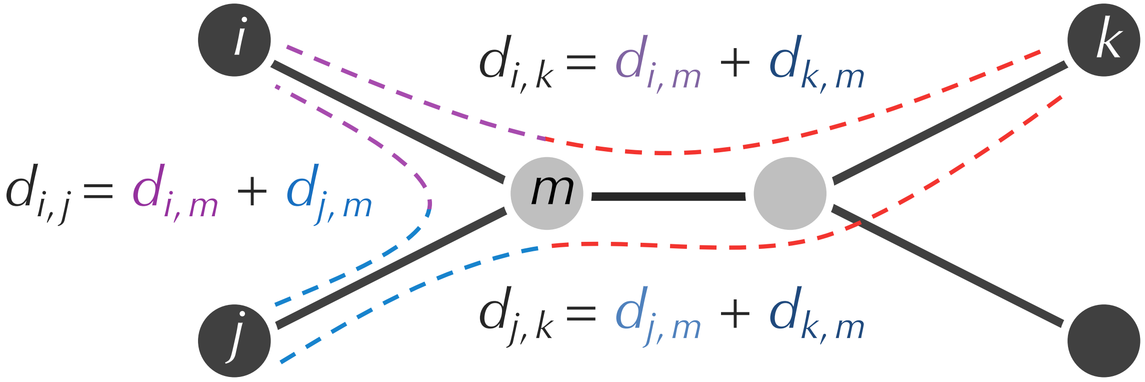
j and *k* are **not** neighbors!

From Neighbors to Limbs

Rather than trying to find **neighbors**, let's instead try to compute the length of **limbs**, the edges attached to leaves.



From Neighbors to Limbs



i
k
m
j
$d_{i,k} = d_{i,m} + d_{k,m}$
$d_{i,j} = d_{i,m} + d_{j,m}$
$d_{j,k} = d_{j,m} + d_{k,m}$

$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

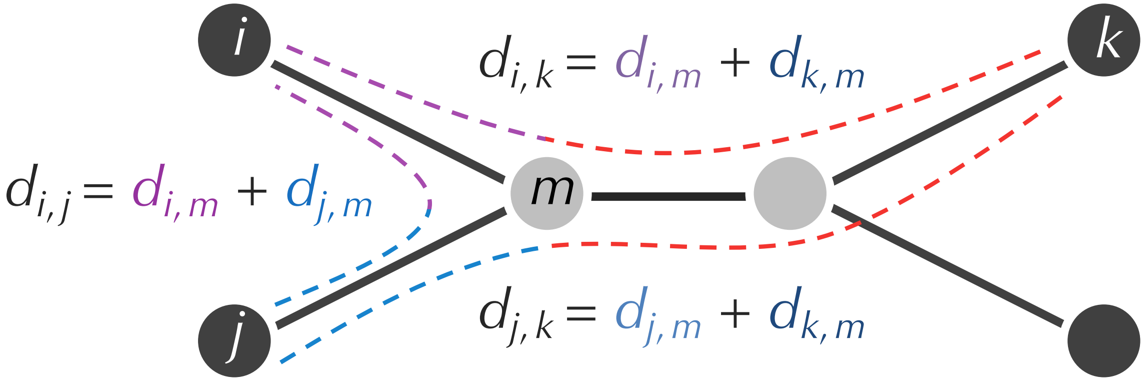
$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$\therefore \quad d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$$

From Neighbors to Limbs



i
k
m
j
$d_{i,k} = d_{i,m} + d_{k,m}$
$d_{i,j} = d_{i,m} + d_{j,m}$
$d_{j,k} = d_{j,m} + d_{k,m}$

$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$\therefore \quad d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

$$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$$

Assumes that *i* and *j* are *neighbors*...

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(*i*) is equal to the minimum value of $(D_{i,k} + D_{i,j} - D_{j,k})/2$ over all leaves *j* and *k*.

Limb Length Problem: *Compute the length of a limb in the simple tree fitting an additive distance matrix.*

- **Input:** An additive distance matrix *D* and an integer *j*.
- **Output:** The length of the limb connecting leaf *j* to its parent, *LimbLength*(*j*).

Code Challenge: Solve the Limb Length Problem.

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{\text{j},k})/2$ over all leaves *j* and *k*.

	Chimp	**Human**	**Seal**	**Whale**
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

$$(D_{\text{chimp, human}} + D_{\text{chimp, seal}} - D_{\text{human, seal}})\ /\ 2 \qquad = (3 + 6 - 7)\ /\ 2 = 1$$

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{\text{j},k})/2$ over all leaves *j* and *k*.

	Chimp	**Human**	**Seal**	**Whale**
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

$$(D_{\text{chimp, human}} + D_{\text{chimp, seal}} - D_{\text{human, seal}}) / 2 = (3 + 6 - 7) / 2 = 1$$

$$(D_{\text{chimp, human}} + D_{\text{chimp, whale}} - D_{\text{human, whale}}) / 2 = (3 + 4 - 5) / 2 = 1$$

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{j,k})/2$ over all leaves *j* and *k*.

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

$$(D_{\text{chimp, human}} + D_{\text{chimp, seal}} - D_{\text{human, seal}}) / 2 = (3 + 6 - 7) / 2 = 1$$

$$(D_{\text{chimp, human}} + D_{\text{chimp, whale}} - D_{\text{human, whale}}) / 2 = (3 + 4 - 5) / 2 = 1$$

$$(D_{\text{chimp, whale}} + D_{\text{chimp, seal}} - D_{\text{whale, seal}}) / 2 = (6 + 4 - 2) / 2 = 4$$

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{\text{j},k})/2$ over all leaves *j* and *k*.

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0

$$(D_{\text{human, chimp}} + D_{\text{chimp, seal}} - D_{\text{human, seal}})\ /\ 2 \quad = (3 + 6 - 7)\ /\ 2 = \mathbf{1}$$

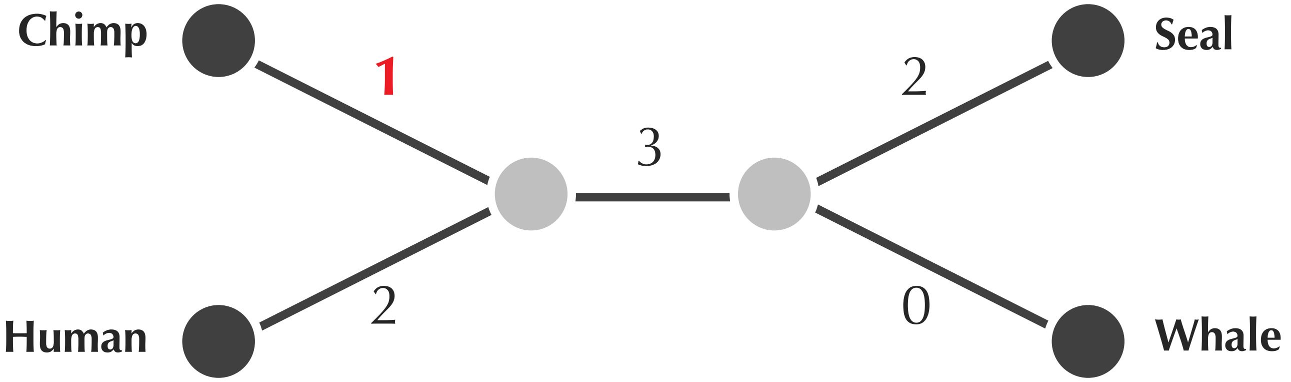
$$(D_{\text{human, chimp}} + D_{\text{chimp, whale}} - D_{\text{human, whale}})\ /\ 2 \quad = (3 + 4 - 5)\ /\ 2 = \mathbf{1}$$

$$(D_{\text{whale, chimp}} + D_{\text{chimp, seal}} - D_{\text{whale, seal}})\ /\ 2 \quad = (6 + 4 - 2)\ /\ 2 = 4$$

Computing Limb Lengths

Limb Length Theorem: *LimbLength*(chimp) is equal to the minimum value of $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{\text{j},k})/2$ over all leaves *j* and *k*.

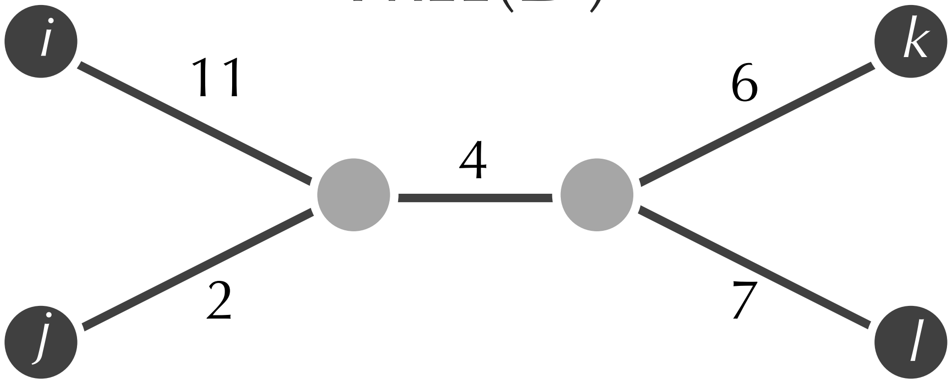
	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



AdditivePhylogeny In Action

$$D \quad \begin{array}{ccccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \boldsymbol{i} & 0 & 13 & 21 & 22 \\ \boldsymbol{j} & 13 & 0 & 12 & 13 \\ \boldsymbol{k} & 21 & 12 & 0 & 13 \\ \boldsymbol{l} & 22 & 13 & 13 & 0 \end{array}$$

TREE(D)



AdditivePhylogeny In Action

$$D \quad \begin{array}{ccccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \boldsymbol{i} & 0 & 13 & 21 & 22 \\ \boldsymbol{j} & 13 & 0 & 12 & 13 \\ \boldsymbol{k} & 21 & 12 & 0 & 13 \\ \boldsymbol{l} & 22 & 13 & 13 & 0 \end{array}$$

1. Pick an arbitrary leaf *j*.

AdditivePhylogeny In Action

$$D \quad \begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \boldsymbol{i} & 0 & 13 & 21 & 22 \\ \boldsymbol{j} & 13 & 0 & 12 & 13 \\ \boldsymbol{k} & 21 & 12 & 0 & 13 \\ \boldsymbol{l} & 22 & 13 & 13 & 0 \end{array}$$

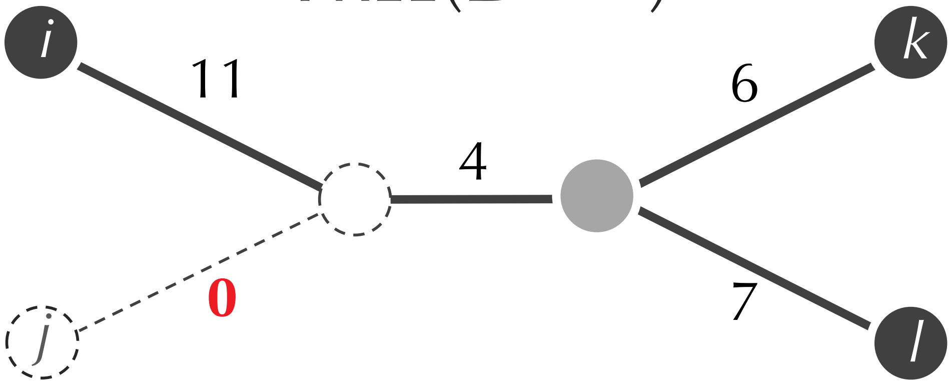
LimbLength(*j*) = 2

2. Compute its limb length, *LimbLength*(*j*).

AdditivePhylogeny In Action

$$D^{\text{bald}} \quad \begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{i} & 0 & 11 & 21 & 22 \\ \boldsymbol{j} & 11 & 0 & 10 & 11 \\ \boldsymbol{k} & 21 & 10 & 0 & 13 \\ \boldsymbol{l} & 22 & 11 & 13 & 0 \end{array}$$

TREE(D^{bald})



3. Subtract *LimbLength*(*j*) from each row and column to produce D^{bald} in which *j* is a **bald limb** (length 0).

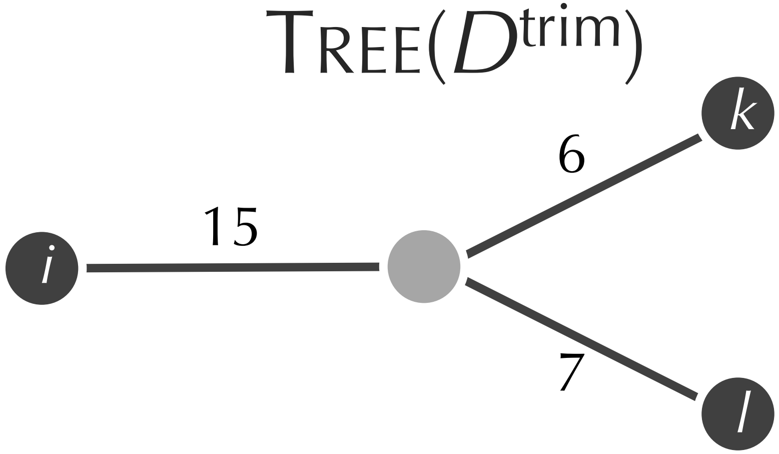
AdditivePhylogeny In Action

$$D^{\text{trim}} \quad \begin{array}{c|cccc} & \boldsymbol{i} & j & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{i} & 0 & 11 & 21 & 22 \\ j & 11 & 0 & 10 & 11 \\ \boldsymbol{k} & 21 & 10 & 0 & 13 \\ \boldsymbol{l} & 22 & 11 & 13 & 0 \end{array}$$

4. Remove the *j*-th row and column of the matrix to form the (*n* – 1) x (*n* – 1) matrix D^{trim}.

AdditivePhylogeny In Action

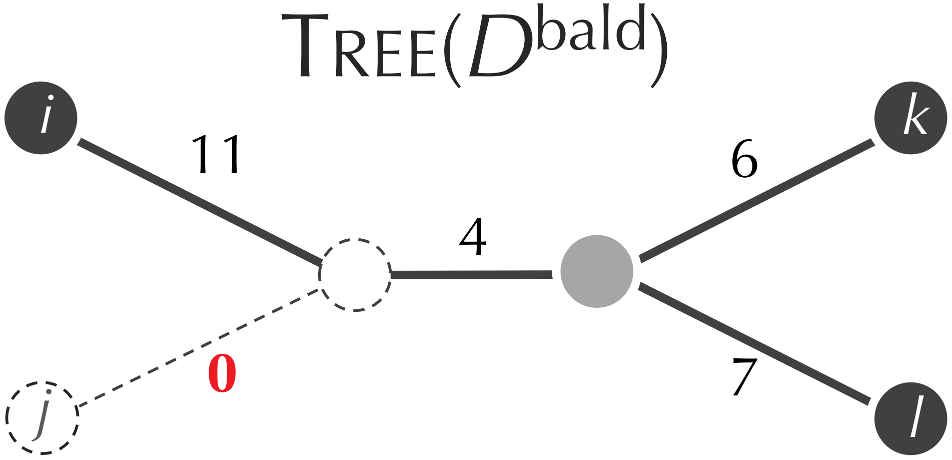
$$D^{\text{trim}} \quad \begin{array}{ccccc} & i & j & k & l \\ i & 0 & 11 & 21 & 22 \\ j & 11 & 0 & 10 & 11 \\ k & 21 & 10 & 0 & 13 \\ l & 22 & 11 & 13 & 0 \end{array}$$



5. Construct *Tree*(D^{trim}).

AdditivePhylogeny In Action

$$D^{\text{bald}} \quad \begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{i} & 0 & 11 & 21 & 22 \\ \boldsymbol{j} & 11 & 0 & 10 & 11 \\ \boldsymbol{k} & 21 & 10 & 0 & 13 \\ \boldsymbol{l} & 22 & 11 & 13 & 0 \end{array}$$

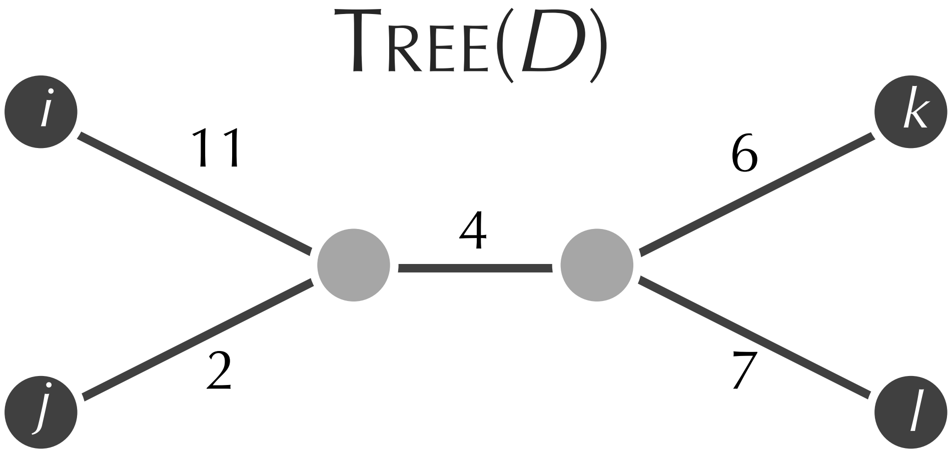


6. Identify the point in *Tree*(D^{trim}) where leaf *j* should be attached.

AdditivePhylogeny In Action

$$D \quad \begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{i} & 0 & 13 & 21 & 22 \\ \boldsymbol{j} & 13 & 0 & 12 & 13 \\ \boldsymbol{k} & 21 & 12 & 0 & 13 \\ \boldsymbol{l} & 22 & 13 & 13 & 0 \end{array}$$

LimbLength(*j*) = 2



7. Attach *j* by an edge of length *LimbLength*(*j*) in order to form *Tree*(*D*).

AdditivePhylogeny

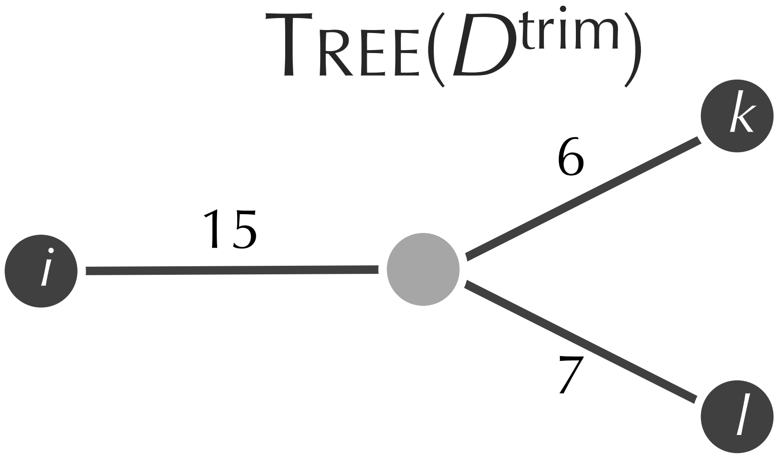
```
AdditivePhylogeny($D$):
1. Pick an arbitrary leaf $j$.
2. Compute its limb length, $LimbLength(j)$.
3. Subtract $LimbLength(j)$ from each row and column to
   produce $D^{\text{bald}}$ in which $j$ is a bald limb (length 0).
4. Remove the $j$-th row and column of the matrix to
   form the $(n - 1)$ x $(n - 1)$ matrix $D^{\text{trim}}$.
5. Construct $Tree(D^{\text{trim}})$.
6. Identify the point in $Tree(D^{\text{trim}})$ where leaf $j$ should
   be attached.
7. Attach $j$ by an edge of length $LimbLength(j)$ in order
   to form $Tree(D)$.
```

AdditivePhylogeny

```
AdditivePhylogeny($D$):
1. Pick an arbitrary leaf $j$.
2. Compute its limb length, $LimbLength(j)$.
3. Subtract $LimbLength(j)$ from each row and column to produce $D^{bald}$ in which $j$ is a bald limb (length 0).
4. Remove the $j$-th row and column of the matrix to form the $(n - 1)$ x $(n - 1)$ matrix $D^{trim}$.
5. Construct $Tree(D^{trim})$.
6. Identify the point in $Tree(D^{trim})$ where leaf $j$ should be attached.
7. Attach $j$ by an edge of length $LimbLength(j)$ in order to form $Tree(D)$.
```

Attaching a Limb

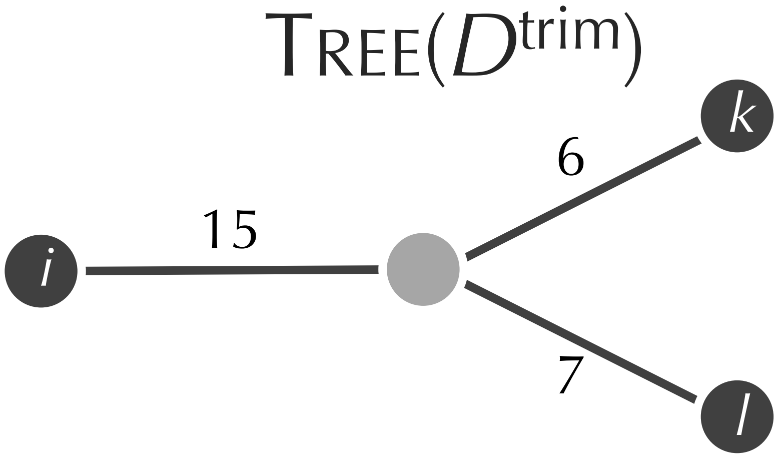
$$D^{\text{bald}} \quad \begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{i} & 0 & 11 & 21 & 22 \\ \boldsymbol{j} & 11 & 0 & 10 & 11 \\ \boldsymbol{k} & 21 & 10 & 0 & 13 \\ \boldsymbol{l} & 22 & 11 & 13 & 0 \end{array}$$



Limb Length Theorem: the length of the limb of j is equal to the minimum value of $(D^{\text{bald}}_{i,j} + D^{\text{bald}}_{j,k} - D^{\text{bald}}_{i,k})/2$ over all leaves i and k.

Attaching a Limb

$$D^{\text{bald}} \quad \begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{i} & 0 & \mathbf{11} & \mathbf{21} & 22 \\ \boldsymbol{j} & 11 & 0 & \mathbf{10} & 11 \\ \boldsymbol{k} & 21 & 10 & 0 & 13 \\ \boldsymbol{l} & 22 & 11 & 13 & 0 \end{array}$$



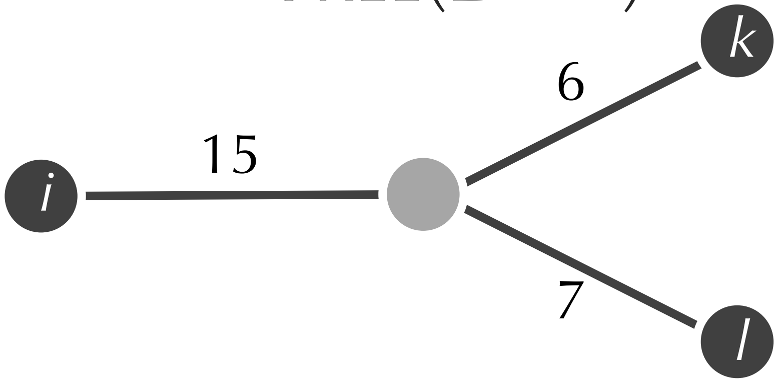
Limb Length Theorem: the length of the limb of *j* is equal to the minimum value of $(D^{\text{bald}}_{i,j} + D^{\text{bald}}_{j,k} - D^{\text{bald}}_{i,k})/2$ over all leaves *i* and *k*.

$$(D^{\text{bald}}_{i,j} + D^{\text{bald}}_{j,k} - D^{\text{bald}}_{i,k})/2 = \mathbf{0}$$

Attaching a Limb

$$D^{\text{bald}} \quad \begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{i} & 0 & \mathbf{11} & \mathbf{21} & 22 \\ \boldsymbol{j} & 11 & 0 & \mathbf{10} & 11 \\ \boldsymbol{k} & 21 & 10 & 0 & 13 \\ \boldsymbol{l} & 22 & 11 & 13 & 0 \end{array}$$

$\text{TREE}(D^{\text{trim}})$

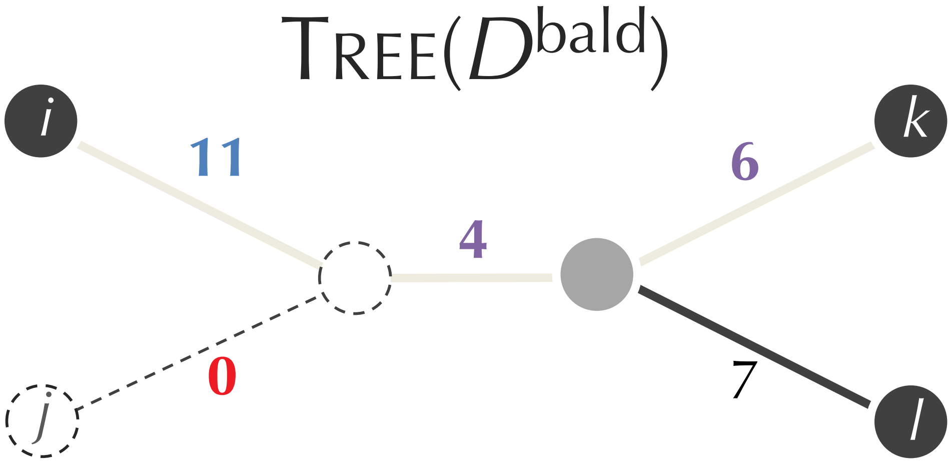


$$(D^{\text{bald}}_{i,j} + D^{\text{bald}}_{j,k} - D^{\text{bald}}_{i,k})/2 = \mathbf{0}$$

$$D^{\text{bald}}_{i,j} + D^{\text{bald}}_{j,k} = D^{\text{bald}}_{i,k}$$

Attaching a Limb

$$D^{\text{bald}} \quad \begin{array}{c|cccc} & i & j & k & l \\ \hline i & 0 & 11 & 21 & 22 \\ j & 11 & 0 & 10 & 11 \\ k & 21 & 10 & 0 & 13 \\ l & 22 & 11 & 13 & 0 \end{array}$$



The attachment point for *j* is found on the path between leaves *i* and *k* at distance $D^{\text{bald}}_{i,j}$ from *i*.

$$D^{\text{bald}}_{i,j} + D^{\text{bald}}_{j,k} = \boxed{D^{\text{bald}}_{i,k}}$$

AdditivePhylogeny

```
AdditivePhylogeny($D$):
1.  Pick an arbitrary leaf $j$.
2.  Compute its limb length, $LimbLength(j)$.
3.  Subtract $LimbLength(j)$ from each row and column to
    produce $D^{bald}$ in which $j$ is a bald limb (length 0).
4.  Remove the $j$-th row and column of the matrix to
    form the $(n - 1) \times (n - 1)$ matrix $D^{trim}$.
5.  Construct $Tree(D^{trim})$.
6.  Identify the point in $Tree(D^{trim})$ where leaf $j$ should
    be attached.
7.  Attach $j$ by an edge of length $LimbLength(j)$ in order
    to form $Tree(D)$.
```

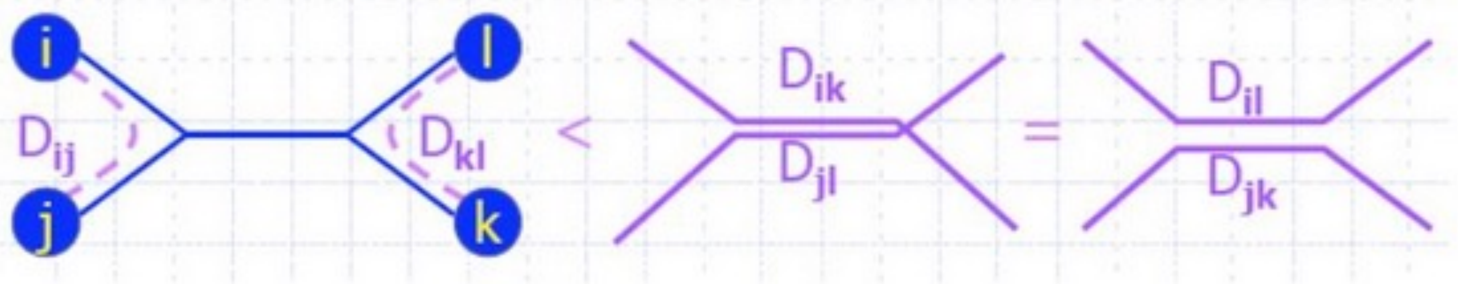
Code Challenge: Implement **AdditivePhylogeny**.

Exercise

AdditivePhylogeny(*D*):

1. A distance matrix corresponding to a tree is called additive
2. D is additive if and only if : For every four indices I,j,k,l the maximum and the median of the three pairwise sum are identical.

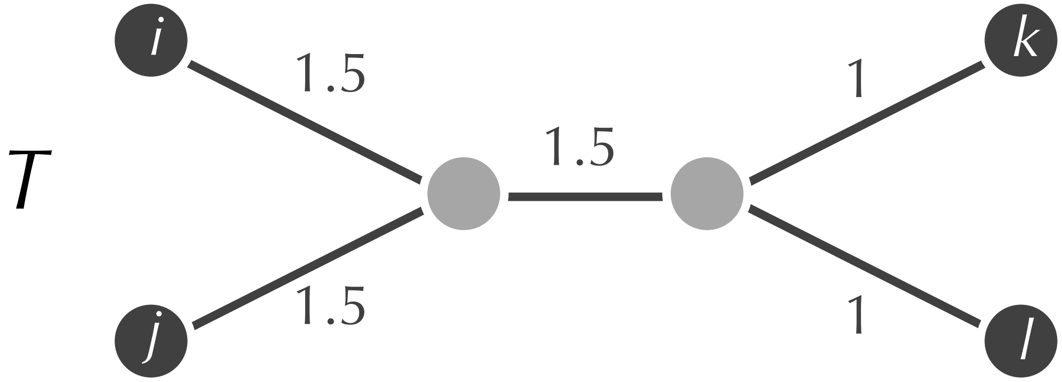
$$D_{ij}+D_{kl} < D_{ik}+D_{jl} = D_{il}+D_{jk}$$



Sum of Squared Errors

$$Discrepancy(T, D) = \Sigma_{1 \leq i < j \leq n} (d_{i,j}(T) - D_{i,j})^2$$

$$= 1^2 + 1^2 = 2$$



D

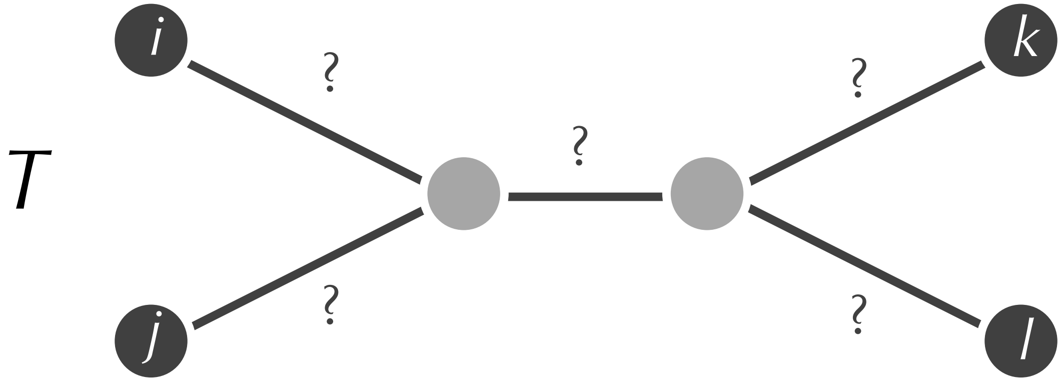
	i	j	k	l
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0

d

	i	j	k	l
i	0	3	4	4
j	3	0	4	4
k	4	4	0	2
l	4	4	2	0

Sum of Squared Errors

Exercise Break: Assign lengths to edges in *T* in order to minimise *Discrepancy*(*T*, *D*).



D

	i	***j***	***k***	***l***
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0

d

	i	***j***	***k***	***l***
i	0	?	?	?
j	?	0	?	?
k	?	?	0	?
l	?	?	?	0

Least-Squares Phylogeny

Least-Squares Distance-Based Phylogeny Problem: *Given a distance matrix, find the tree that minimizes the sum of squared errors*.

- **Input:** An n x n distance matrix D.
- **Output:** A weighted tree T with n leaves minimising *Discrepancy*(T, D) over all weighted trees with n leaves.

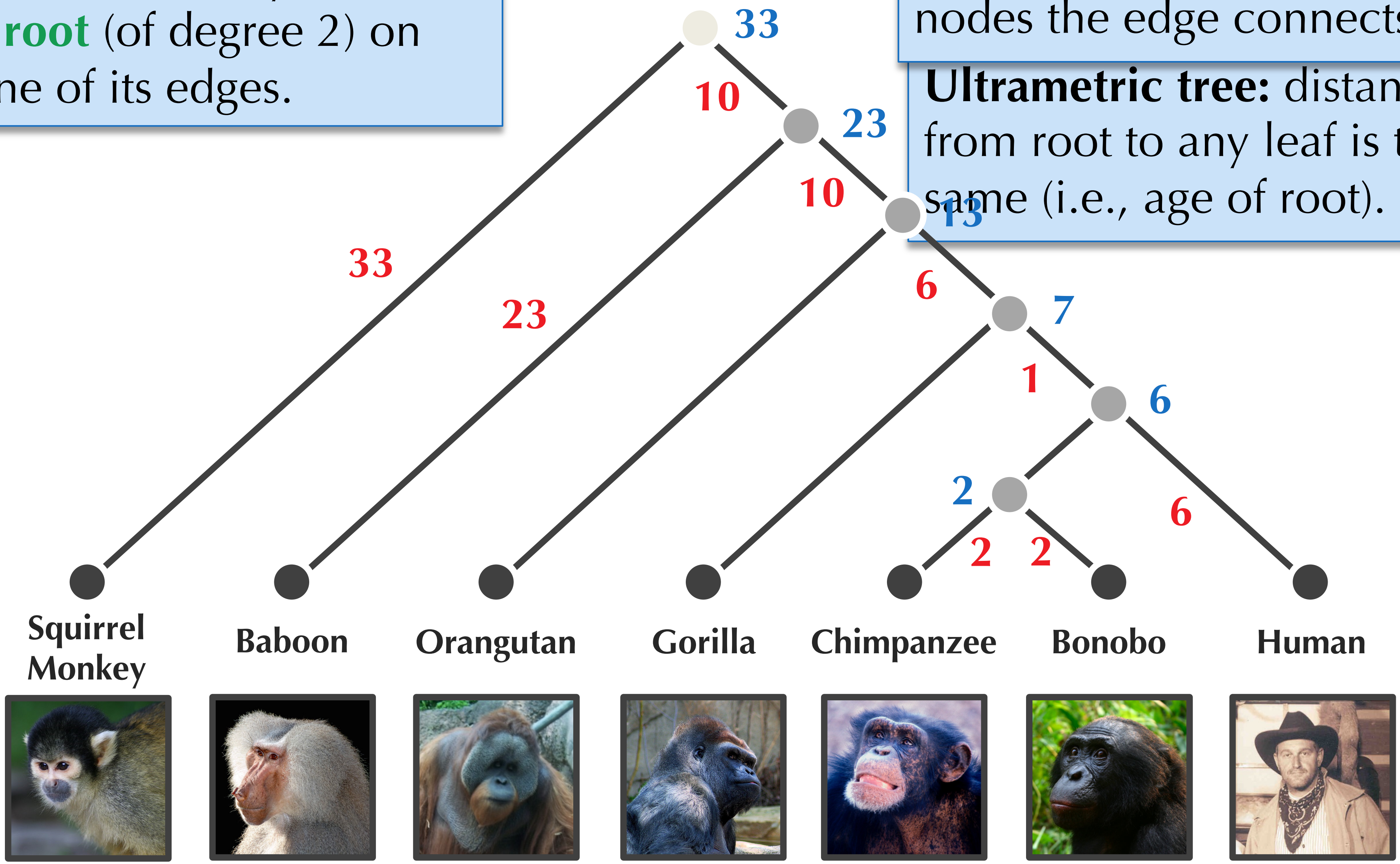
Unfortunately, this problem is *NP*-Complete...

Ultrametric Trees

Rooted binary tree: an unrooted binary tree with a **root** (of degree 2) on one of its edges.

edge weights: correspond to difference in ages on the nodes the edge connects.

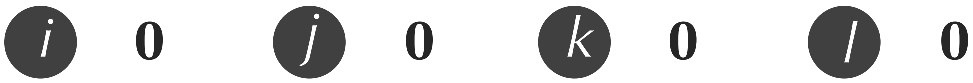
Ultrametric tree: distance from root to any leaf is the same (i.e., age of root).



UPGMA: A Clustering Heuristic

1. Form a cluster for each present-day species, each containing a single leaf.

	i	***j***	***k***	***l***
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0



UPGMA: A Clustering Heuristic

2. Find the two closest clusters C_1 and C_2 according to the average distance

$$D_{\text{avg}}(C_1, C_2) = \Sigma_{i \text{ in } C1,\, j \text{ in } C2}\; D_{i,j} \,/\, |C_1| \bullet |C_2|$$

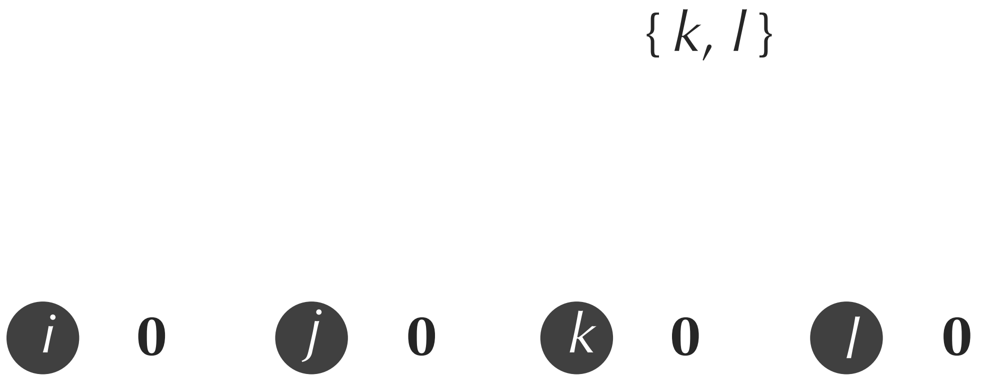
where $|C|$ denotes the number of elements in C.

	i	***j***	***k***	***l***
i	0	3	4	3
j	3	0	4	5
k	4	4	0	**2**
l	3	5	**2**	0

UPGMA: A Clustering Heuristic

3. Merge C_1 and C_2 into a single cluster C.

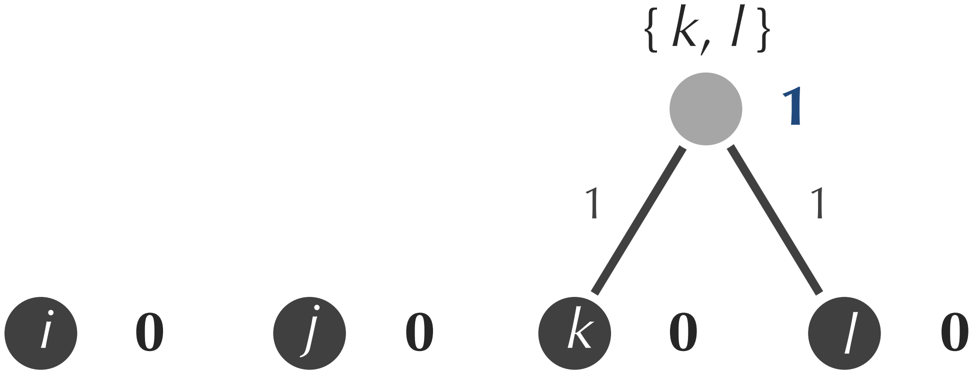
	i	***j***	***k***	***l***
i	0	3	4	3
j	3	0	4	5
k	4	4	0	**2**
l	3	5	**2**	0



UPGMA: A Clustering Heuristic

4. Form a new node for C and connect to C_1 and C_2 by an edge. Set age of C as $D_{avg}(C_1, C_2)/2$.

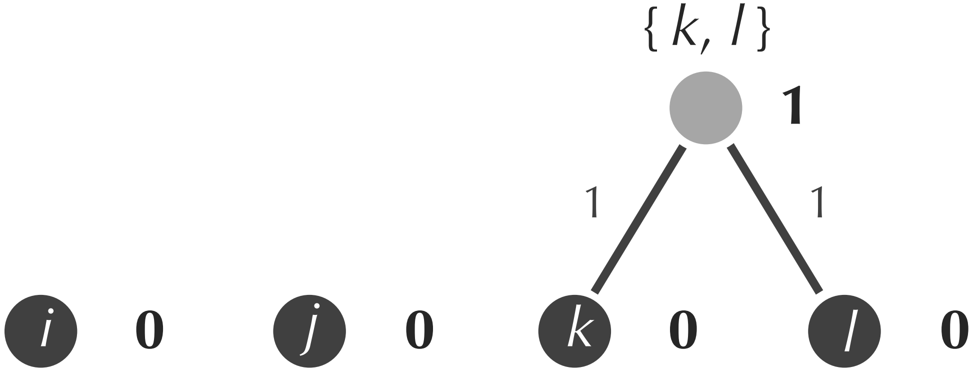
$$\begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{i} & 0 & 3 & 4 & 3 \\ \boldsymbol{j} & 3 & 0 & 4 & 5 \\ \boldsymbol{k} & 4 & 4 & 0 & \mathbf{2} \\ \boldsymbol{l} & 3 & 5 & \mathbf{2} & 0 \end{array}$$



UPGMA: A Clustering Heuristic

5. Update the distance matrix by computing the average distance between each pair of clusters.

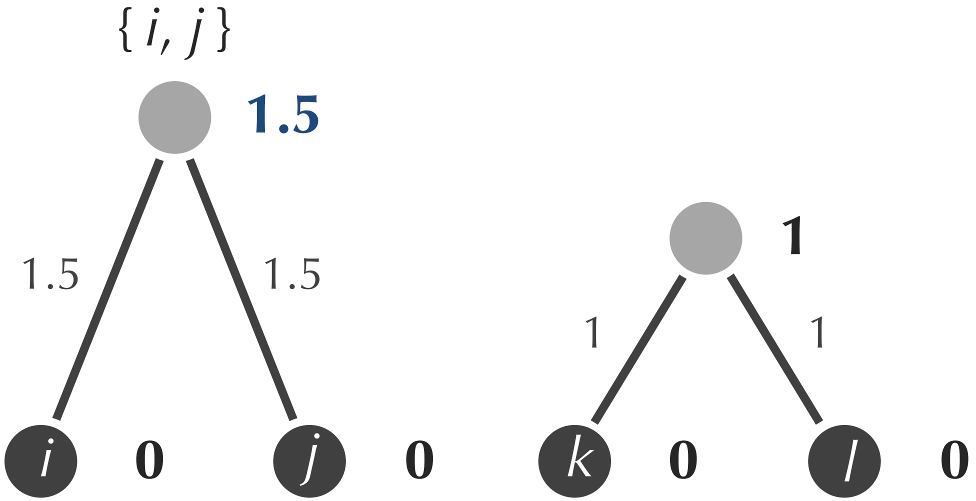
	i	***j***	**{*k*, *l*}**
i	0	3	3.5
j	3	0	4.5
{*k*, *l*}	3.5	4.5	0



UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.

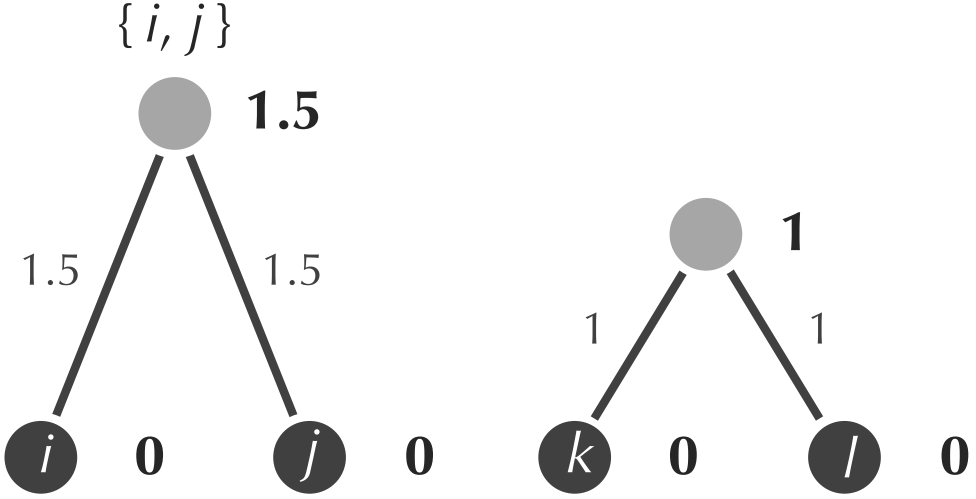
	i	***j***	**{*k*, *l*}**
i	0	**3**	3.5
j	**3**	0	4.5
{*k*, *l*}	3.5	4.5	0



UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.

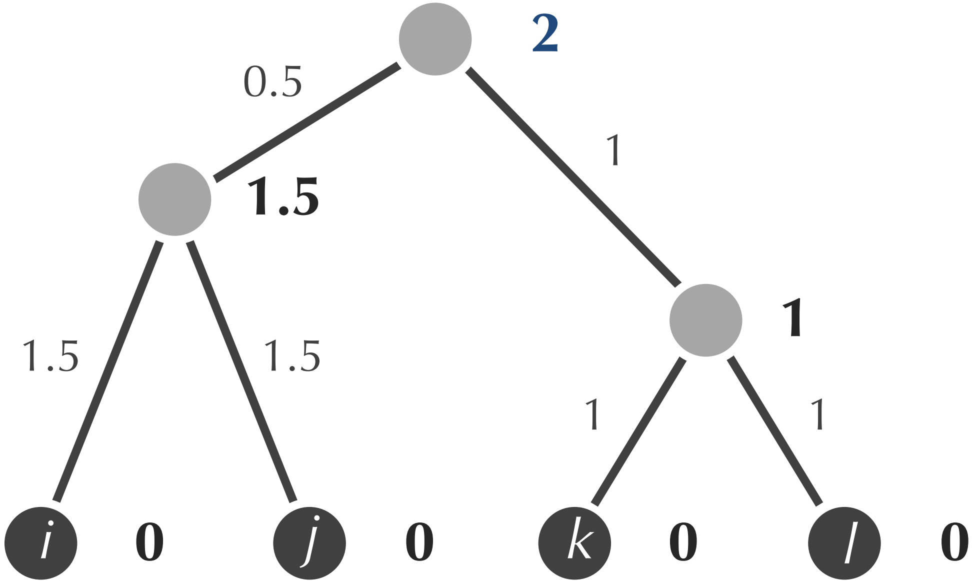
	{*i, j*}	**{ *k, l* }**
{*i, j*}	0	4
{ *k, l* }	4	0



UPGMA: A Clustering Heuristic

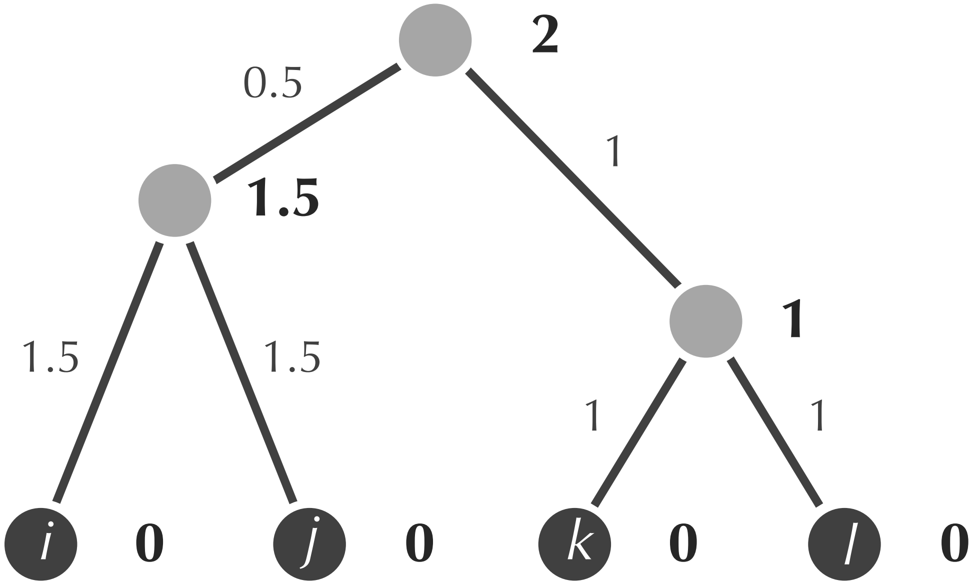
6. Iterate until a single cluster contains all species.

	{*i*, *j*}	{ *k*, *l* }
{*i*, *j*}	0	**4**
{ *k*, *l* }	**4**	0



UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.

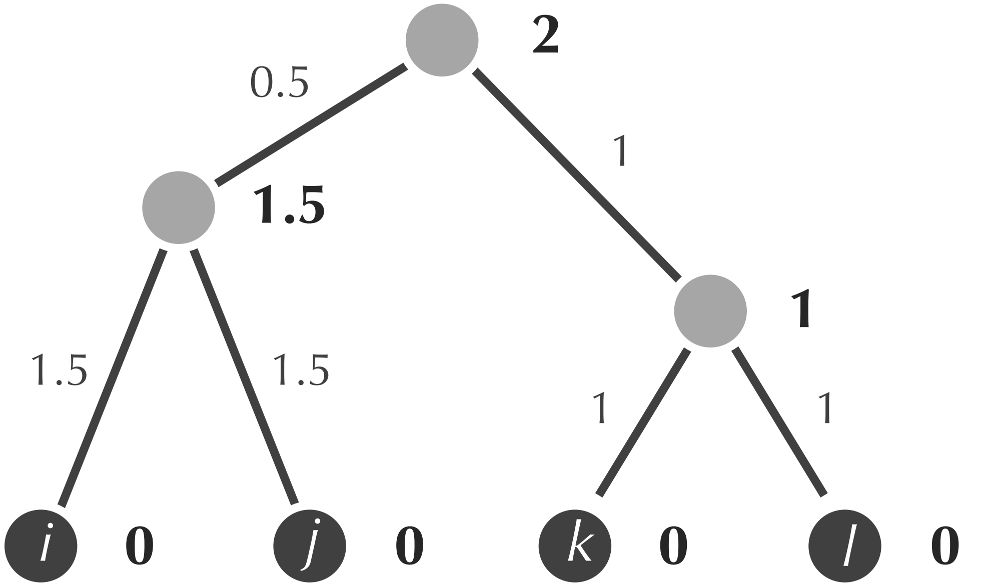


UPGMA: A Clustering Heuristic

```
UPGMA($D$):
1. Form a cluster for each present-day species, each containing a single leaf.
2. Find the two closest clusters $C_1$ and $C_2$ according to the average distance
        $D_{\text{avg}}(C_1, C_2) = \Sigma_{i \text{ in } C1,\ j \text{ in } C2}\ D_{i,j} / |C_1| \bullet |C_2|$
   where $|C|$ denotes the number of elements in $C$
3. Merge $C_1$ and $C_2$ into a single cluster $C$.
4. Form a new node for $C$ and connect to $C_1$ and $C_2$ by an edge. Set age of $C$ as $D_{\text{avg}}(C_1, C_2)/2$.
5. Update the distance matrix by computing the average distance between each pair of clusters.
6. Iterate steps 2-5 until a single cluster contains all species.
```

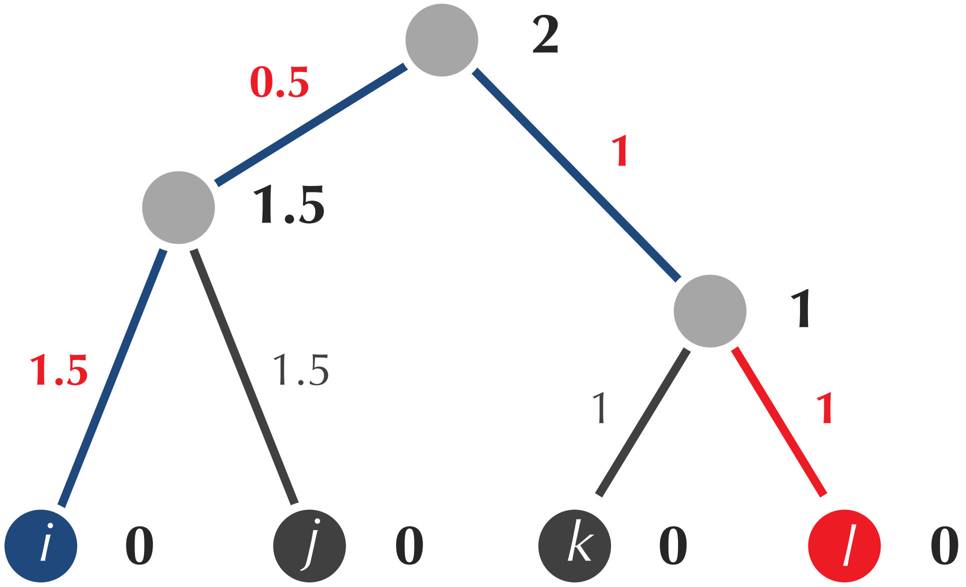
UPGMA Doesn't "Fit" a Tree to a Matrix

	i	***j***	***k***	***l***
i	0	3	4	3
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0



UPGMA Doesn’t “Fit” a Tree to a Matrix

	i	*j*	*k*	*l*
i	0	3	4	**3**
j	3	0	4	5
k	4	4	0	2
l	3	5	2	0



In Summary...

- **AdditivePhylogeny:**
  - good: produces the tree fitting an *additive* matrix
  - bad: fails completely on a *non-additive* matrix
- **UPGMA:**
  - good: produces a tree for any matrix
  - bad: tree doesn't necessarily fit an additive matrix
- **?????:**
  - good: produces the tree fitting an additive matrix
  - good: provides heuristic for a non-additive matrix

Neighbor-Joining Theorem

Given an n x n distance matrix D, its **neighbor-joining matrix** is the matrix D^* defined as

$$D^*_{i,j} = (n - 2)\bullet D_{i,j} - TotalDistance_D(i) - TotalDistance_D(j)$$

where $TotalDistance_D(i)$ is the sum of distances from i to all other leaves.

$$D \quad \begin{array}{c|cccc|c} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} & \boldsymbol{TotalDistance_D} \\ \boldsymbol{i} & 0 & 13 & 21 & 22 & 56 \\ \boldsymbol{j} & 13 & 0 & 12 & 13 & 38 \\ \boldsymbol{k} & 21 & 12 & 0 & 13 & 46 \\ \boldsymbol{l} & 22 & 13 & 13 & 0 & 48 \end{array}$$

$$D^* \quad \begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \boldsymbol{i} & 0 & -68 & -60 & -60 \\ \boldsymbol{j} & -68 & 0 & -60 & -60 \\ \boldsymbol{k} & -60 & -60 & 0 & -68 \\ \boldsymbol{l} & -60 & -60 & -68 & 0 \end{array}$$

Neighbor-Joining Theorem

Neighbor-Joining Theorem: If D is additive, then the smallest element of D^* corresponds to neighboring leaves in $Tree(D)$.

D

	i	*j*	*k*	*l*	$TotalDistance_D$
i	0	13	21	22	56
j	13	0	**12**	13	38
k	21	**12**	0	13	46
l	22	13	13	0	48

D^*

	i	*j*	*k*	*l*
i	0	**-68**	-60	-60
j	**-68**	0	-60	-60
k	-60	-60	0	**-68**
l	-60	-60	**-68**	0

Neighbor-Joining in Action

$$D^* \quad \begin{array}{cccccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} & \boldsymbol{TotalDistance_D} \\ \boldsymbol{i} & 0 & -68 & -60 & -60 & 56 \\ \boldsymbol{j} & -68 & 0 & -60 & -60 & 38 \\ \boldsymbol{k} & -60 & -60 & 0 & -68 & 46 \\ \boldsymbol{l} & -60 & -60 & -68 & 0 & 48 \end{array}$$

1. Construct neighbor-joining matrix *D** from *D*.

Neighbor-Joining in Action

$$D^* \quad \begin{array}{c|cccc|c}
 & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} & \boldsymbol{TotalDistance_D} \\
\boldsymbol{i} & 0 & \mathbf{-68} & -60 & -60 & 56 \\
\boldsymbol{j} & \mathbf{-68} & 0 & -60 & -60 & 38 \\
\boldsymbol{k} & -60 & -60 & 0 & \mathbf{-68} & 46 \\
\boldsymbol{l} & -60 & -60 & \mathbf{-68} & 0 & 48
\end{array}$$

2. Find a minimum element $D^*_{i,j}$ of D^*.

Neighbor-Joining in Action

$$D^* \quad \begin{array}{c|cccc|c} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} & \boldsymbol{TotalDistance_D} \\ \hline \boldsymbol{i} & 0 & \mathbf{-68} & -60 & -60 & 56 \\ \boldsymbol{j} & \mathbf{-68} & 0 & -60 & -60 & 38 \\ \boldsymbol{k} & -60 & -60 & 0 & -68 & 46 \\ \boldsymbol{l} & -60 & -60 & -68 & 0 & 48 \end{array}$$

2. Find a minimum element $D^*_{i,j}$ of D^*.

Neighbor-Joining in Action

$$D^* \quad \begin{array}{c|cccc|c}
 & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} & \boldsymbol{TotalDistance_D} \\
\hline
\boldsymbol{i} & 0 & \mathbf{-68} & -60 & -60 & \mathbf{56} \\
\boldsymbol{j} & \mathbf{-68} & 0 & -60 & -60 & \mathbf{38} \\
\boldsymbol{k} & -60 & -60 & 0 & -68 & 46 \\
\boldsymbol{l} & -60 & -60 & -68 & 0 & 48
\end{array}$$

$$\begin{aligned} \Delta_{i,j} &= (\mathbf{56} - \mathbf{38}) \,/\, (4 - 2) \\ &= 9 \end{aligned}$$

3. Compute $\Delta_{i,j} = (TotalDistance_D(i) - TotalDistance_D(j)) \,/\, (n - 2)$.

Neighbor-Joining in Action

$$D \quad \begin{array}{cccccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} & \boldsymbol{TotalDistance_D} \\ \boldsymbol{i} & 0 & \mathbf{13} & 21 & 22 & 56 \\ \boldsymbol{j} & \mathbf{13} & 0 & 12 & 13 & 38 \\ \boldsymbol{k} & 21 & 12 & 0 & 13 & 46 \\ \boldsymbol{l} & 22 & 13 & 13 & 0 & 48 \end{array}$$

$$\Delta_{i,j} = (56 - 38) / (4 - 2)$$
$$= 9$$

$LimbLength(i) = ½(13 + 9) = 11$

$LimbLength(i) = ½(13 - 9) = 2$

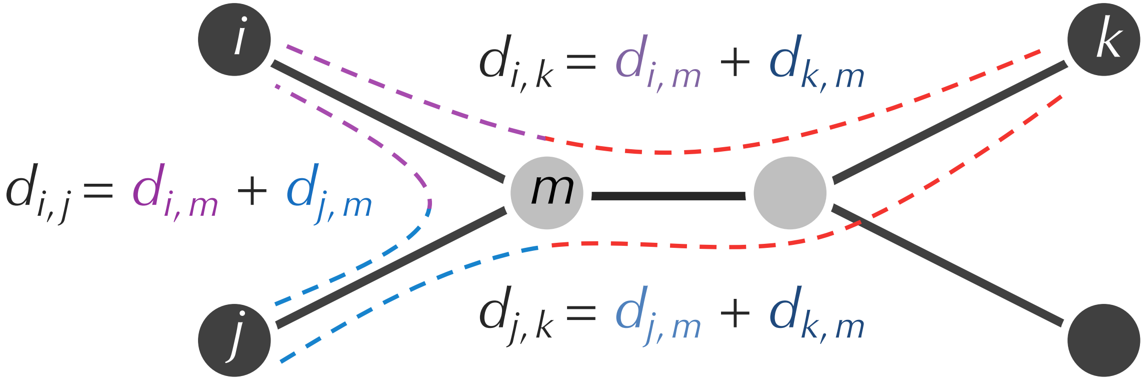
4. Set $LimbLength(i)$ equal to $½(D_{i,j} + \Delta_{i,j})$ and $LimbLength(j)$ equal to $½(D_{i,j} - \Delta_{i,j})$.

Neighbor-Joining in Action

$$D' \qquad \begin{array}{ccccc} & \boldsymbol{m} & \boldsymbol{k} & \boldsymbol{l} & \qquad \boldsymbol{TotalDistance_D} \\ \boldsymbol{m} & 0 & 10 & 11 & \qquad 21 \\ \boldsymbol{k} & 10 & 0 & 13 & \qquad 23 \\ \boldsymbol{l} & 11 & 13 & 0 & \qquad 24 \end{array}$$

5. Form a matrix D' by removing i-th and j-th row/column from D and adding an m-th row/column such that for any k, $D_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j})\ /\ 2$.

Flashback: Computation of $d_{k,m}$


i
k
j
m
$d_{i,k} = d_{i,m} + d_{k,m}$
$d_{i,j} = d_{i,m} + d_{j,m}$
$d_{j,k} = d_{j,m} + d_{k,m}$

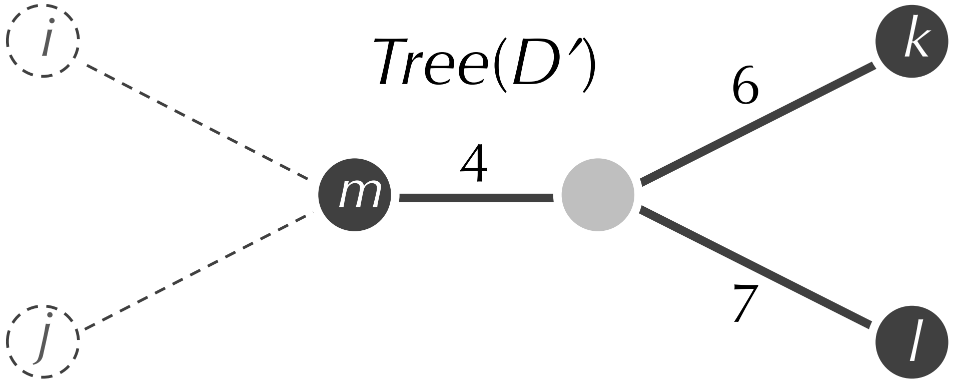
$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$$

$$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$$

Neighbor-Joining in Action

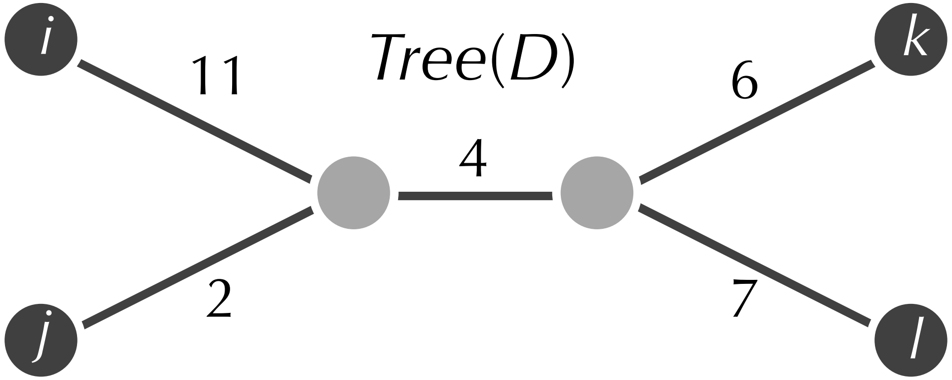
$$D' \quad \begin{array}{c|ccc} & \boldsymbol{m} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{m} & 0 & 10 & 11 \\ \boldsymbol{k} & 10 & 0 & 13 \\ \boldsymbol{l} & 11 & 13 & 0 \end{array}$$



6. Apply **NeighborJoining** to D' to obtain $Tree(D')$.

Neighbor-Joining in Action

$$D' \quad \begin{array}{c|ccc} & \boldsymbol{m} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{m} & 0 & 10 & 11 \\ \boldsymbol{k} & 10 & 0 & 13 \\ \boldsymbol{l} & 11 & 13 & 0 \end{array}$$



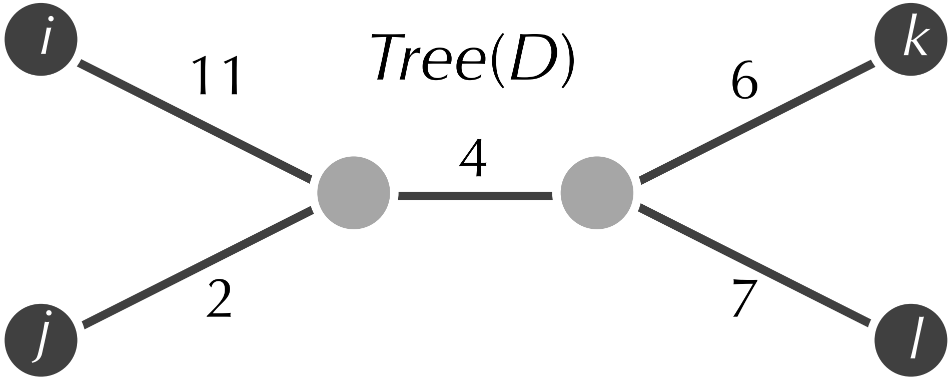
$LimbLength(i) = ½(13 + 9) = 11$

$LimbLength(i) = ½(13 – 9) = 2$

7. Reattach limbs of *i* and *j* to obtain *Tree*(*D*).

Neighbor-Joining in Action

$$D' \quad \begin{array}{c|ccc} & \boldsymbol{m} & \boldsymbol{k} & \boldsymbol{l} \\ \hline \boldsymbol{m} & 0 & 10 & 11 \\ \boldsymbol{k} & 10 & 0 & 13 \\ \boldsymbol{l} & 11 & 13 & 0 \end{array}$$



7. Reattach limbs of *i* and *j* to obtain *Tree*(*D*).

Neighbor-Joining

```
NeighborJoining($D$):
1. Construct neighbor-joining matrix $D^*$ from $D$.
2. Find a minimum element $D^*_{i,j}$ of $D^*$.
3. Compute $\Delta_{i,j} = (TotalDistance_D(i) - TotalDistance_D(j)) / (n - 2)$.
4. Set $LimbLength(i)$ equal to $\frac{1}{2}(D_{i,j} + \Delta_{i,j})$ and $LimbLength(j)$ equal to $\frac{1}{2}(D_{i,j} - \Delta_{i,j})$.
5. Form a matrix $D'$ by removing $i$-th and $j$-th row/column from $D$ and adding an $m$-th row/column such that for any $k$, $D_{k,m} = (D_{k,i} + D_{k,j} - D_{i,j}) / 2$.
6. Apply NeighborJoining to $D'$ to obtain $Tree(D')$.
7. Reattach limbs of $i$ and $j$ to obtain $Tree(D)$.
```

Reference: Saitou, N.; Nei, M. (1 July 1987). "The neighbor-joining method: a new method for reconstructing phylogenetic trees". Molecular Biology and Evolution. ***4*** *(4): 406–425. doi:10.1093/oxfordjournals.molbev.a040454. PMID 3447015.*

Complexity

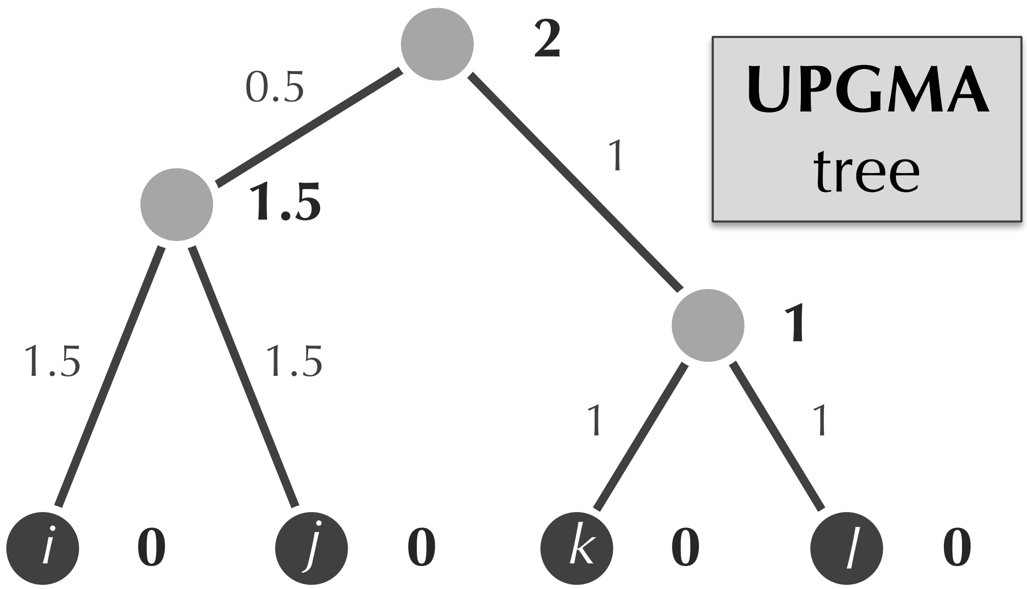
Exercise Break, check the following: Neighbor joining on a set of r taxa (species, leaves) requires r-3 iterations. At each step one has to build and search a *D** matrix. Initially the *D** matrix is size r^2, then the next step it is $(r -1)^2$, etc. This leads to a time complexity of $O(r^3)$.

Code Challenge: Implement **NeighborJoining**.

Neighbor-Joining

Exercise Break: Find the tree returned by **NeighborJoining** on the following non-additive matrix. How does the result compare with the tree produced by **UPGMA**?

$$D \quad \begin{array}{c|cccc} & \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ \boldsymbol{i} & 0 & 3 & 4 & 3 \\ \boldsymbol{j} & 3 & 0 & 4 & 5 \\ \boldsymbol{k} & 4 & 4 & 0 & 2 \\ \boldsymbol{l} & 3 & 5 & 2 & 0 \end{array}$$



UPGMA tree

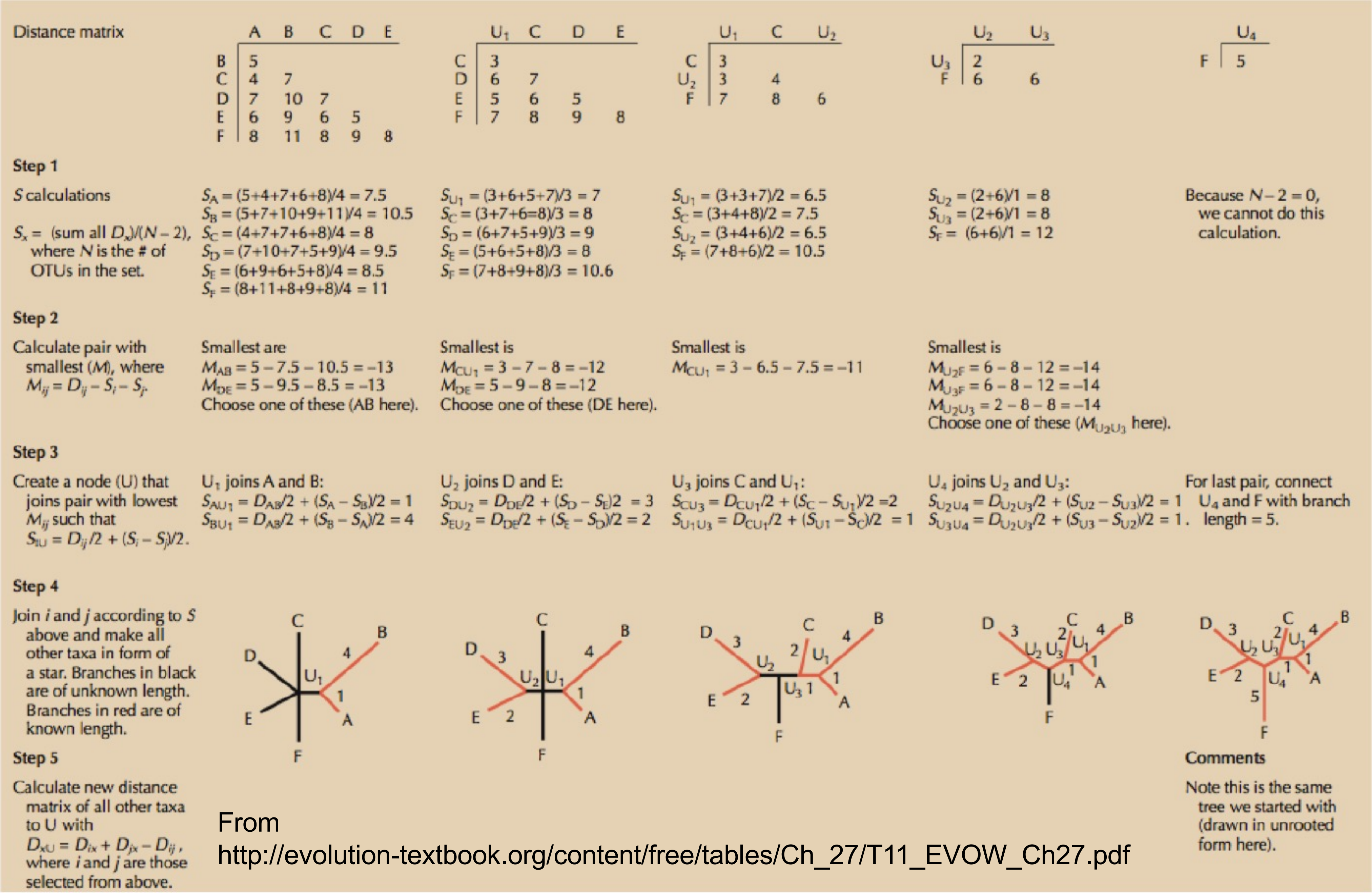
Weakness of Distance-Based Methods

Distance-based algorithms for evolutionary tree reconstruction say nothing about ancestral states at internal nodes.

We *lost* information when we converted a multiple alignment to a distance matrix…

Species	Alignment	Distance Matrix			
		Chimp	**Human**	**Seal**	**Whale**
Chimp	`ACGTAGGCCT`	0	3	6	4
Human	`ATGTAAGACT`	3	0	7	5
Seal	`TCGAGAGCAC`	6	7	0	2
Whale	`TCGAAAGCAT`	4	5	2	0

Example and RECAP (note different notation)

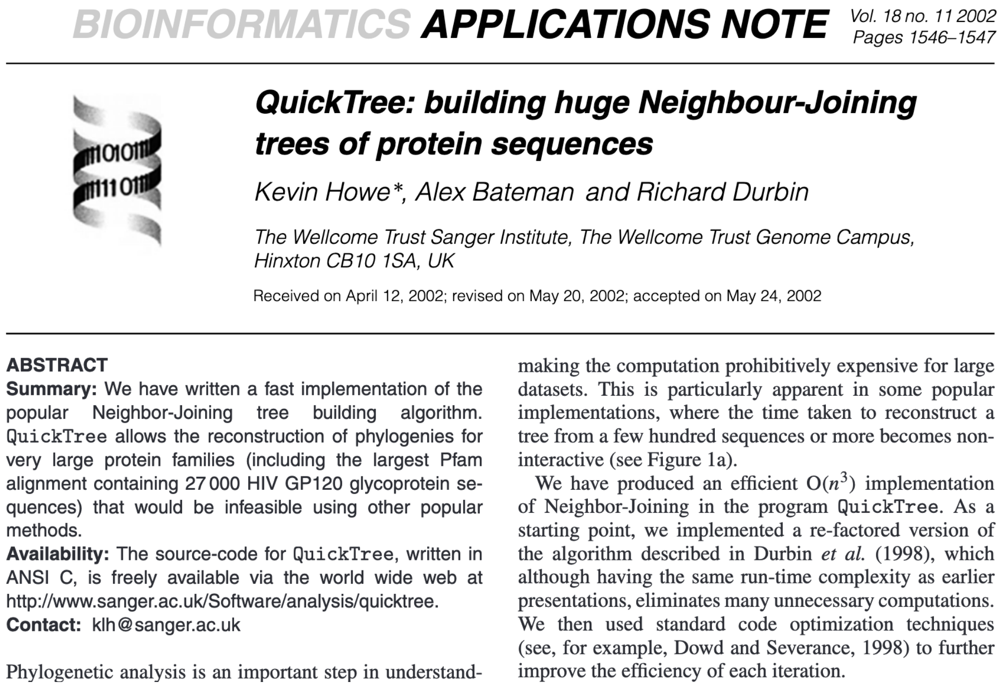


Distance matrix	$\begin{array}{cccccc} & A & B & C & D & E \\ B & 5 \\ C & 4 & 7 \\ D & 7 & 10 & 7 \\ E & 6 & 9 & 6 & 5 \\ F & 8 & 11 & 8 & 9 & 8 \end{array}$	$\begin{array}{ccccc} & U_1 & C & D & E \\ C & 3 \\ D & 6 & 7 \\ E & 5 & 6 & 5 \\ F & 7 & 8 & 9 & 8 \end{array}$	$\begin{array}{cccc} & U_1 & C & U_2 \\ C & 3 \\ U_2 & 3 & 4 \\ F & 7 & 8 & 6 \end{array}$	$\begin{array}{ccc} & U_2 & U_3 \\ U_3 & 2 \\ F & 6 & 6 \end{array}$	$\begin{array}{cc} & U_4 \\ F & 5 \end{array}$
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: $S_{CU_3} = D_{CU_1}/2 + (S_C - S_{U_1})/2 = 2$ $S_{U_1U_3} = D_{CU_1}/2 + (S_{U1} - S_C)/2 = 1$	U_4 joins U_2 and U_3: $S_{U_2U_4} = D_{U_2U_3}/2 + (S_{U2} - S_{U3})/2 = 1$ $S_{U_3U_4} = D_{U_2U_3}/2 + (S_{U3} - S_{U2})/2 = 1$.	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					**Comments**
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.					Note this is the same tree we started with (drawn in unrooted form here).

From
http://evolution-textbook.org/content/free/tables/Ch_27/T11_EVOW_Ch27.pdf

Implementation

Example (Non examinable)



BIOINFORMATICS **APPLICATIONS NOTE**

Vol. 18 no. 11 2002
Pages 1546–1547

QuickTree: building huge Neighbour-Joining trees of protein sequences

Kevin Howe, Alex Bateman and Richard Durbin*

The Wellcome Trust Sanger Institute, The Wellcome Trust Genome Campus,
Hinxton CB10 1SA, UK

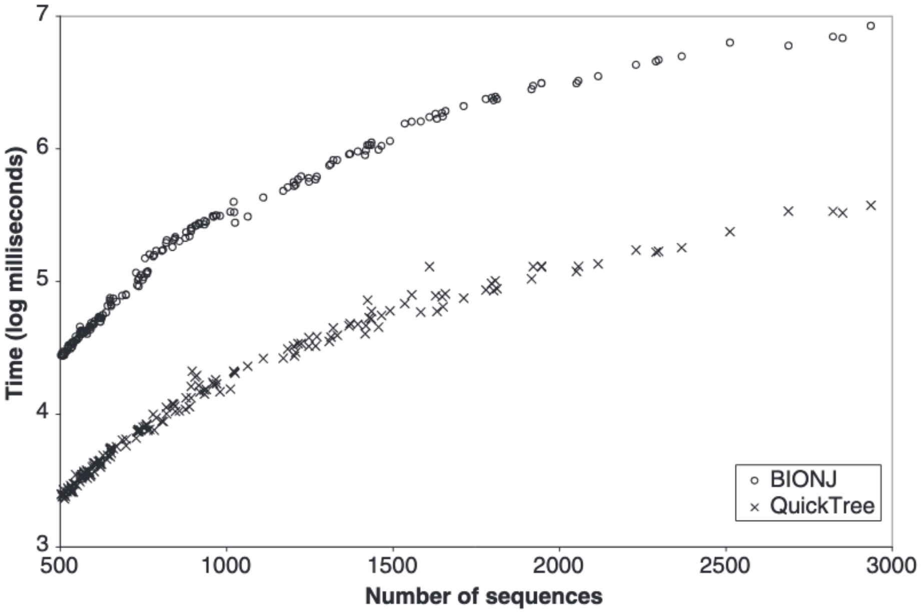
Received on April 12, 2002; revised on May 20, 2002; accepted on May 24, 2002

ABSTRACT

Summary: We have written a fast implementation of the popular Neighbor-Joining tree building algorithm. `QuickTree` allows the reconstruction of phylogenies for very large protein families (including the largest Pfam alignment containing 27 000 HIV GP120 glycoprotein sequences) that would be infeasible using other popular methods.
Availability: The source-code for `QuickTree`, written in ANSI C, is freely available via the world wide web at http://www.sanger.ac.uk/Software/analysis/quicktree.
Contact: klh@sanger.ac.uk

Phylogenetic analysis is an important step in understanding the computation prohibitively expensive for large datasets. This is particularly apparent in some popular implementations, where the time taken to reconstruct a tree from a few hundred sequences or more becomes non-interactive (see Figure 1a).

We have produced an efficient O(n^3) implementation of Neighbor-Joining in the program `QuickTree`. As a starting point, we implemented a re-factored version of the algorithm described in Durbin *et al.* (1998), which although having the same run-time complexity as earlier presentations, eliminates many unnecessary computations. We then used standard code optimization techniques (see, for example, Dowd and Severance, 1998) to further improve the efficiency of each iteration.



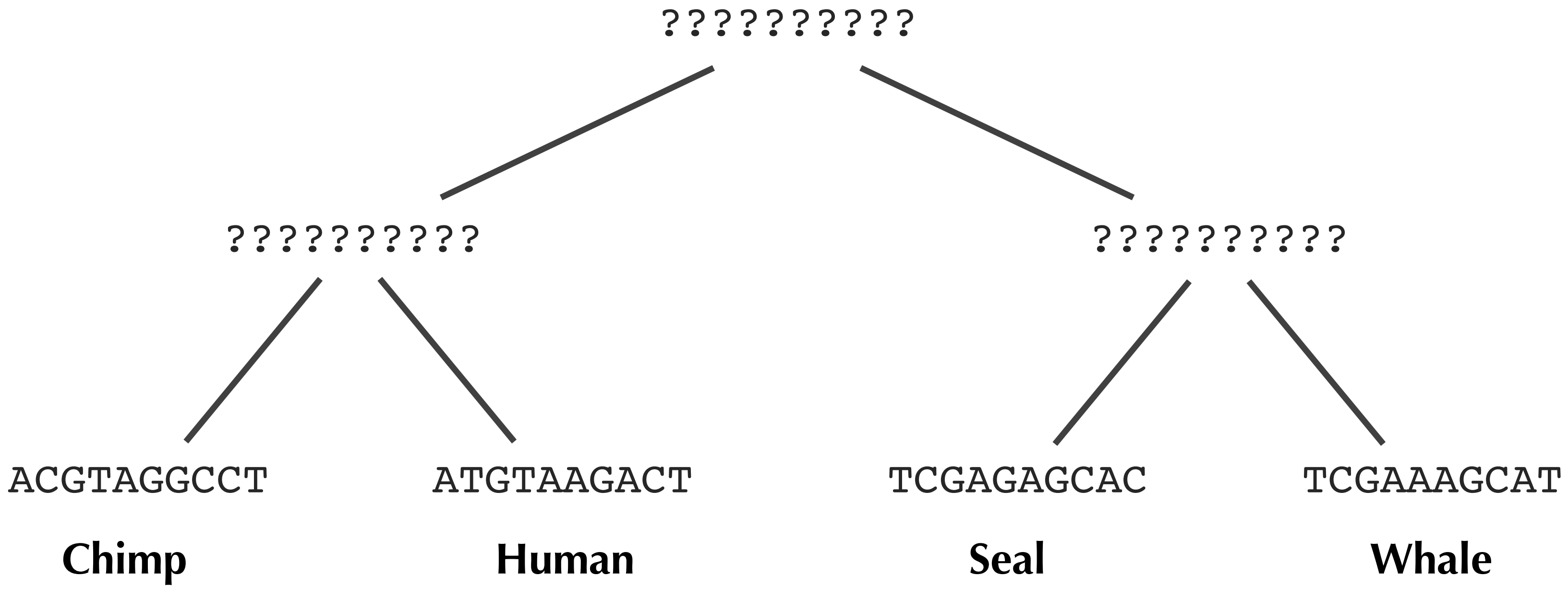
PARSIMONY: An Alignment As a Character Table

Here we do not use a distance matrix and we value each column of the alignment; each column could output a tree

Chimp	ACGTAGGCCT
Human	ATGTAAGACT
Seal	TCGAGAGCAC
Whale	TCGAAAGCAT

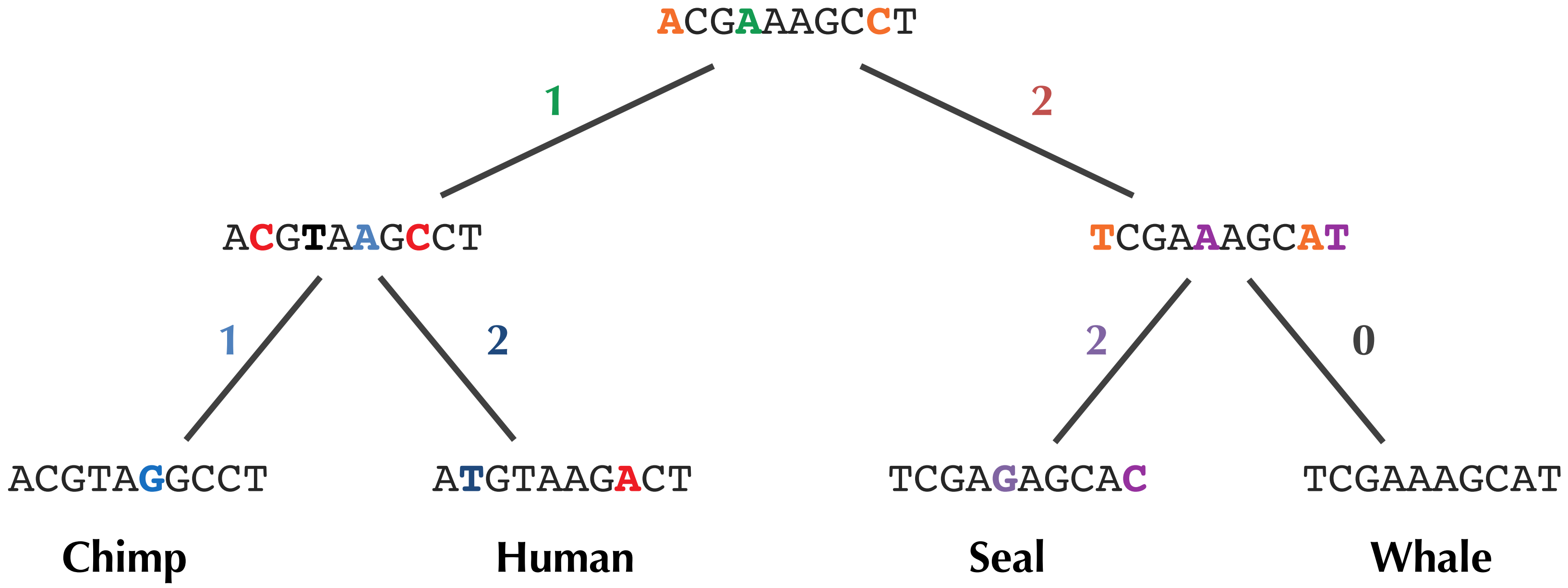
n species

m characters



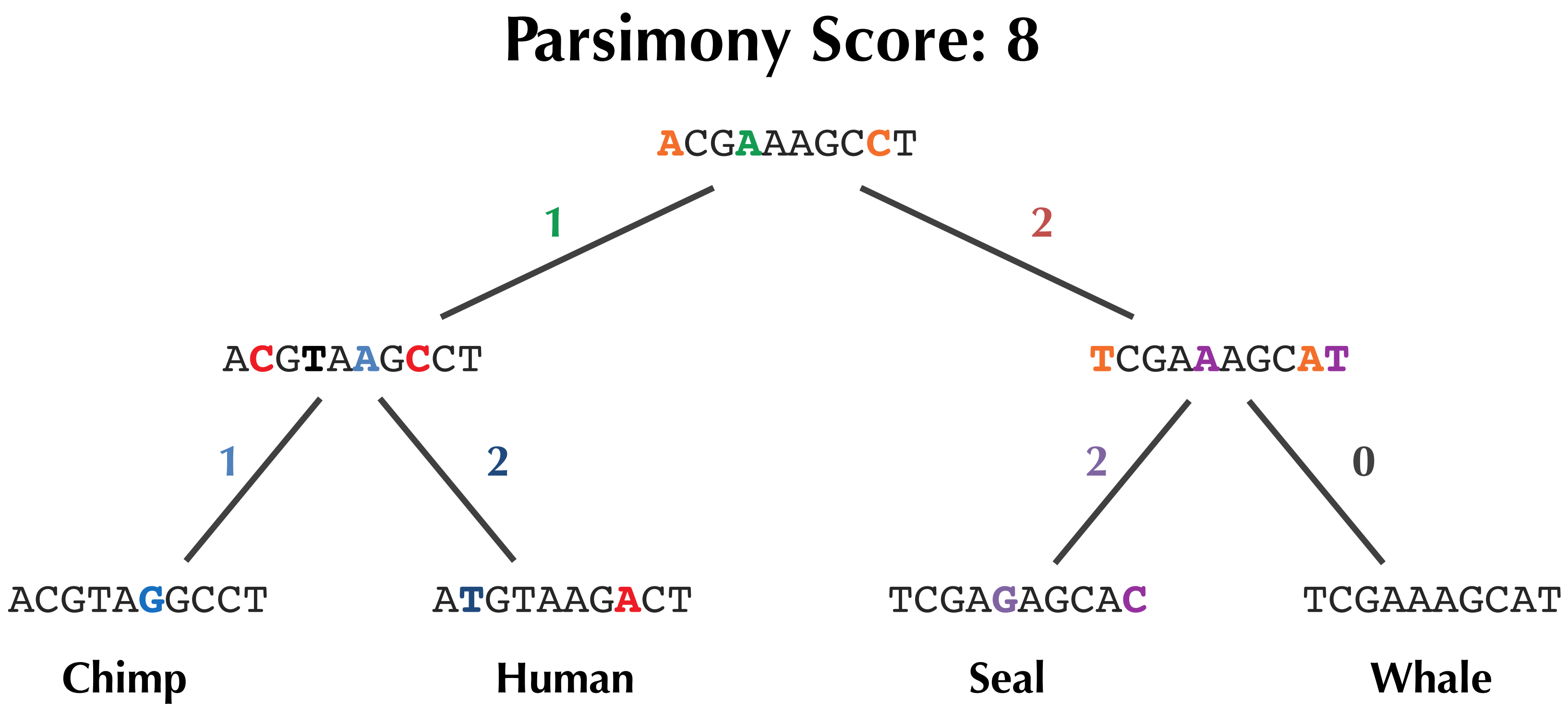
Toward a Computational Problem

Parsimony score: sum of Hamming distances along each edge.



Toward a Computational Problem

Parsimony score: sum of Hamming distances along each edge.



Toward a Computational Problem

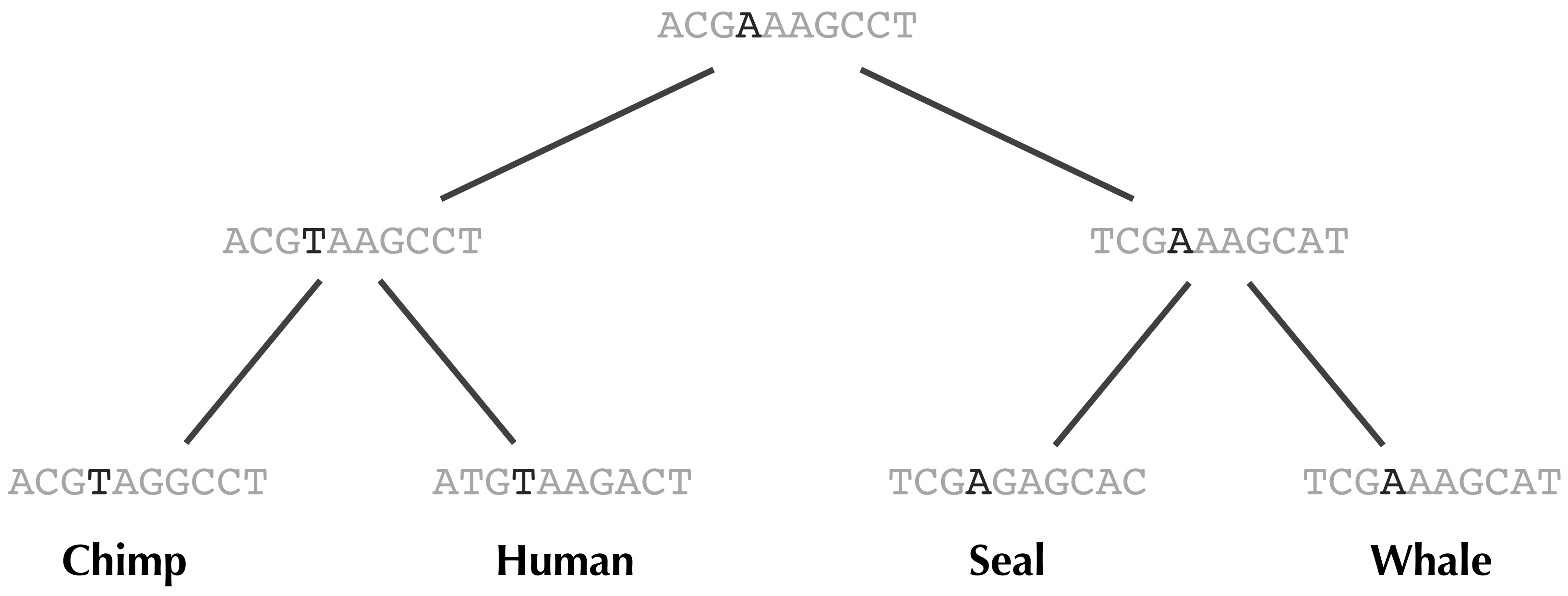
Small Parsimony Problem: *Find the most parsimonious labeling of the internal nodes of a rooted tree.*

- **Input:** A rooted binary tree with each leaf labeled by a string of length *m*.
- **Output:** A labeling of all other nodes of the tree by strings of length *m* that minimises the tree's parsimony score.

Is there any way we can simplify this problem statement?

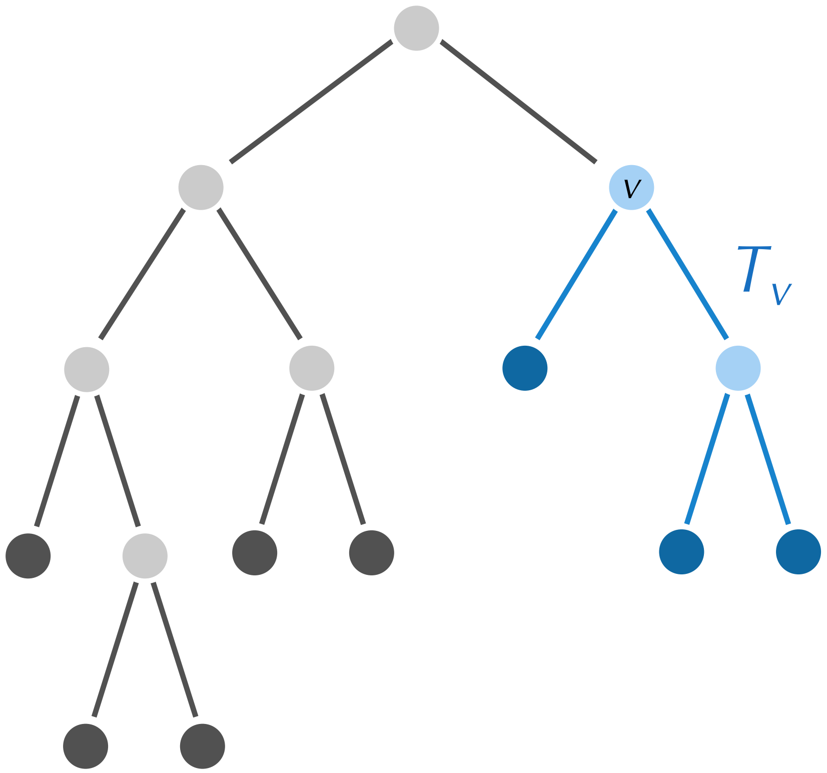
Small Parsimony Problem: *Find the most parsimonious labeling of the internal nodes of a rooted tree.*

- **Input:** A rooted binary tree with each leaf labeled by a **single symbol**.
- **Output:** A labeling of all other nodes of the tree by **single symbols** that minimises the tree's parsimony score.



A Dynamic Programming Algorithm

Let T_v denote the subtree of T whose root is v.



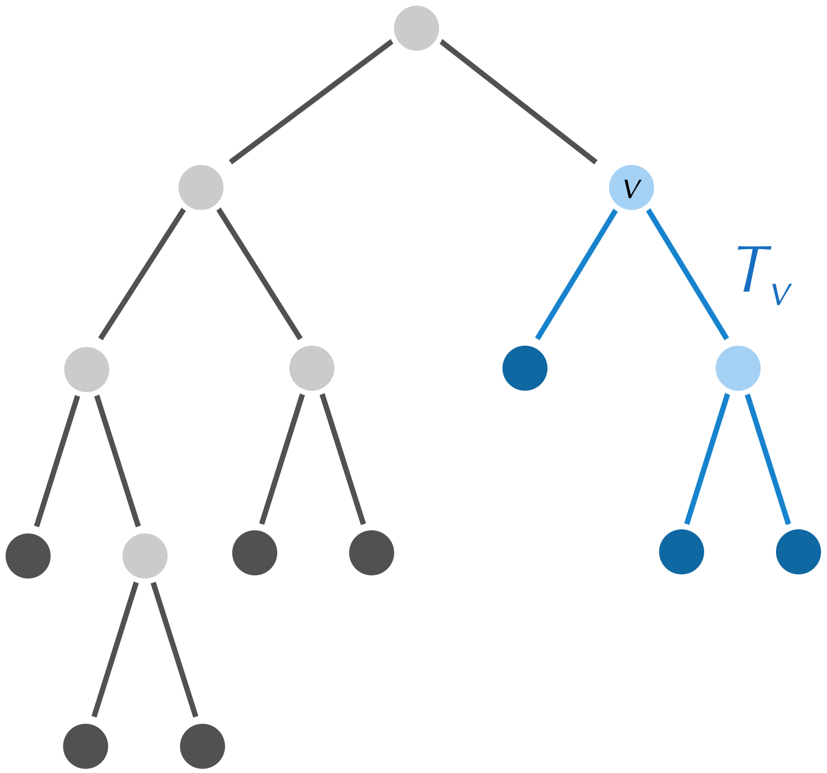
Define $s_k(v)$ as the minimum parsimony score of T_v over all labelings of T_v, assuming that v is labeled by k.

The minimum parsimony score for the tree is equal to the minimum value of $s_k(root)$ over all symbols k.

A Dynamic Programming Algorithm

For symbols *i* and *j*, define

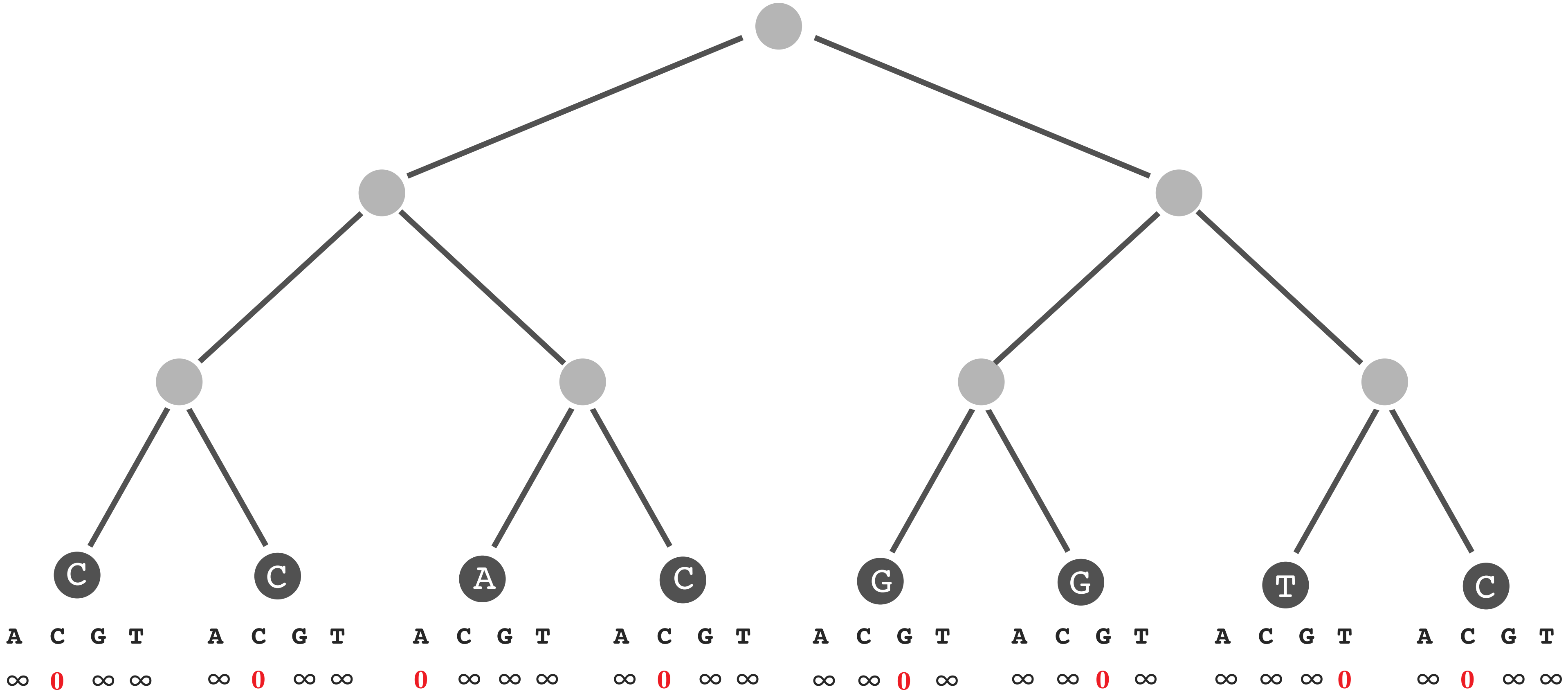
- $\delta_{i,j} = 0$ if $i = j$
- $\delta_{i,j} = 1$ otherwise.



Exercise Break: Prove the following recurrence relation:

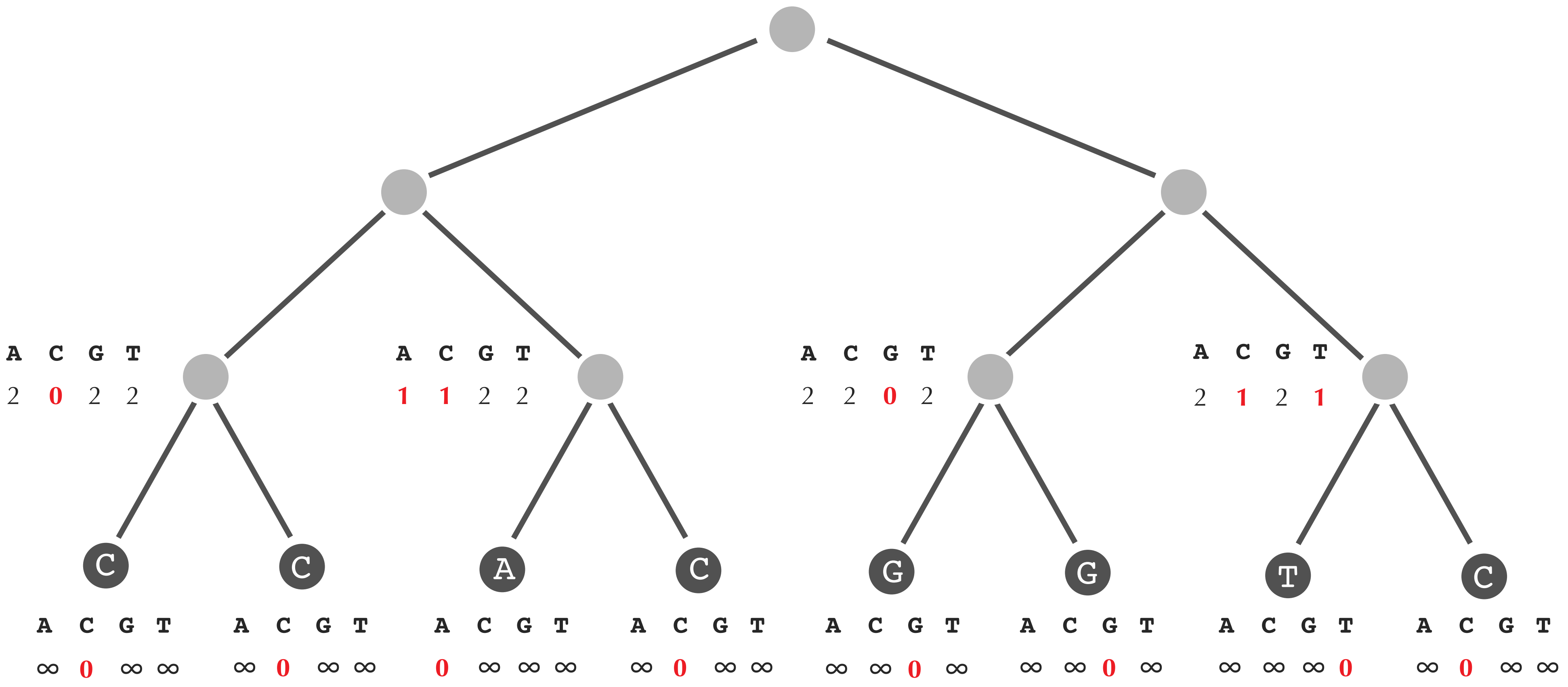
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



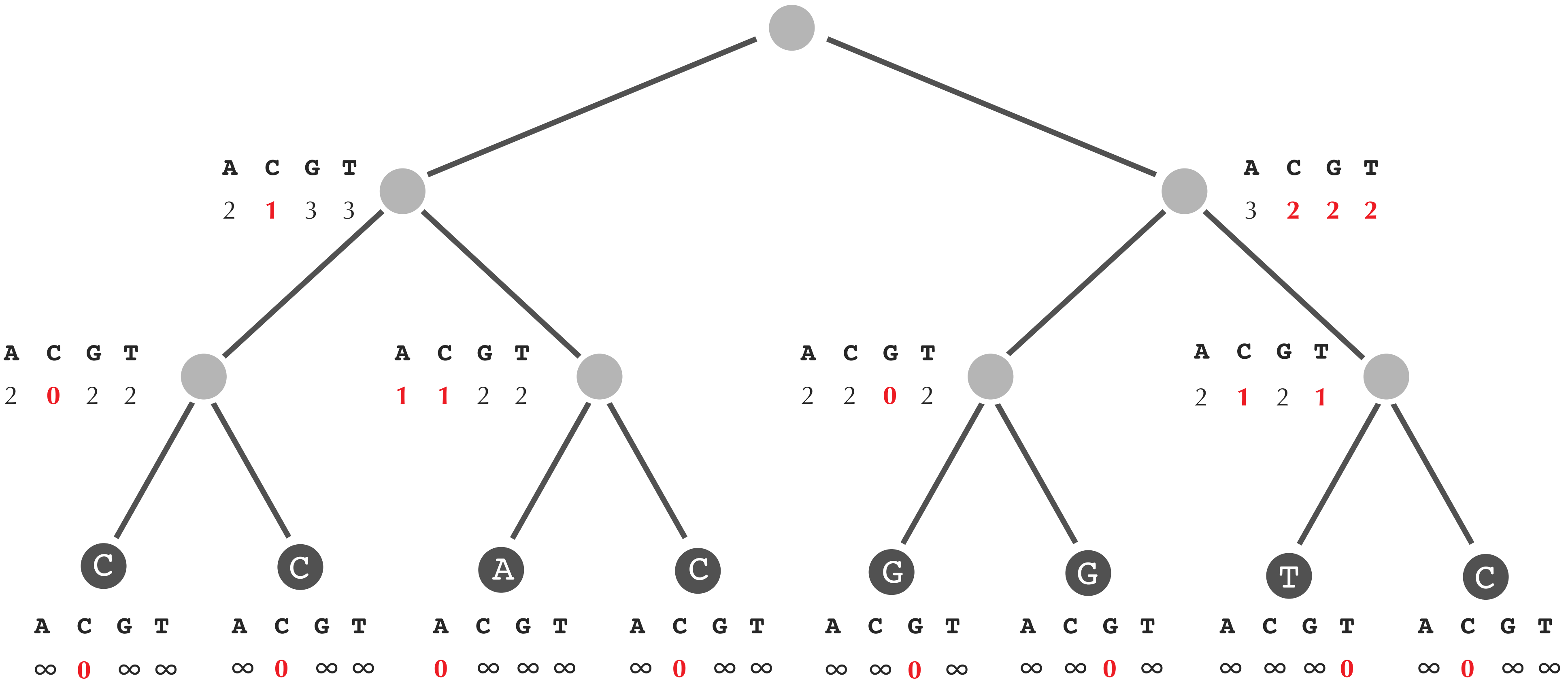
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



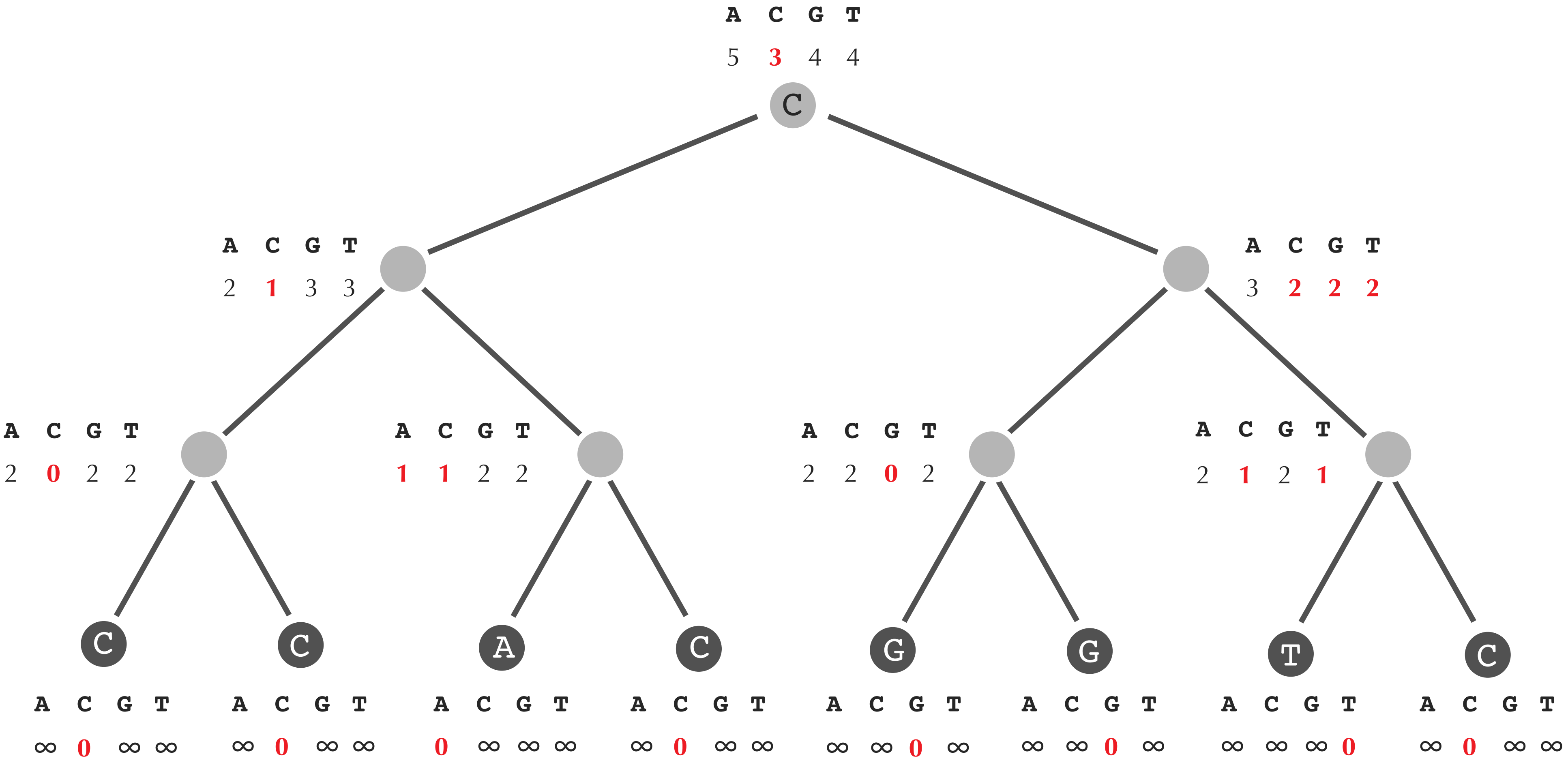
$$s_k(v) = \min_{\text{all symbols } i}\{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i}\{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



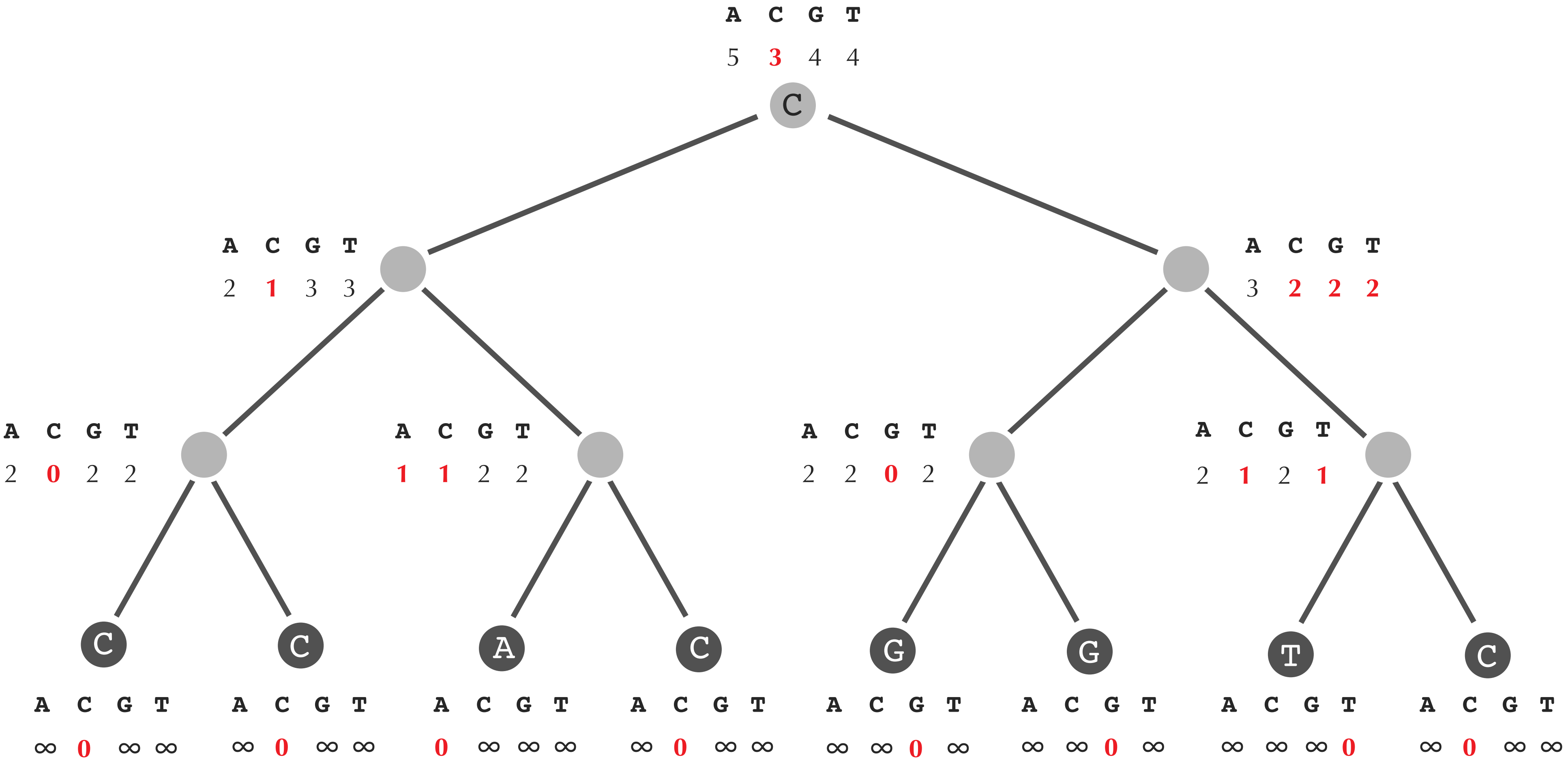
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



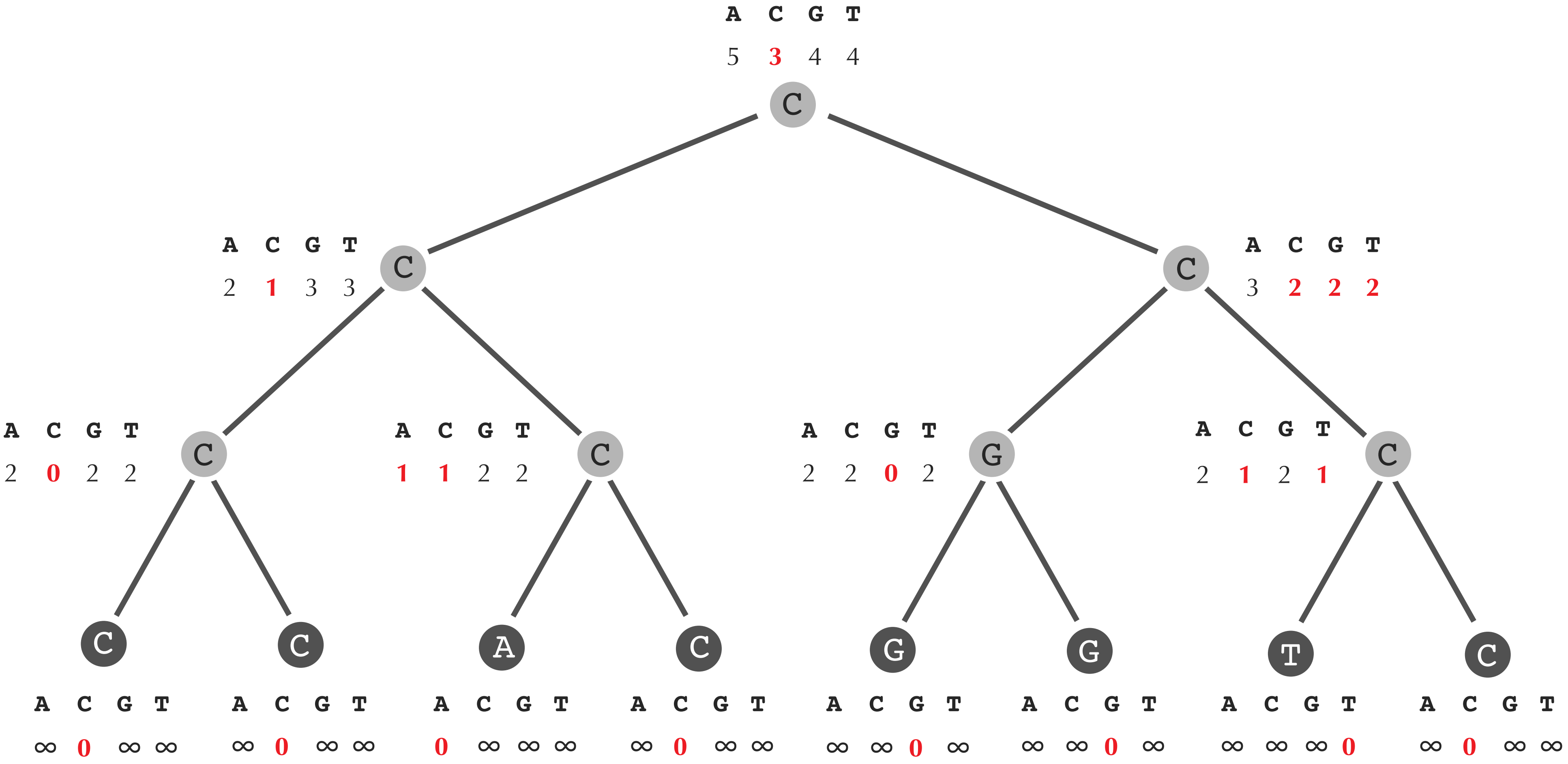
$$s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{j,k}\}$$

A Dynamic Programming Algorithm



Exercise Break: "Backtrack" to fill in the remaining nodes of the tree.

A Dynamic Programming Algorithm



Complexity: if we want to calculate the overall length (cost) of a tree with m species, n characters, and k states, the Parsimony algorithm is of complexity $O(mnk^2)$.

Small Parsimony for Unrooted Trees

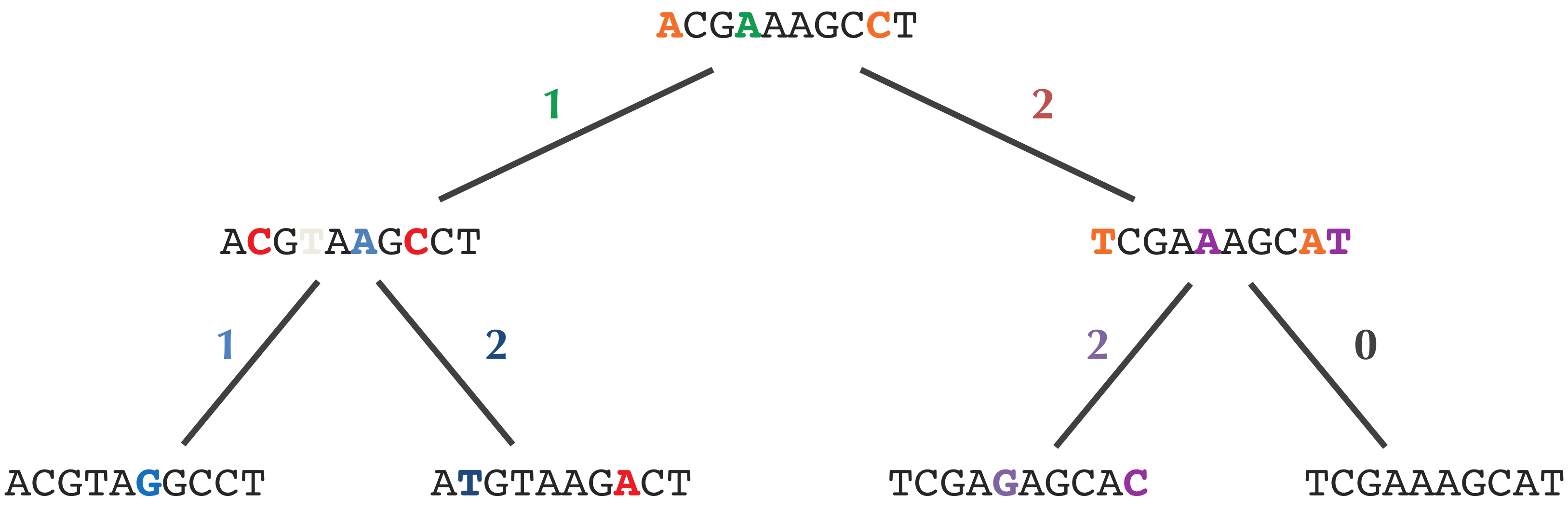
David Sankoff

Small Parsimony in an Unrooted Tree Problem: *Find the most parsimonious labeling of the internal nodes of an unrooted tree.*

- **Input:** An unrooted binary tree with each leaf labeled by a string of length *m*.
- **Output:** A position of the root and a labeling of all other nodes of the tree by strings of length *m* that minimises the tree's parsimony score.

Code Challenge: Solve this problem.

Finding the Most Parsimonious Tree



ACGAAAGCCT

1

2

ACGTAAGCCT

TCGAAAGCAT

1

2

2

0

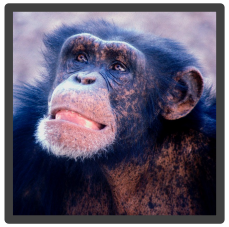
ACGTAGGCCT

ATGTAAGACT

TCGAGAGCAC

TCGAAAGCAT

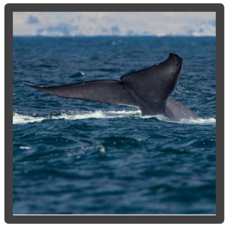
Chimp



Human

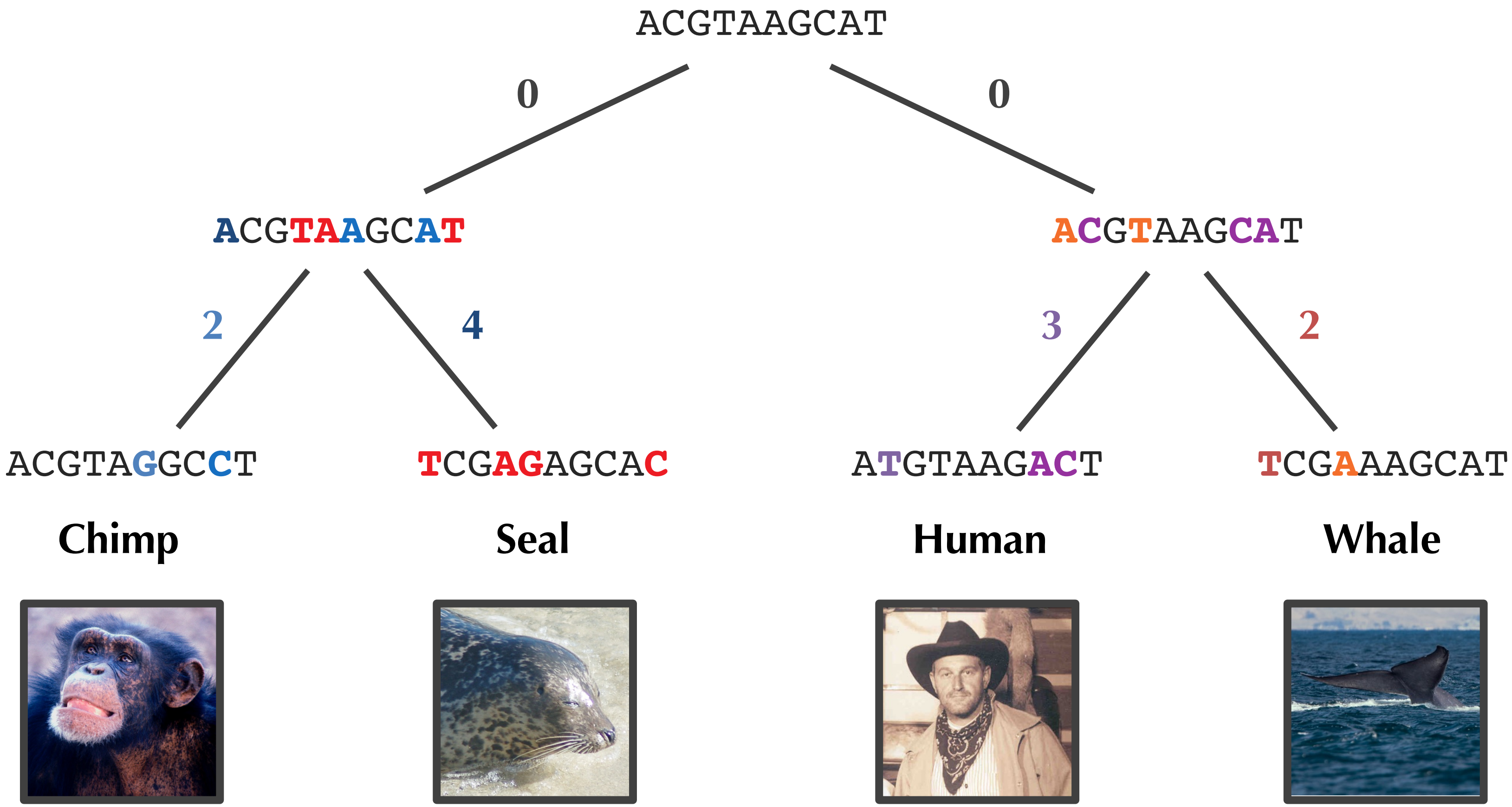
Seal

Whale



Parsimony Score: 8

Finding the Most Parsimonious Tree



ACGTAAGCAT

0

0

ACGTAAGCAT

ACGTAAGCAT

2

4

3

2

ACGTAGGCCT

TCGAGAGCAC

ATGTAAGACT

TCGAAAGCAT

Chimp

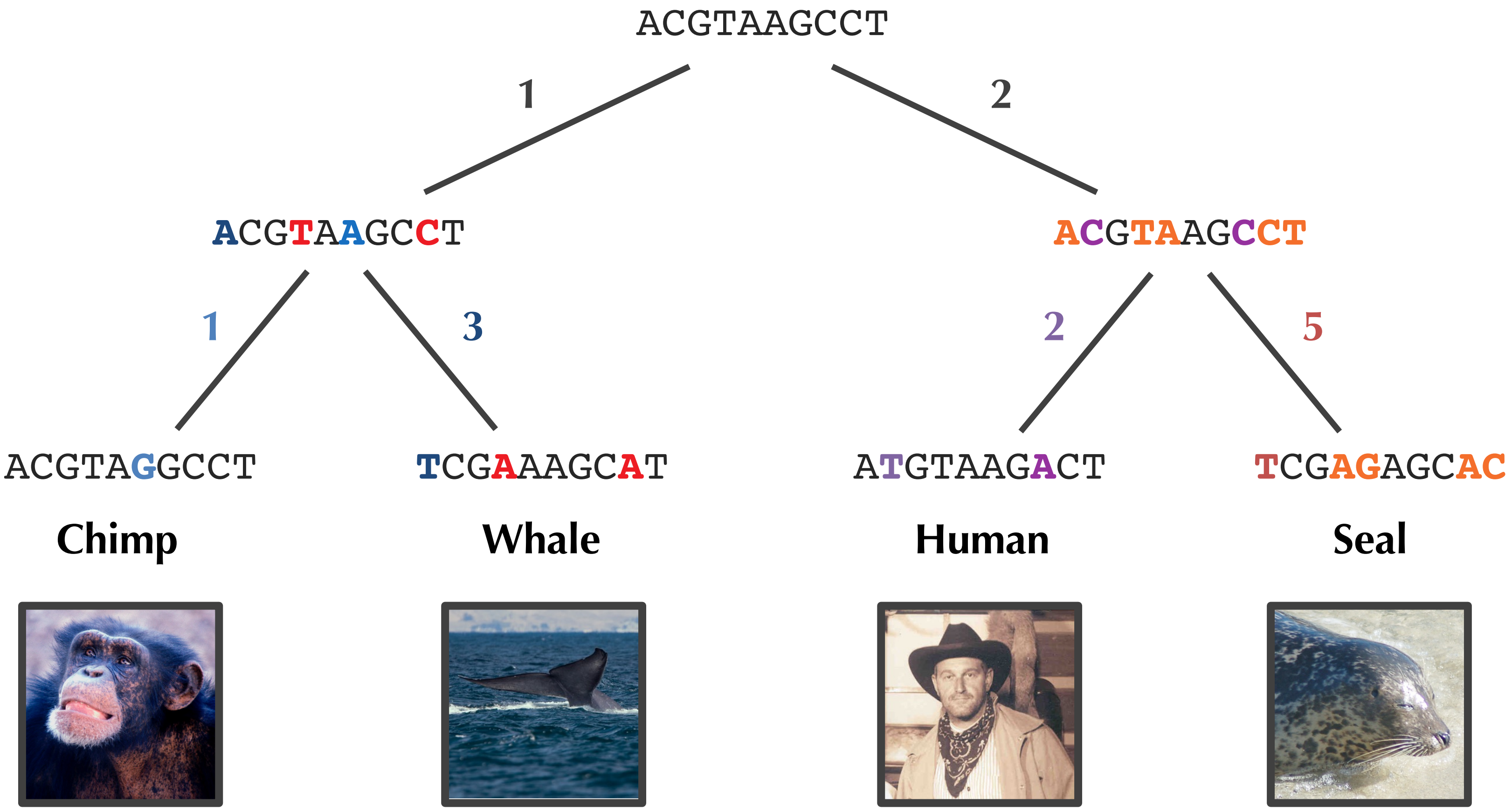
Seal

Human

Whale

Parsimony Score: 11

Finding the Most Parsimonious Tree



Parsimony Score: 14

Finding the Most Parsimonious Tree

Large Parsimony Problem: *Given a set of strings, find a tree* (*with leaves labeled by all these strings*) *having minimum parsimony score.*

- **Input:** A collection of strings of equal length.
- **Output:** A rooted binary tree *T* that minimises the parsimony score among all possible rooted binary trees with leaves labeled by these strings.

Finding the Most Parsimonious Tree

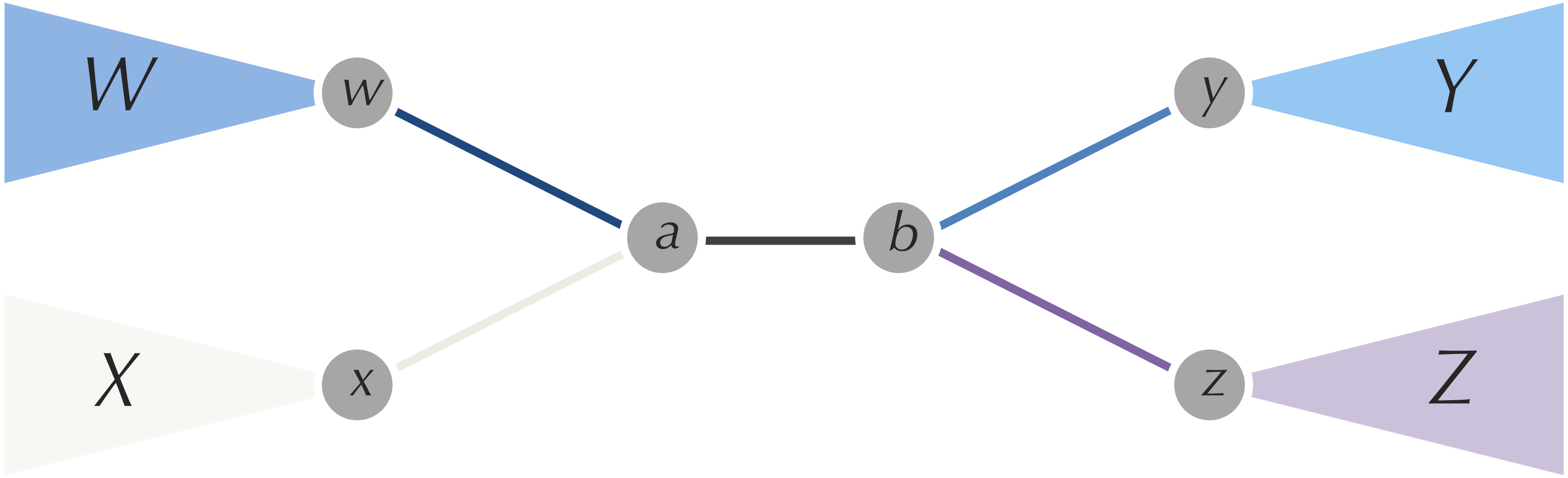
Large Parsimony Problem: *Given a set of strings, find a tree (with leaves labeled by all these strings) having minimum parsimony score.*

- **Input:** A collection of strings of equal length.
- **Output:** A rooted binary tree *T* that minimises the parsimony score among all possible rooted binary trees with leaves labeled by these strings.

Unfortunately, this problem is *NP*-Complete…

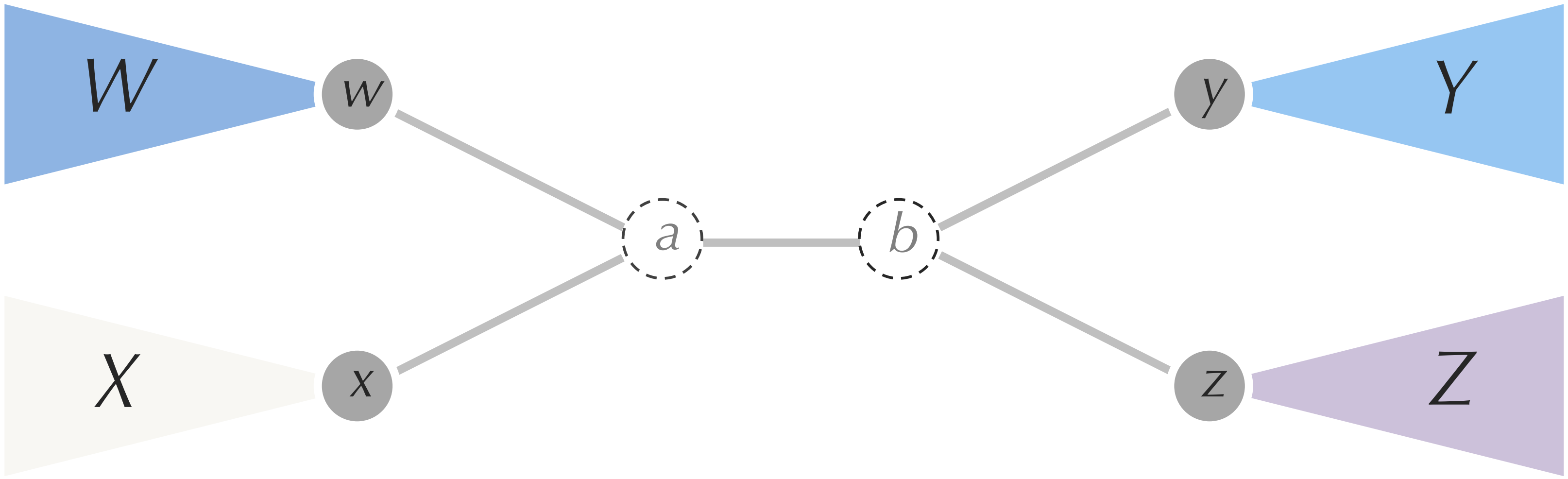
A Greedy Heuristic for Large Parsimony

Note that removing an **internal edge**, an edge connecting two internal nodes (along with the nodes), produces four subtrees (*W*, *X*, *Y*, *Z*).



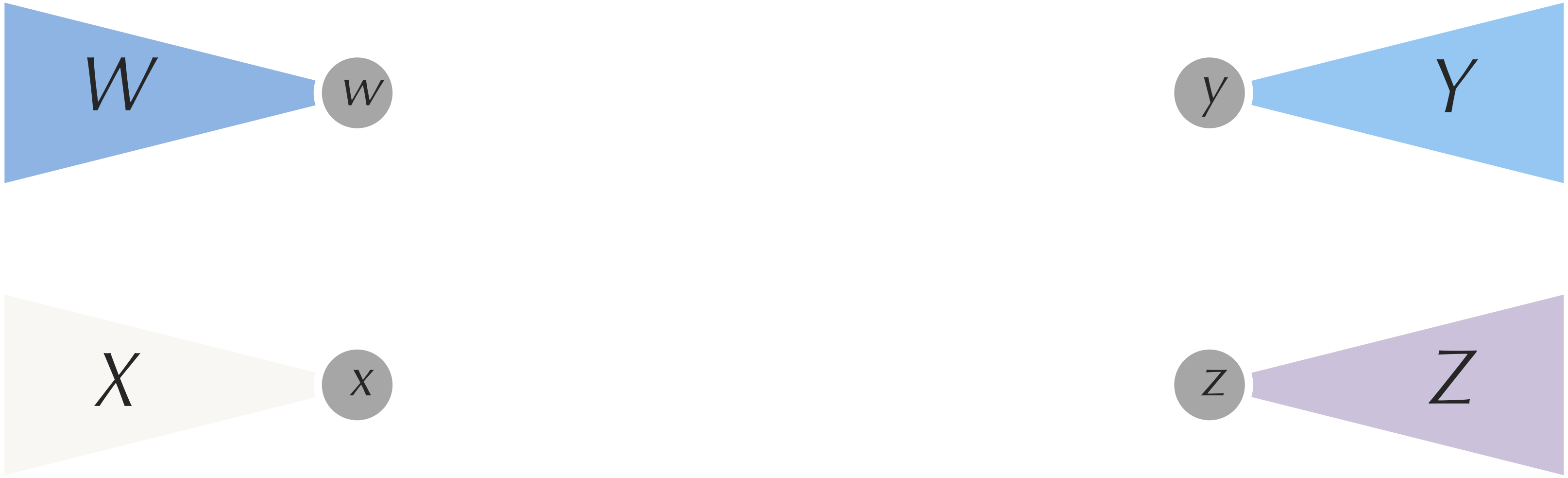
A Greedy Heuristic for Large Parsimony

Note that removing an **internal edge**, an edge connecting two internal nodes (along with the nodes), produces four subtrees (*W*, *X*, *Y*, *Z*).



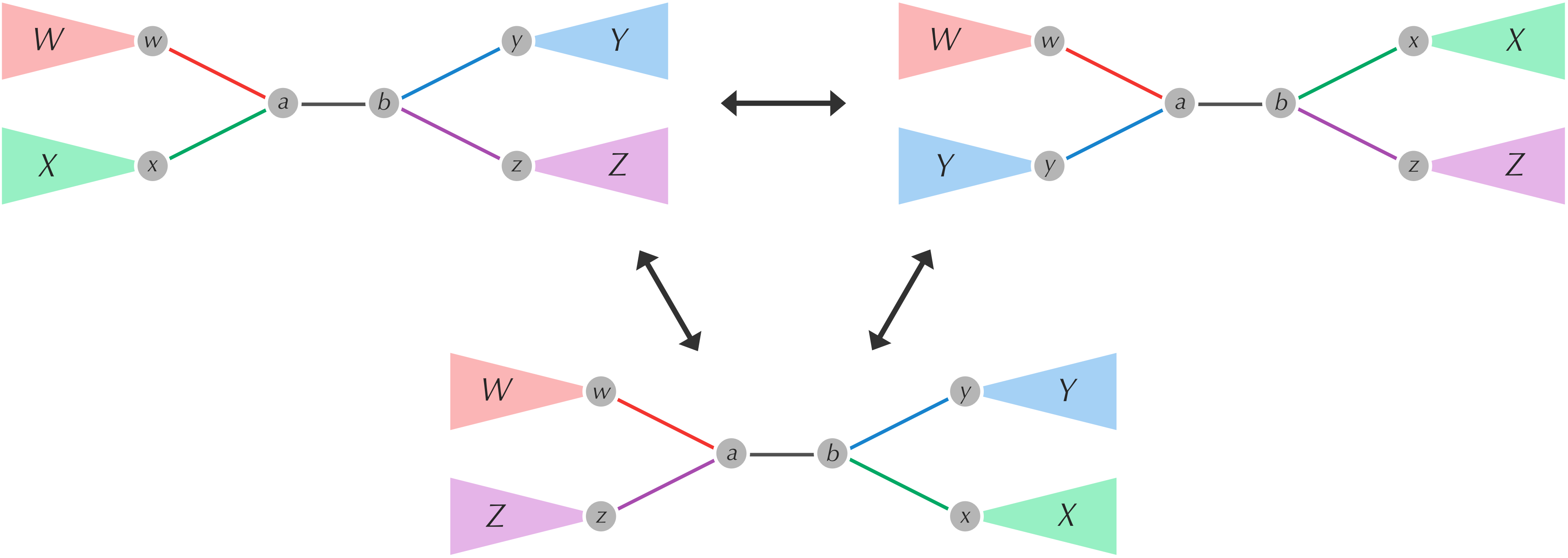
A Greedy Heuristic for Large Parsimony

Note that removing an **internal edge**, an edge connecting two internal nodes (along with the nodes), produces four subtrees (*W*, *X*, *Y*, *Z*).



A Greedy Heuristic for Large Parsimony

Rearranging these subtrees is called a **nearest neighbor interchange.**



A Greedy Heuristic for Large Parsimony

Nearest Neighbors of a Tree Problem: *Given an edge in a binary tree, generate the two neighbors of this tree.*

- **Input:** An internal edge in a binary tree.
- **Output:** The two nearest neighbors of this tree (for the given internal edge).

Code Challenge: Solve this problem.

A Greedy Heuristic for Large Parsimony

Nearest Neighbor Interchange Heuristic:

1. Set current tree equal to arbitrary binary rooted tree structure.
2. Go through all internal edges and perform all possible nearest neighbor interchanges.
3. Solve Small Parsimony Problem on each tree.
4. If any tree has parsimony score improving over optimal tree, set it equal to the current tree. Otherwise, return current tree.

Code Challenge: Implement the nearest-neighbor interchange heuristic.

Tree validation: the bootstrap algorithm

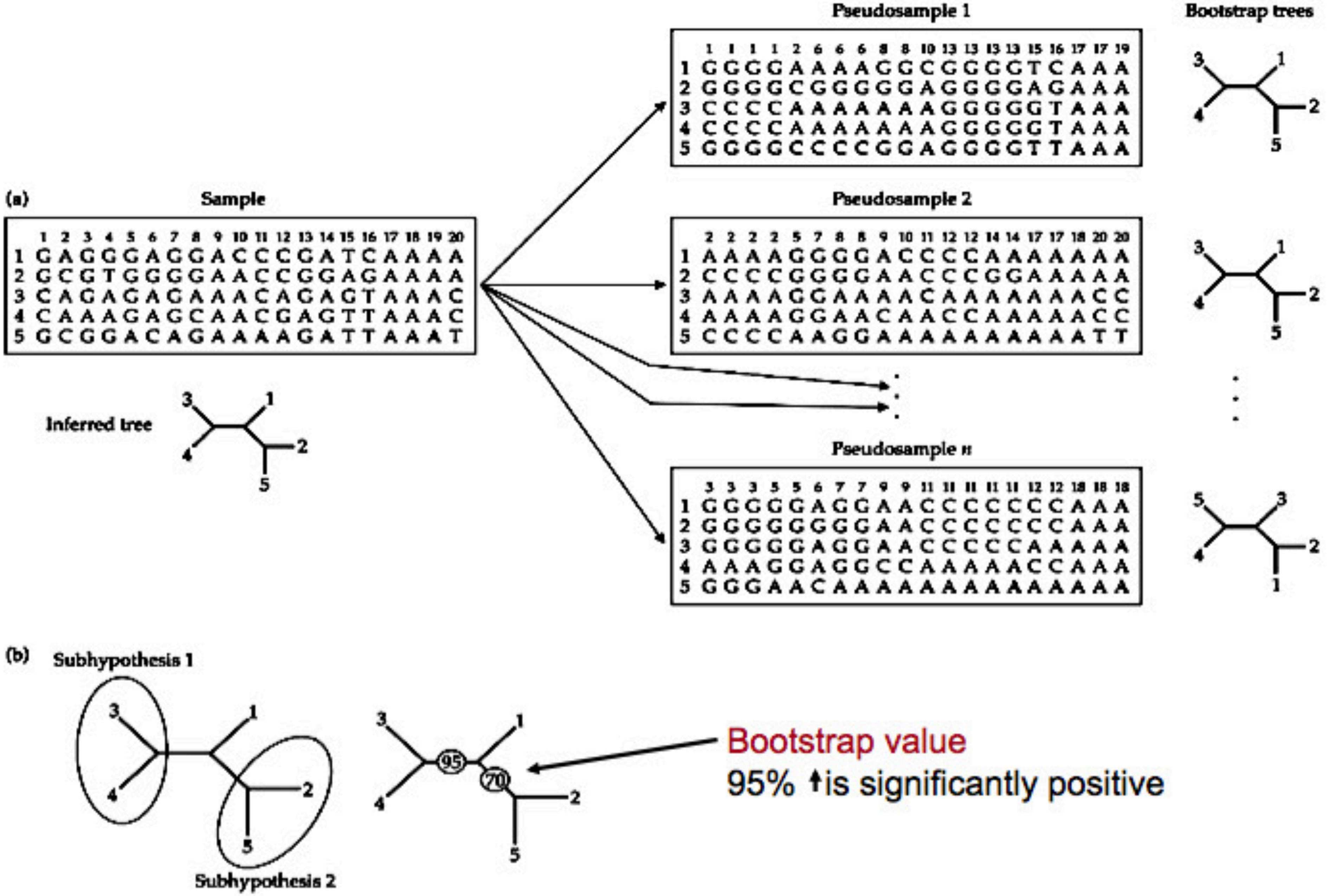
Consider m sequences, each with n nucleotides, a phylogenetic tree is reconstructed using some tree building methods.

1. From each sequence, n nucleotides are randomly chosen with replacements, giving rise to m rows of n columns each. These now constitute a new set of sequences.
2. A tree is then reconstructed with these new sequences using the same tree building method as before.
3. the topology of this tree is compared to that of the original tree. Each interior branch of the original tree that is different from the bootstrap tree is given a score of 0; all other interior branches are given the value 1.

Tree validation: the bootstrap algorithm

1.
2.
3.
4. This procedure of resampling the sites and tree reconstruction is repeated several hundred times, and the percentage of times each interior branch is given a value of 1 is noted.
5. This is known as the bootstrap value. As a general rule, if the bootstrap value for a given interior branch is 95% or higher, then the topology at that branch is considered "correct".

Tree validation: the bootstrap algorithm



Sample

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

Pseudosample 1

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

Pseudosample 2

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

Pseudosample *n*

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

Bootstrap value

95% ↑is significantly positive

Tree validation: the bootstrap algorithm

Consider m sequences, each with n nucleotides, a phylogenetic tree is reconstructed using some tree building methods.

1. From each sequence, n nucleotides are randomly chosen with replacements, giving rise to m rows of n columns each. These now constitute a new set of sequences.
2. A tree is then reconstructed with these new sequences using the same tree building method as before.
3. the topology of this tree is compared to that of the original tree. Each interior branch of the original tree that is different from the bootstrap tree is given a score of 0; all other interior branches are given the value 1.
4. This procedure of resampling the sites and tree reconstruction is repeated several hundred times, and the percentage of times each interior branch is given a value of 1 is noted.
5. This is known as the bootstrap value. As a general rule, if the bootstrap value for a given interior branch is 95% or higher, then the topology at that branch is considered "correct".

Going back to the aligment problem: Generalising Pairwise to Multiple Alignment

- Alignment of 2 sequences is a 2-row matrix.
- Alignment of 3 sequences is a 3-row matrix

```
A T - G C G -
A - C G T - A
A T C A C - A
```

- Our scoring function should score alignments with conserved columns higher.

Alignments = Paths in 3-D

- Alignment of ATGC, AATC, and ATGC

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

#symbols up to a given position

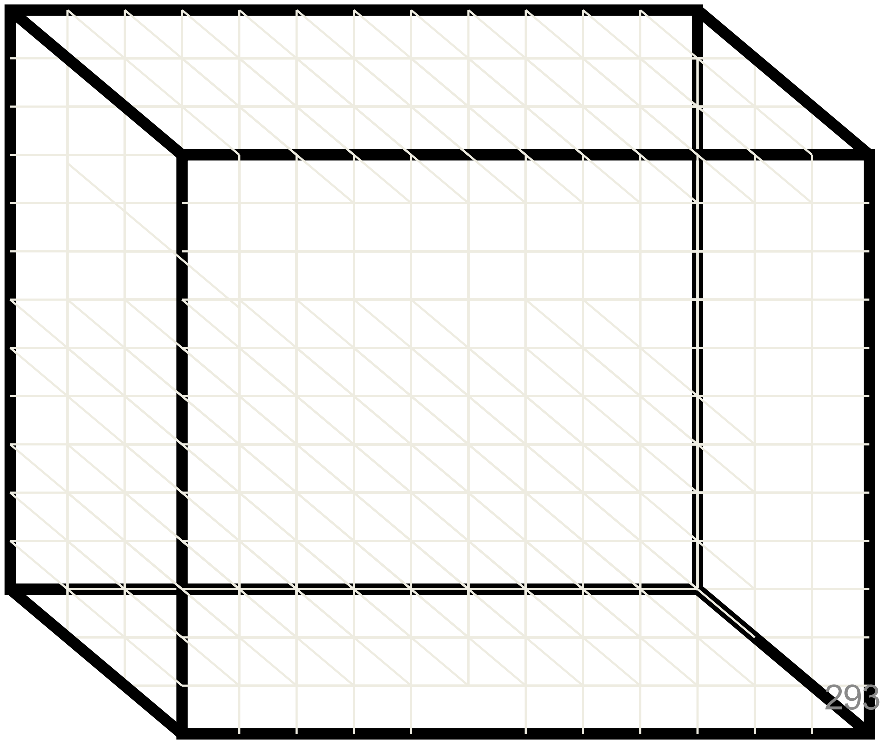
	--	A	T	G	C

Alignments = Paths in 3-D

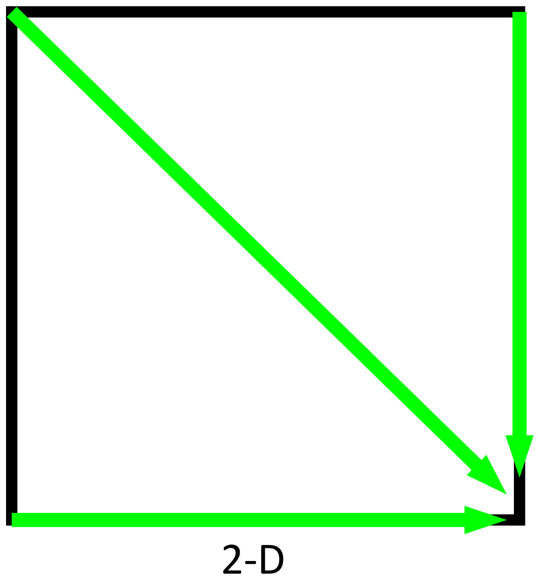
- Alignment of ATGC, AATC, and ATGC

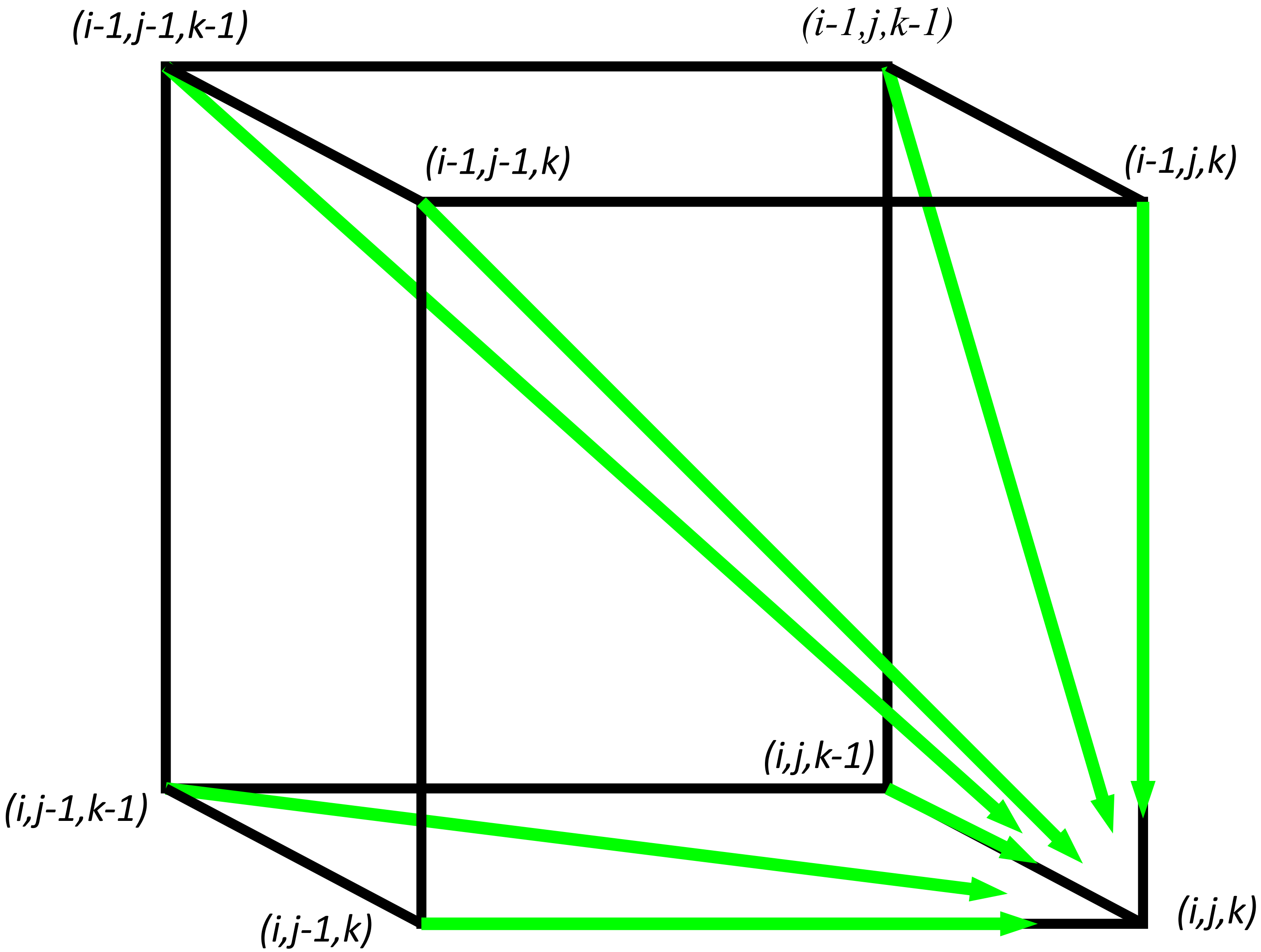
(0,0,0)→(1,1,0)→(1,2,1) →(2,3,2) →(3,3,3) →(4,4,4)

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



2-D Alignment Cell versus 3-D Alignment Cell





Multiple Alignment: Dynamic Programming

$$s_{i,j,k} = \max \begin{cases} s_{i-1,j-1,k-1} + \delta(v_i, w_j, u_k) \\ s_{i-1,j-1,k} + \delta(v_i, w_j, -) \\ s_{i-1,j,k-1} + \delta(v_i, -, u_k) \\ s_{i,j-1,k-1} + \delta(-, w_j, u_k) \\ s_{i-1,j,k} + \delta(v_i, -, -) \\ s_{i,j-1,k} + \delta(-, w_j, -) \\ s_{i,j,k-1} + \delta(-, -, u_k) \end{cases}$$

- $\delta(x, y, z)$ is an entry in the 3-D scoring matrix.

Multiple Alignment: Running Time

- For 3 sequences of length n, the run time is proportional to $7n^3$

- For a k-way alignment, build a k-dimensional Manhattan graph with
  - n^k nodes
  - most nodes have $2^k - 1$ incoming edges.
  - Runtime: $O(2^k n^k)$

Multiple Alignment Induces Pairwise Alignments

Every multiple alignment induces pairwise alignments:

```
AC-GCGG-C
AC-GC-GAG
GCCGC-GAG
```

↓

```
ACGCGG-C   AC-GCGG-C   AC-GCGAG
ACGC-GAC   GCCGC-GAG   GCCGCGAG
```

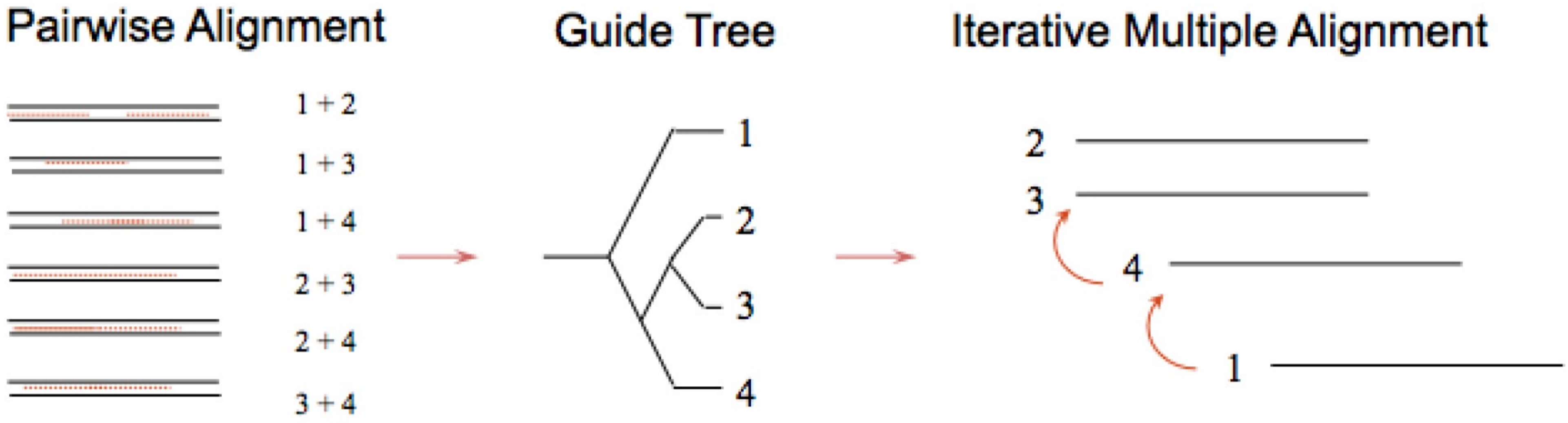
Idea: Construct Multiple from Pairwise Alignments

Given a set of **arbitrary** pairwise alignments, can we construct a multiple alignment that induces them?

```
AAAATTTT----      ----AAAATTTT      TTTTGGGG----
----TTTTGGGG      GGGGAAAA----      ----GGGGAAAA
```

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 the software CLUSTAL (https://www.ebi.ac.uk/Tools/msa/clustalo/)



Progressive Alignment

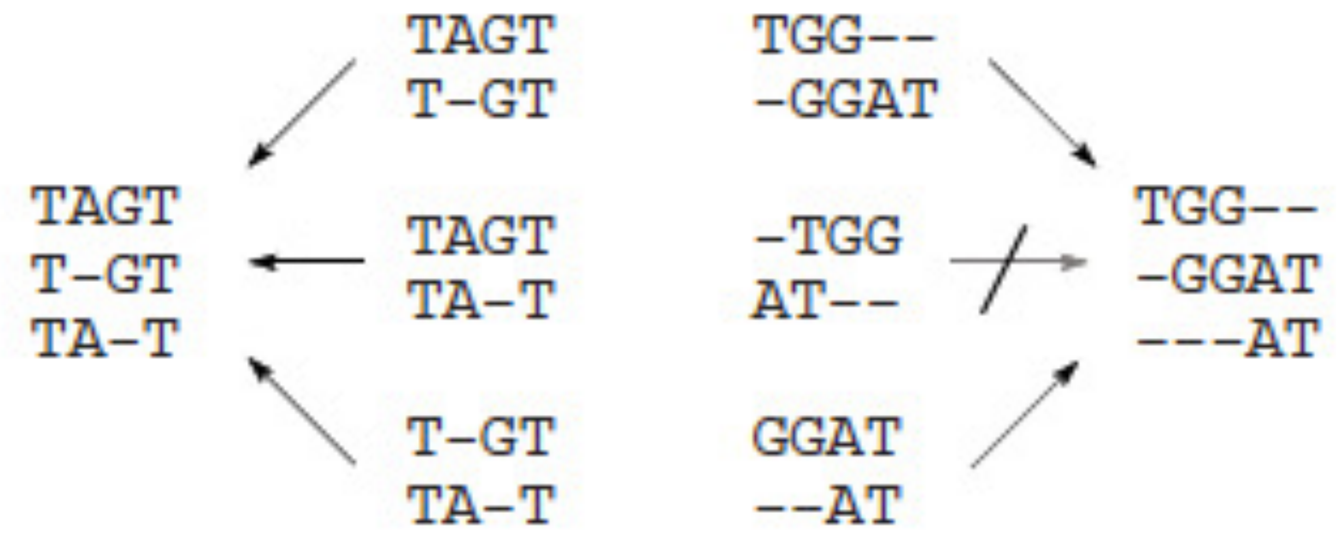
Clustalw:

1. Given N sequences, align each sequence against each other.
2. Use the score of the pairwise alignments to compute a distance matrix.
3. Build a guide tree (tree shows the best order of progressive alignment).
4. Progressive Alignment guided by the tree.

Progressive alignment (Clustal). Input: a set of sequences in Fasta format (also thousands).
Output: alignment of the set of sequences: multi sequence alignment (MSA). Interest: find conserved patterns (across sequences, i.e. columns retaining similar patterns) may indicate functional constraints. In other words, if the same pattern is conserved in multiple sequences from different species, the substring could have an important functional role.

Progressive Alignment

Not all the pairwise alignments build well into a multiple sequence alignment (compare the alignments below and the figure next page)



```
TAGT
T-GT
```

```
TGG--
-GGAT
```

```
TAGT
T-GT
TA-T
```

```
TAGT
TA-T
```

```
-TGG
AT--
```

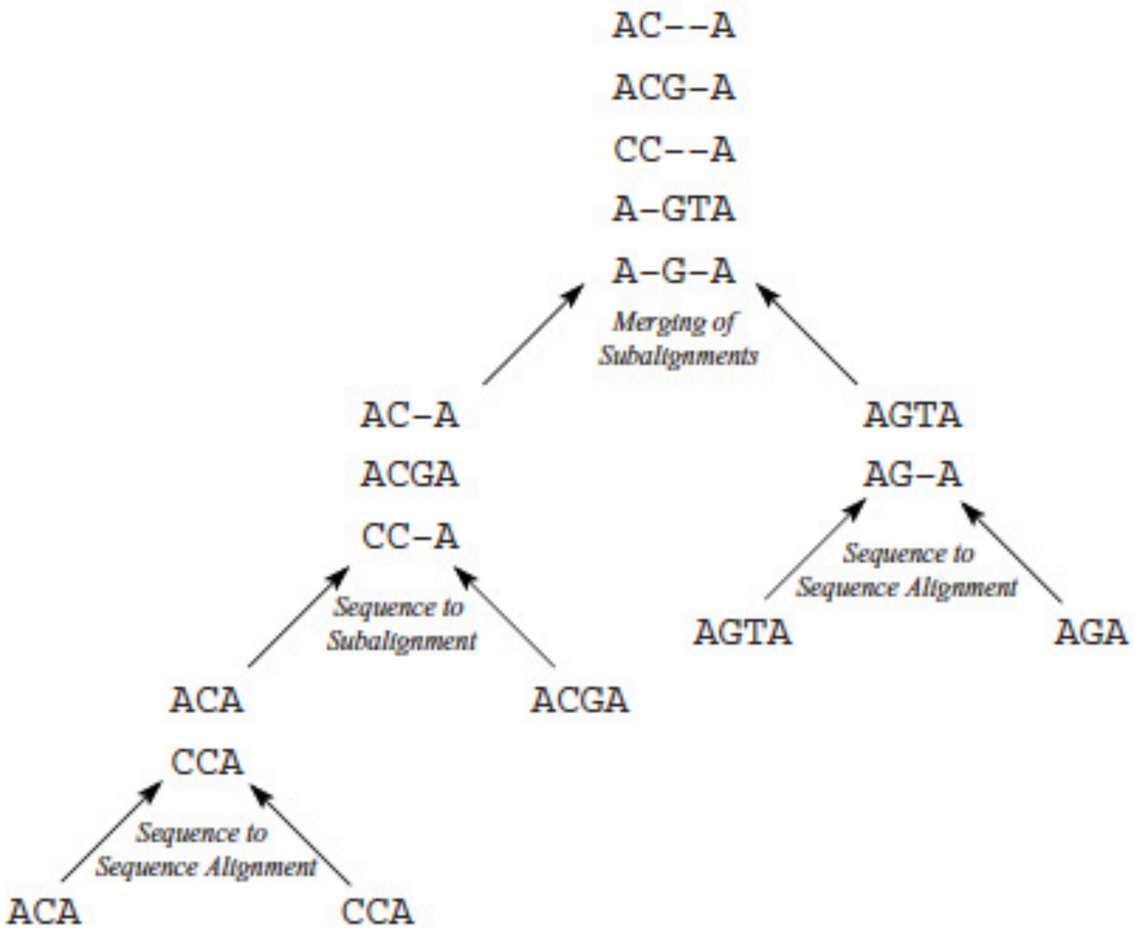
```
TGG--
-GGAT
---AT
```

```
T-GT
TA-T
```

```
GGAT
--AT
```

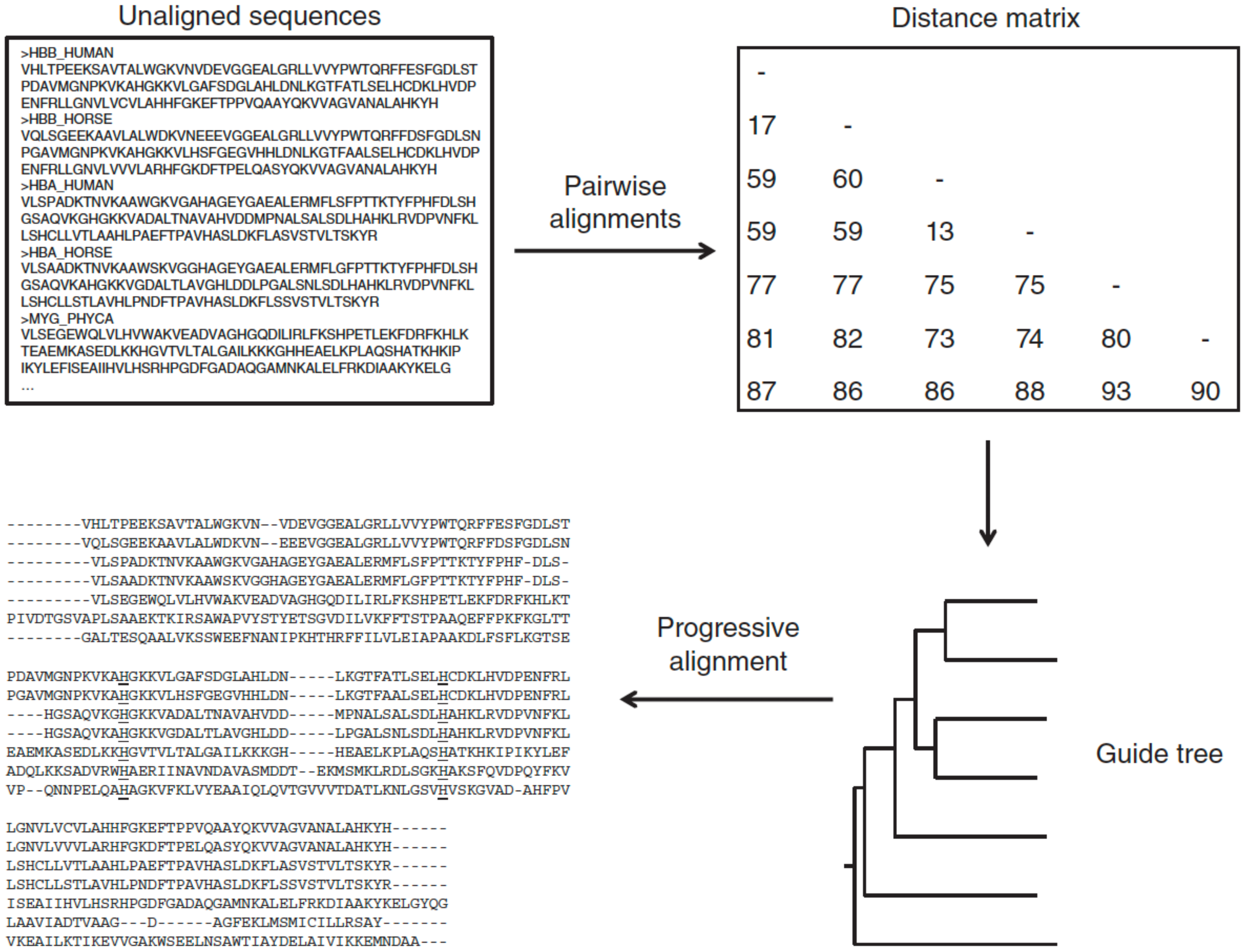
Progressive Alignment

The progressive alignment (see below) builds a final alignment by merging sub-alignments (bottom to top) with a guide tree



The tree allows the ordering the multi alignment

Progressive Alignment



Unaligned sequences

```
>HBB_HUMAN
VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLST
PDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDP
ENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
>HBB_HORSE
VQLSGEEKAAVLALWDKVNEEEVGGEALGRLLVVYPWTQRFFDSFGDLSN
PGAVMGNPKVKAHGKKVLHSFGEGVHHLDNLKGTFAALSELHCDKLHVDP
ENFRLLGNVLVVVLARHFGKDFTPELQASYQKVVAGVANALAHKYH
>HBA_HUMAN
VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSH
GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKL
LSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR
>HBA_HORSE
VLSAADKTNVKAAWSKVGGHAGEYGAEALERMFLGFPTTKTYFPHFDLSH
GSAQVKAHGKKVGDALTLAVGHLDDLPGALSNLSDLHAHKLRVDPVNFKL
LSHCLLSTLAVHLPNDFTPAVHASLDKFLSSVSTVLTSKYR
>MYG_PHYCA
VLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRFKHLK
TEAEMKASEDLKKHGVTVLTALGAILKKKGHHEAELKPLAQSHATKHKIP
IKYLEFISEAIIHVLHSRHPGDFGADAQGAMNKALELFRKDIAAKYKELG
...
```

Pairwise alignments

Distance matrix

-					
17	-				
59	60	-			
59	59	13	-		
77	77	75	75	-	
81	82	73	74	80	-
87	86	86	88	93	90

Guide tree

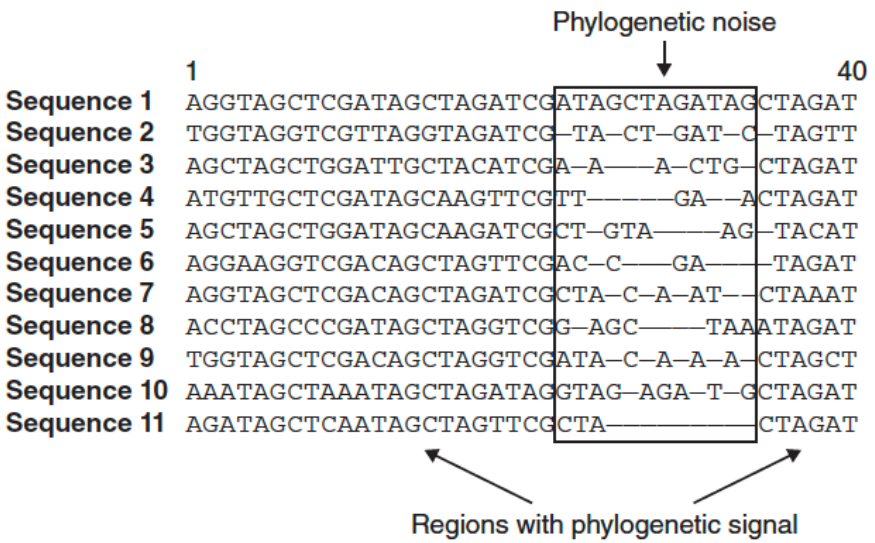
Progressive alignment

```
--------VHLTPEEKSAVTALWGKVN--VDEVGGEALGRLLVVYPWTQRFFESFGDLST
--------VQLSGEEKAAVLALWDKVN--EEEVGGEALGRLLVVYPWTQRFFDSFGDLSN
---------VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-
---------VLSAADKTNVKAAWSKVGGHAGEYGAEALERMFLGFPTTKTYFPHF-DLS-
---------VLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRFKHLKT
PIVDTGSVAPLSAAEKTKIRSAWAPVYSTYETSGVDILVKFFTSTPAAQEFFPKFKGLTT
--------GALTESQAALVKSSWEEFNANIPKHTHRFFILVLEIAPAAKDLFSFLKGTSE

PDAVMGNPKVKAHGKKVLGAFSDGLAHLDN-----LKGTFATLSELHCDKLHVDPENFRL
PGAVMGNPKVKAHGKKVLHSFGEGVHHLDN-----LKGTFAALSELHCDKLHVDPENFRL
----HGSAQVKGHGKKVADALTNAVAHVDD-----MPNALSALSDLHAHKLRVDPVNFKL
----HGSAQVKAHGKKVGDALTLAVGHLDD-----LPGALSNLSDLHAHKLRVDPVNFKL
EAEMKASEDLKKHGVTVLTALGAILKKKGH-----HEAELKPLAQSHATKHKIPIKYLEF
ADQLKKSADVRWHAERIINAVNDAVASMDDT--EKMSMKLRDLSGKHAKSFQVDPQYFKV
VP--QNNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKGVAD-AHFPV

LGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH------
LGNVLVVVLARHFGKDFTPELQASYQKVVAGVANALAHKYH------
LSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR------
LSHCLLSTLAVHLPNDFTPAVHASLDKFLSSVSTVLTSKYR------
ISEAIIHVLHSRHPGDFGADAQGAMNKALELFRKDIAAKYKELGYQG
LAAVIADTVAAG---D------AGFEKLMSMICILLRSAY-------
VKEAILKTIKEVVGAKWSEELNSAWTIAYDELAIVIKKEMNDAA---
```

Software: muscle, MAFFT

Challenges

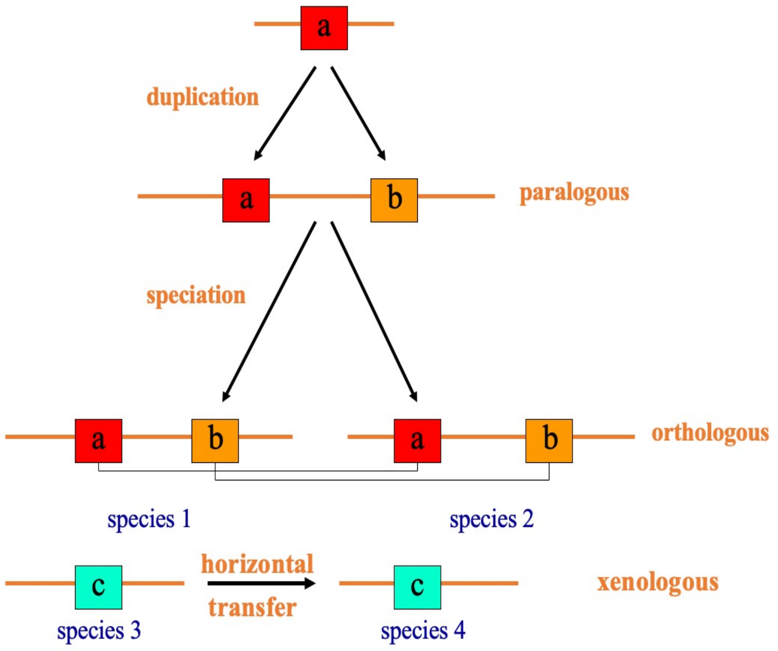
Non examinable



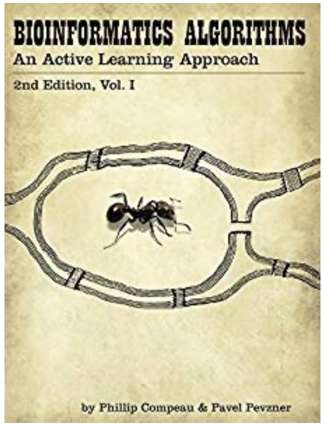
```
           1                                      40
Sequence 1  AGGTAGCTCGATAGCTAGATCGATAGCTAGATAGCTAGAT
Sequence 2  TGGTAGGTCGTTAGGTAGATCG-TA-CT-GAT-C-TAGTT
Sequence 3  AGCTAGCTGGATTGCTACATCGA-A---A-CTG-CTAGAT
Sequence 4  ATGTTGCTCGATAGCAAGTTCGTT-----GA--ACTAGAT
Sequence 5  AGCTAGCTGGATAGCAAGATCGCT-GTA----AG-TACAT
Sequence 6  AGGAAGGTCGACAGCTAGTTCGAC-C---GA----TAGAT
Sequence 7  AGGTAGCTCGACAGCTAGATCGCTA-C-A-AT--CTAAAT
Sequence 8  ACCTAGCCCGATAGCTAGGTCGG-AGC----TAAATAGAT
Sequence 9  TGGTAGCTCGACAGCTAGGTCGATA-C-A-A-A-CTAGCT
Sequence 10 AAATAGCTAAATAGCTAGATAGGTAG-AGA-T-GCTAGAT
Sequence 11 AGATAGCTCAATAGCTAGTTCGCTA---------CTAGAT
```

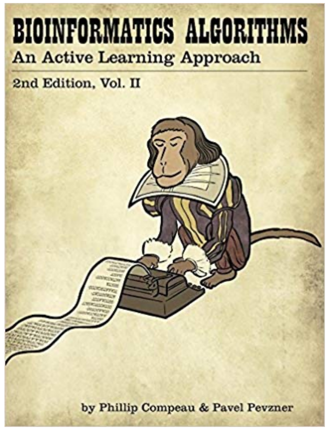
Very little information

Identification of paralogous, orthologous, xenologous events



Reference for this section


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. I
by Phillip Compeau & Pavel Pevzner


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. II
by Phillip Compeau & Pavel Pevzner

- Chapter 7 Vol 2 (pag 3-62)

Reference: D.G. Higgins, J.D. Thompson, and T.J. Gibson. Using CLUSTAL for multiple sequence alignments. Methods in Enzymology, 266:383402, 1996.

Section 4

Clustering biological data

- The Lloyd algorithm for k-means clustering
- From Hard to Soft Clustering
- From Coin Flipping to k-means Clustering
- Expectation Maximisation
- Soft k-means Clustering
- Hierarchical Clustering
- Markov Clustering Algorithm
- Leiden and Lovain

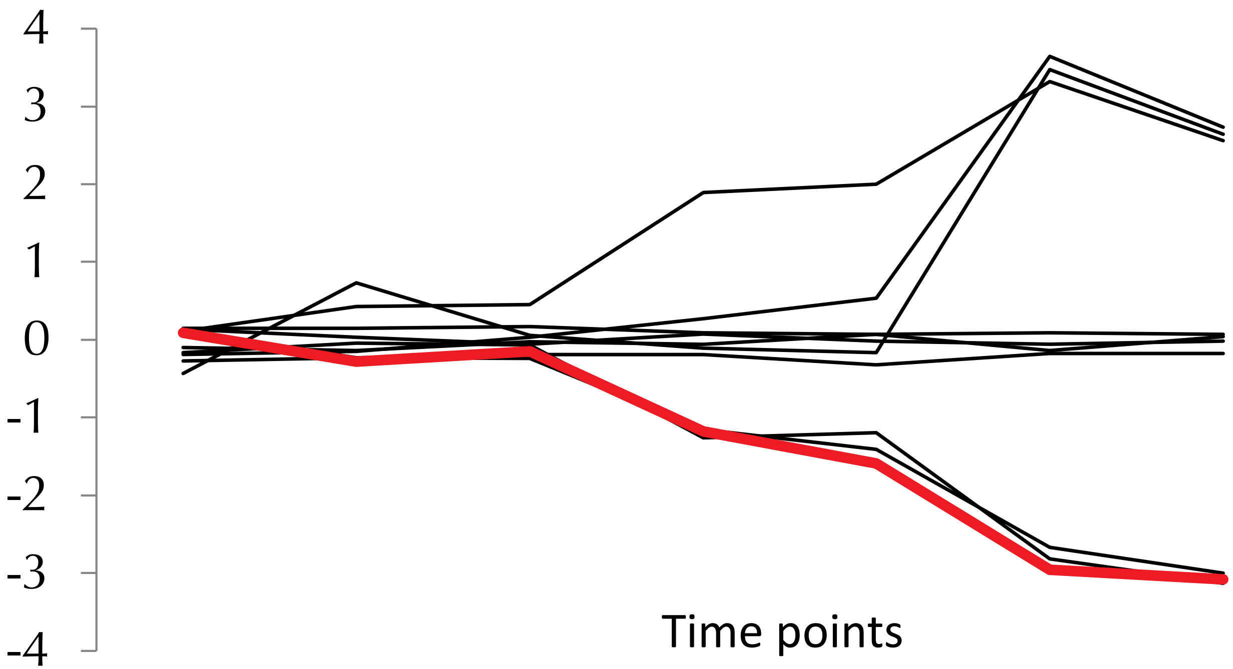
Biologists need algorithms to find similar behaving genes

	Time points						
YLR361C	0.14	0.03	-0.06	0.07	-0.01	-0.06	-0.01
YMR290C	0.12	-0.23	-0.24	-1.16	-1.40	-2.67	-3.00
YNR065C	-0.10	-0.14	-0.03	-0.06	-0.07	-0.14	-0.04
YGR043C	-0.43	-0.73	-0.06	-0.11	-0.16	3.47	2.64
YLR258W	0.11	0.43	0.45	1.89	2.00	3.32	2.56
YPL012W	0.09	-0.28	-0.15	-1.18	-1.59	-2.96	-3.08
YNL141W	-0.16	-0.04	-0.07	-1.26	-1.20	-2.82	-3.13
YJL028W	-0.28	-0.23	-0.19	-0.19	-0.32	-0.18	-0.18
YKL026C	-0.19	-0.15	0.03	0.27	0.54	3.64	2.74
YPR055W	0.15	0.15	0.17	0.09	0.07	0.09	0.07

e_{ij} **= expression level** of gene *i* at checkpoint *j*

gene expression vector (log[expression])

Genes (yeast genome)



Biologists need algorithms to find similar behaving genes

Genes as Points in Multidimensional Space

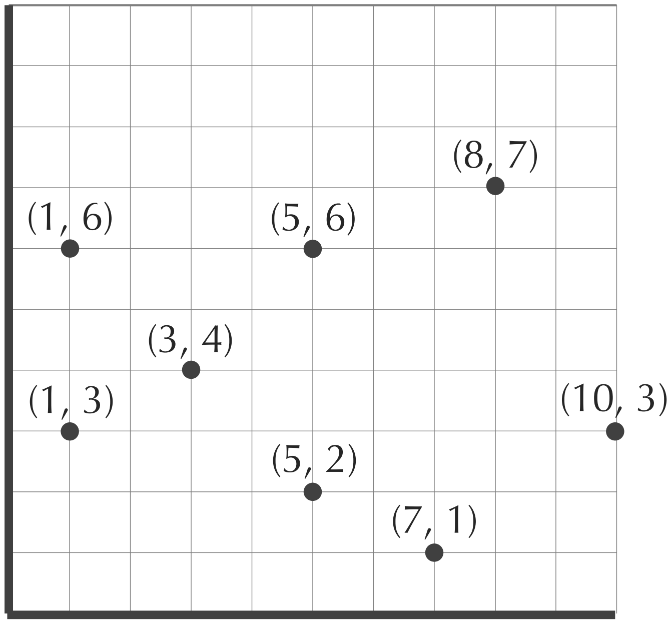
YLR361C	0.14	0.03	-0.06	0.07	-0.01	-0.06	-0.01
YMR290C	0.12	-0.23	-0.24	-1.16	-1.40	-2.67	-3.00
YNR065C	-0.10	-0.14	-0.03	-0.06	-0.07	-0.14	-0.04
YGR043C	-0.43	-0.73	-0.06	-0.11	-0.16	3.47	2.64
YLR258W	0.11	0.43	0.45	1.89	2.00	3.32	2.56
YPL012W	0.09	-0.28	-0.15	-1.18	-1.59	-2.96	-3.08
YNL141W	-0.16	-0.04	-0.07	-1.26	-1.20	-2.82	-3.13
YJL028W	-0.28	-0.23	-0.19	-0.19	-0.32	-0.18	-0.18
YKL026C	-0.19	-0.15	0.03	0.27	0.54	3.64	2.74
YPR055W	0.15	0.15	0.17	0.09	0.07	0.09	0.07

n x *m*

gene expression matrix

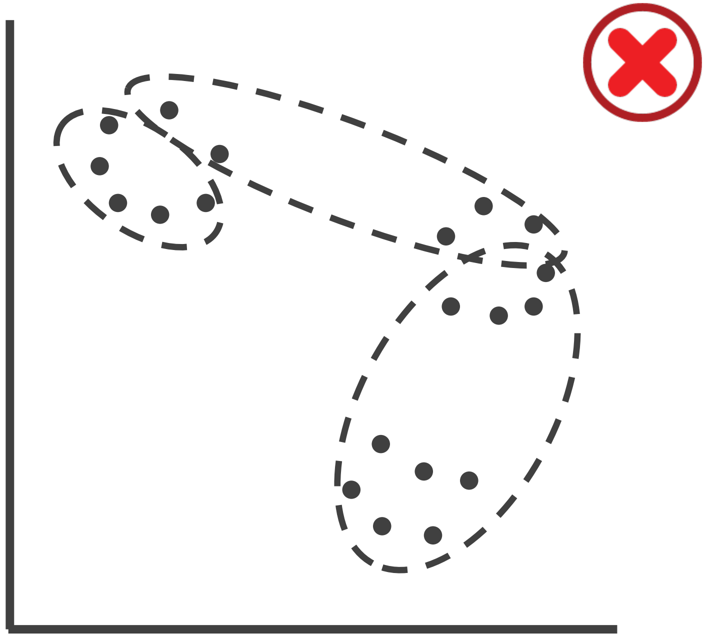
↓

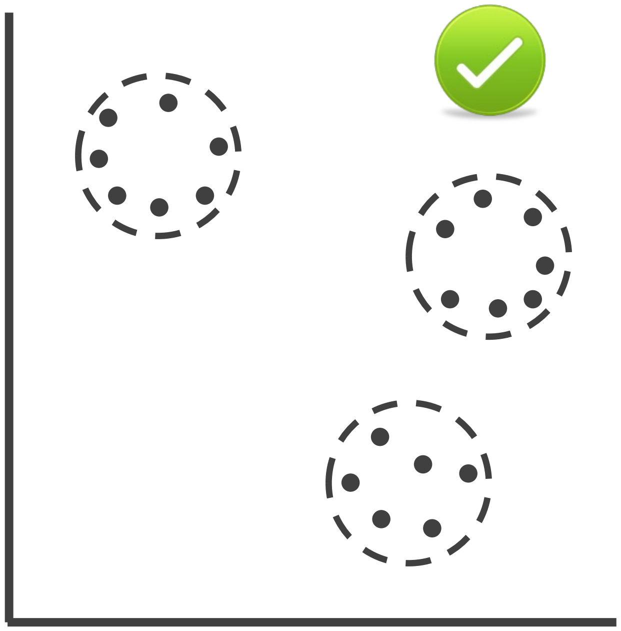
n points in *m*-dimensional space



Good Clustering Principle: Elements within the same cluster are closer to each other than elements in different clusters.

- distance between elements in the same cluster < Δ
- distance between elements in different clusters > Δ

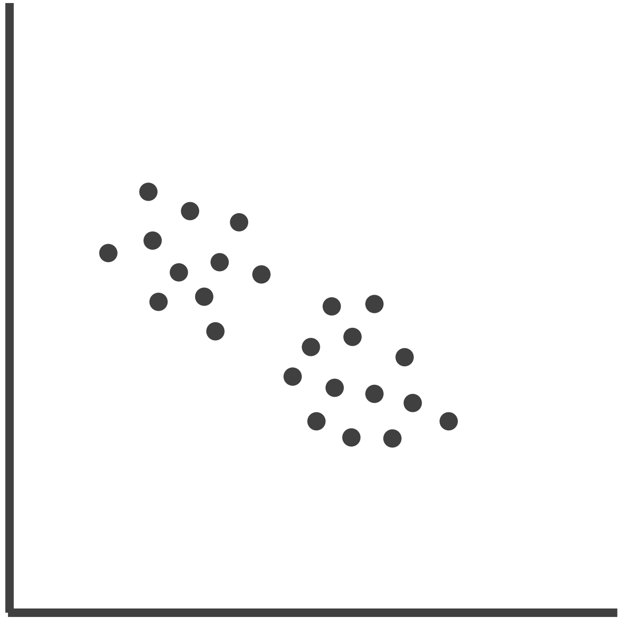




Clustering Problem

Clustering Problem: *Partition a set of expression vectors into clusters.*

- **Input**: A collection of *n* vectors and an integer *k*.
- **Output**: Partition of *n* vectors into *k* disjoint clusters satisfying the Good Clustering Principle.

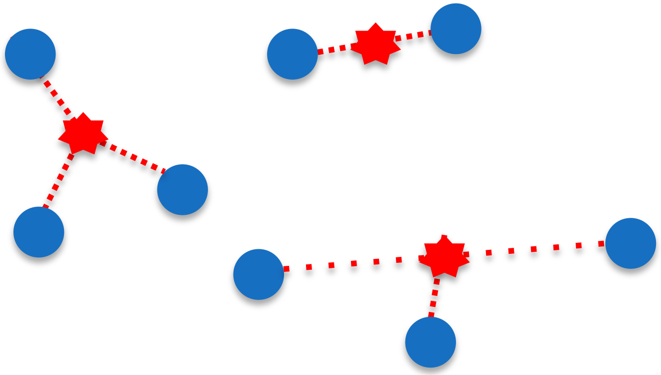


Any partition into two clusters **does not** satisfy the Good Clustering Principle!

Clustering as Finding Centers

Goal: partition a set *Data* into *k* clusters.

Equivalent goal: find a set of *k* points *Centers* that will serve as the "centers" of the *k* clusters in *Data*.

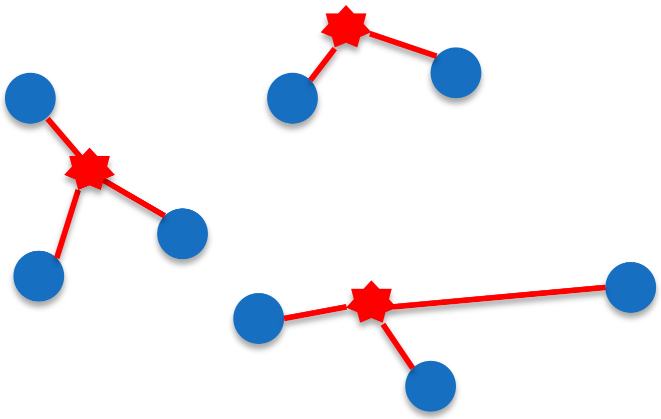


Clustering as Finding Centers

Goal: partition a set *Data* into *k* clusters.

Equivalent goal: find a set of *k* points *Centers* that will serve as the "centers" of the *k* clusters in *Data* and will minimise some notion of distance from *Centers* to *Data* .

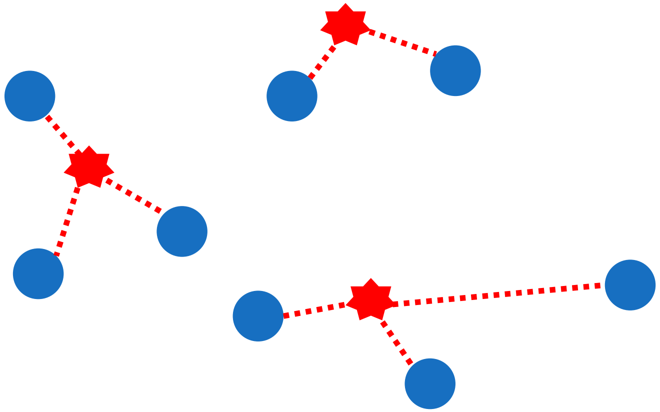
What is the "distance" from *Centers* to *Data*?



Distance from a *Single* *DataPoint* to *Centers*

The distance from *DataPoint* in *Data* to *Centers* is the distance from *DataPoint* to the closest center:

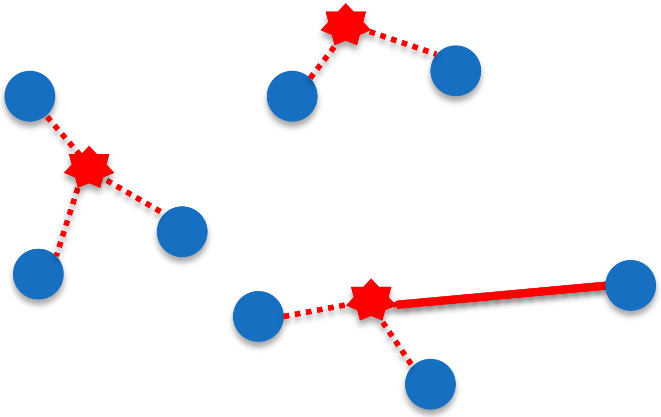
$$d(\mathit{DataPoint}, \mathit{Centers}) = \min_{\text{all points } x \text{ from } \mathit{Centers}} d(\mathit{DataPoint}, x)$$



Distance from *Data* to *Centers*

$$MaxDistance(Data, Centers) =$$

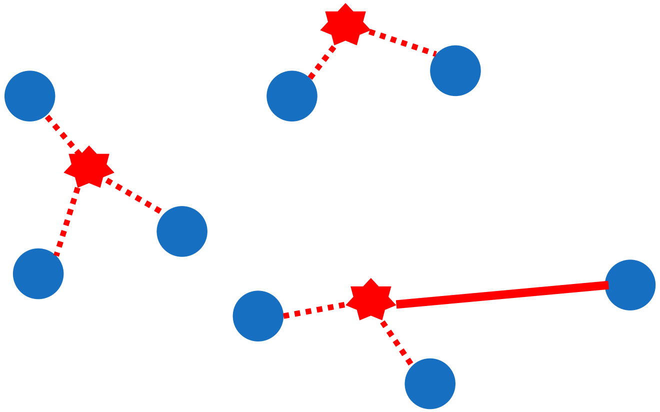
$$\max_{\text{all points } DataPoint \text{ from } Data} d(DataPoint, Centers)$$



k-Center Clustering Problem

***k*-Center Clustering Problem.** Given a set of points *Data*, find *k* centers minimising *MaxDistance*(*Data*, *Centers*).

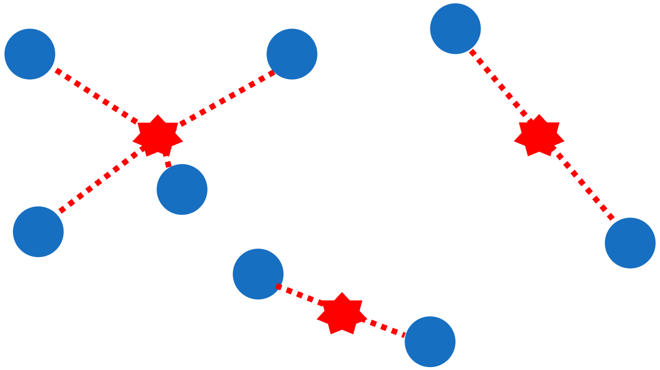
- **Input:** A set of points *Data* and an integer *k*.
- **Output:** A set of *k* points *Centers* that minimises *MaxDistance*(*DataPoints*, *Centers*) over all possible choices of *Centers*.



k-Center Clustering Problem

***k*-Center Clustering Problem.** Given a set of points *Data*, find *k* centers minimising *MaxDistance*(*Data*, *Centers*).

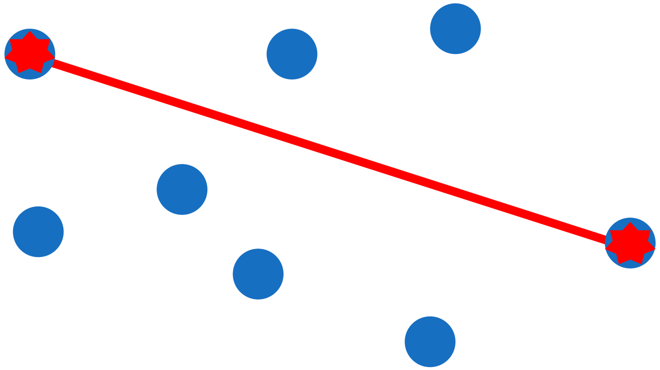
- **Input:** A set of points *Data* and an integer *k*.
- **Output:** A set of *k* points *Centers* that minimises *MaxDistance*(*DataPoints*, *Centers*) over all possible choices of *Centers*.



An even better set of centers!

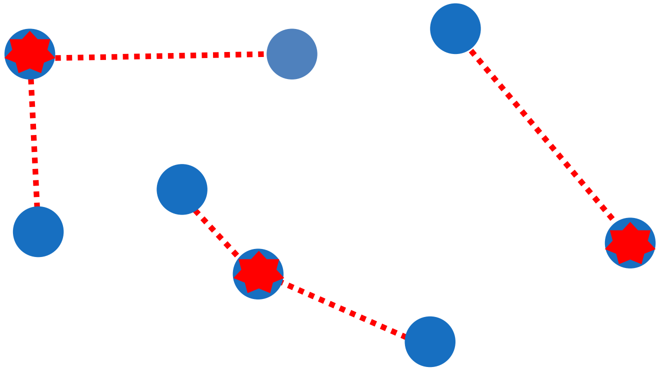
k-Center Clustering Heuristic

```
**FarthestFirstTraversal**(*Data*, *k*)
    *Centers* ← the set consisting of a single *DataPoint* from *Data*
    **while** *Centers* have fewer than *k* points
        *DataPoint* ← a point in *Data* maximising *d*(*DataPoint*, *Centers*)
                              among all data points
        add *DataPoint* to *Centers*
```



k-Center Clustering Heuristic

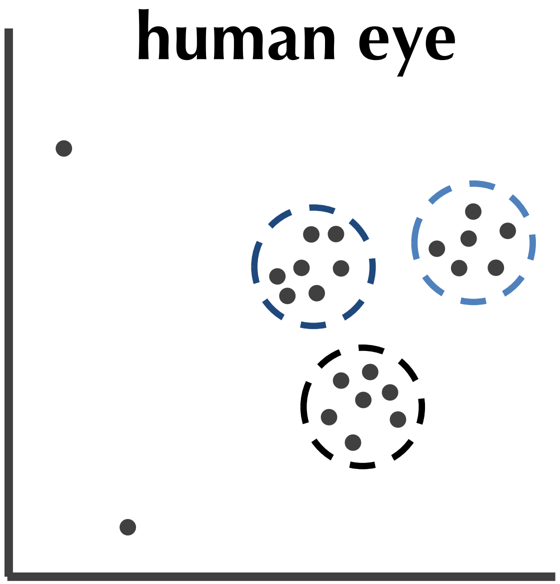
```
**FarthestFirstTraversal**(*Data*, *k*)
    *Centers* ← the set consisting of a single *DataPoint* from *Data*
    while *Centers* have fewer than *k* points
        *DataPoint* ← a point in *Data* maximising *d*(*DataPoint*, *Centers*)
                                among all data points
        add *DataPoint* to *Centers*
```

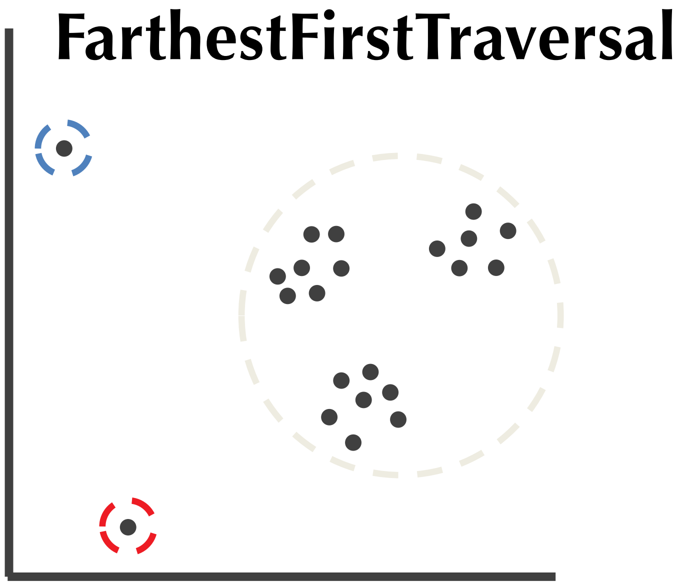


What Is Wrong with **FarthestFirstTraversal**?

FarthestFirstTraversal selects *Centers* that minimise *MaxDistance*(*Data*, *Centers*).

But biologists are interested in **typical** rather than **maximum** deviations, since maximum deviations may represent **outliers** (experimental errors).


human eye


FarthestFirstTraversal

Modifying the Objective Function

The **maximal distance** between *Data* and *Centers*:

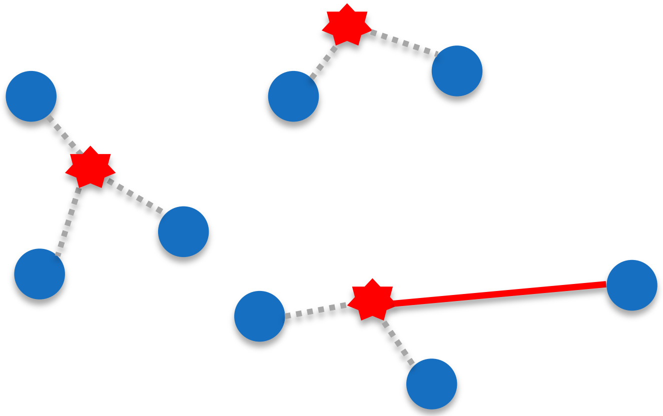
$$MaxDistance(Data, Centers) = \max_{DataPoint\ \text{from}\ Data} d(DataPoint, Centers)$$

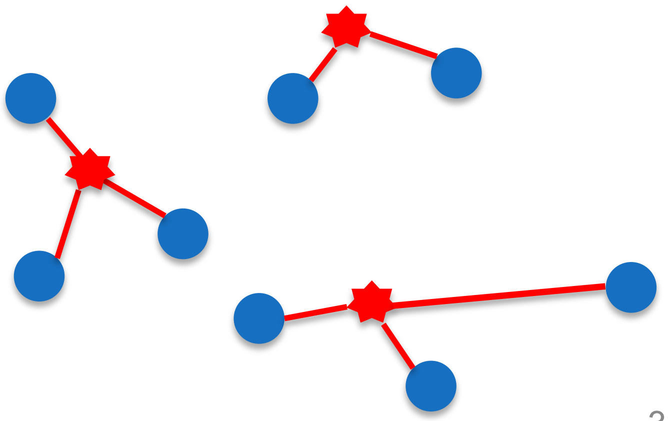
The **squared error distortion** between *Data* and *Centers*:

$$Distortion(Data, Centers) = \sum_{DataPoint\ \text{from}\ Data} d(DataPoint, Centers)^2/n$$

A single data point contributes to *MaxDistance*

All data points contribute to *Distortion*





k-Means Clustering Problem

k-Center **Clustering Problem:**
Input: A set of points *Data* and an integer *k*.
Output: A set of *k* points *Centers* that minimises

MaxDistance*(DataPoints,Centers)*

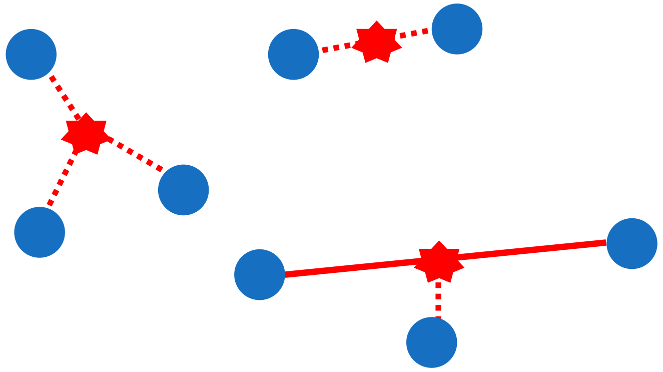
over all choices of *Centers*.

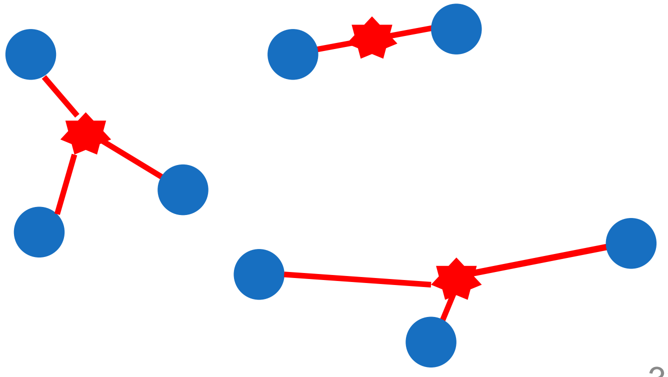
k-Means **Clustering Problem:**
Input: A set of points *Data* and an integer *k*.
Output: A set of *k* points *Centers* that minimises

Distortion(*Data*,*Centers*)

over all choices of *Centers*.

NP-Hard for $k > 1$



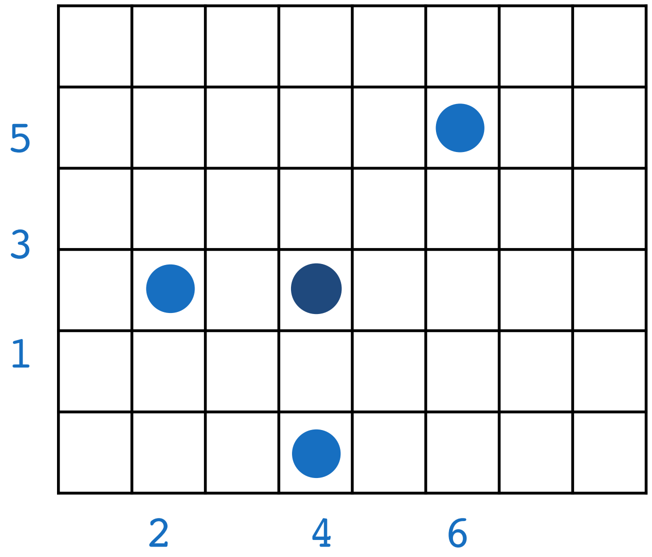


k-Means Clustering for *k* = 1

Center of Gravity Theorem: The center of gravity of points *Data* is the only point solving the 1-Means Clustering Problem.

The **center of gravity** of points *Data* is

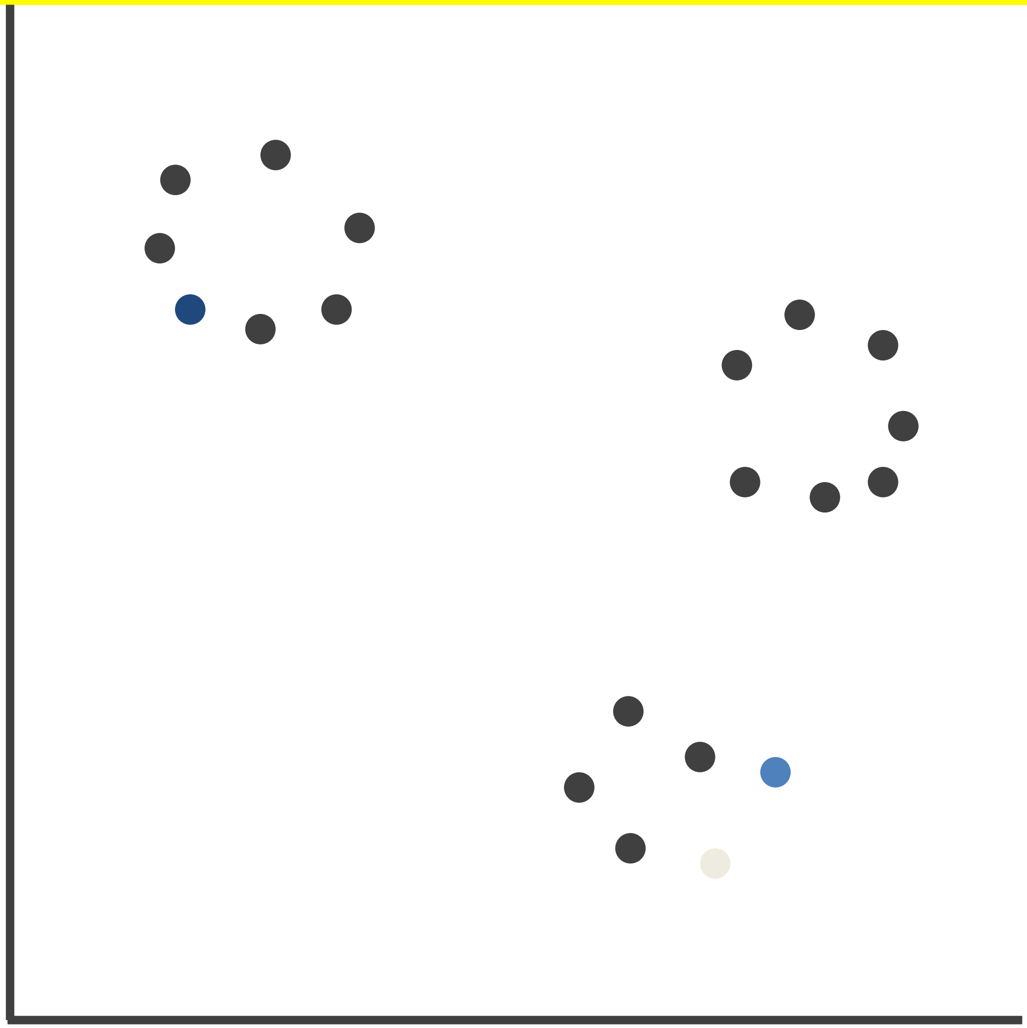
$$\sum_{\text{all points } DataPoint \text{ in } Data} DataPoint \;/\; \#\text{points in } Data$$



i-th coordinate of the center of gravity = the average of the *i*-th coordinates of datapoints:

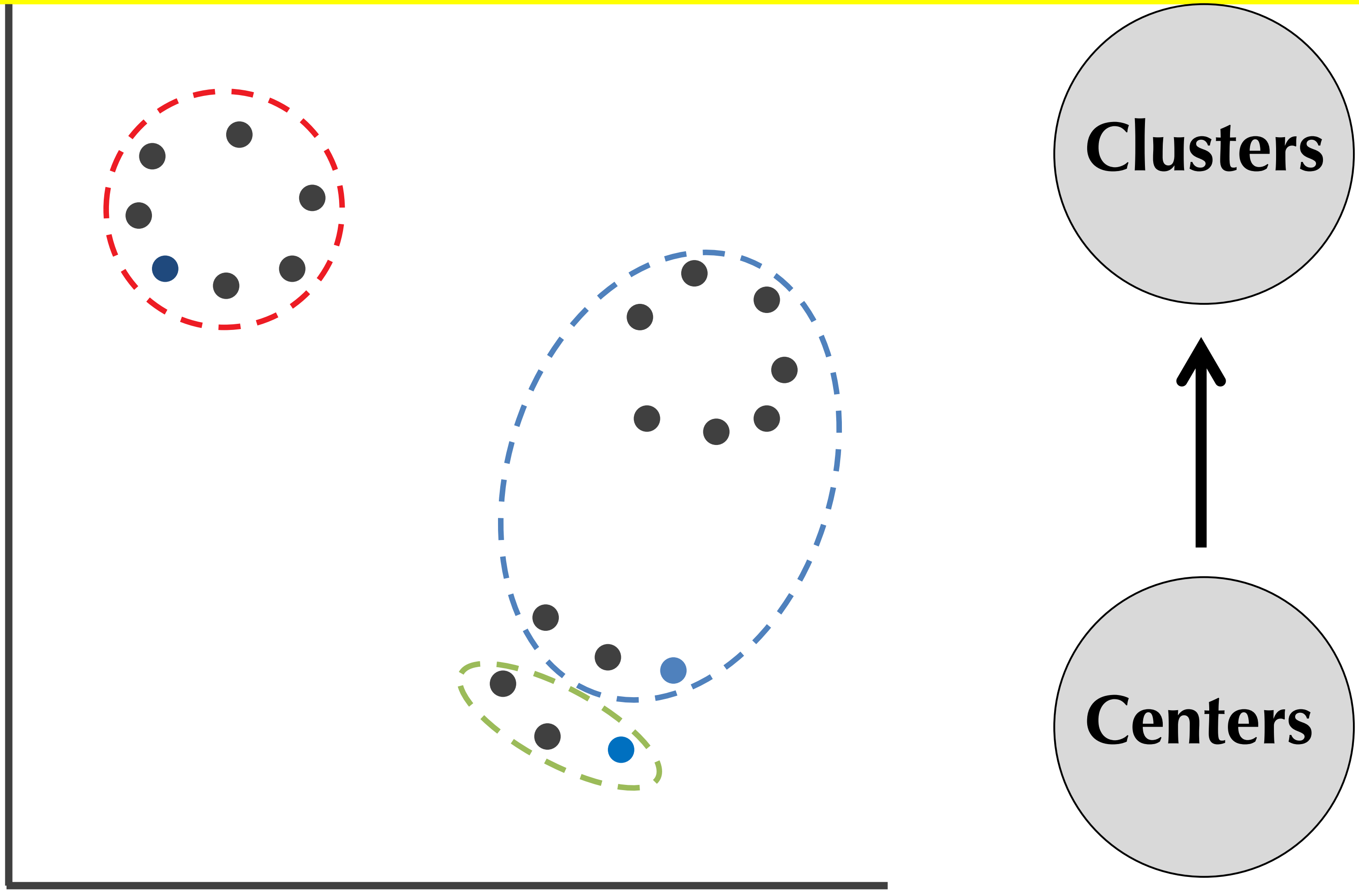
((2+4+6)/3, (3+1+5)/3) = (4, 3)

The Lloyd Algorithm in Action



Select *k* arbitrary data points as *Centers*

The Lloyd Algorithm in Action

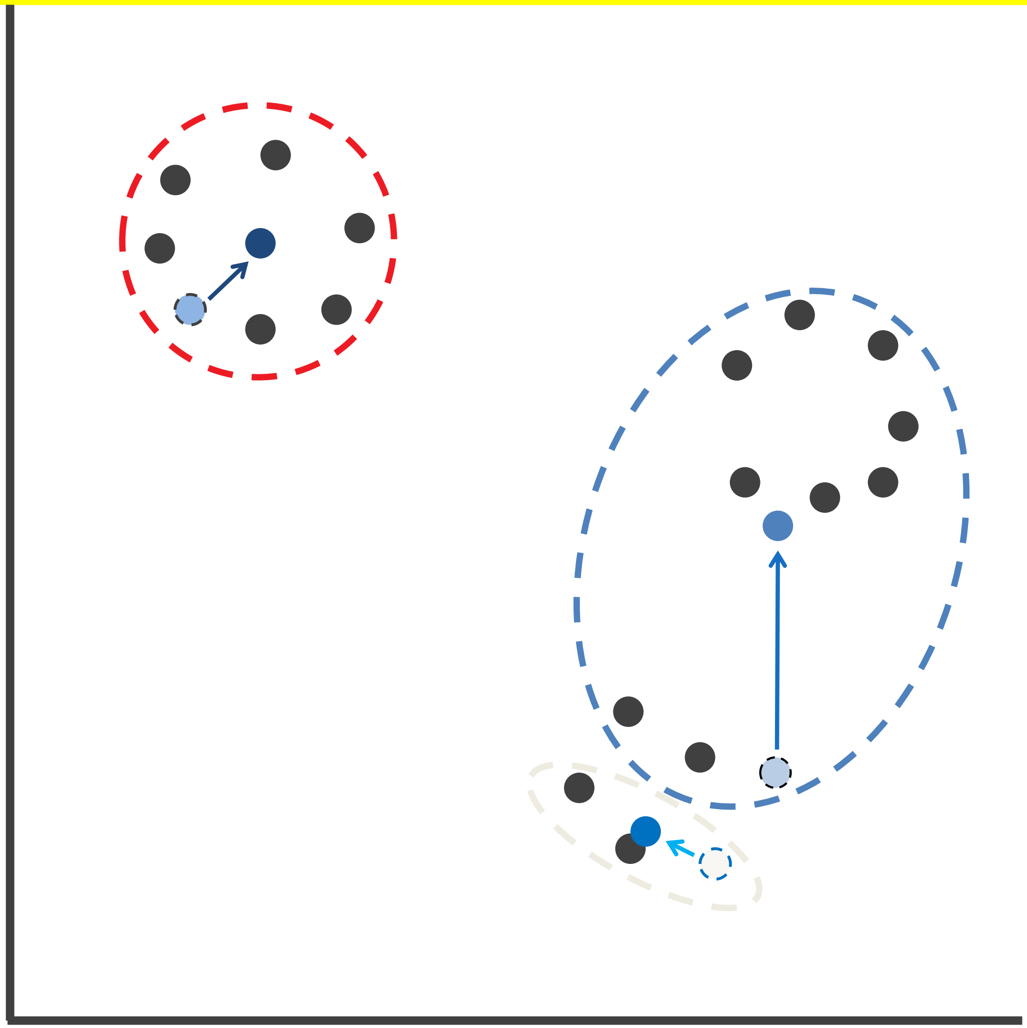


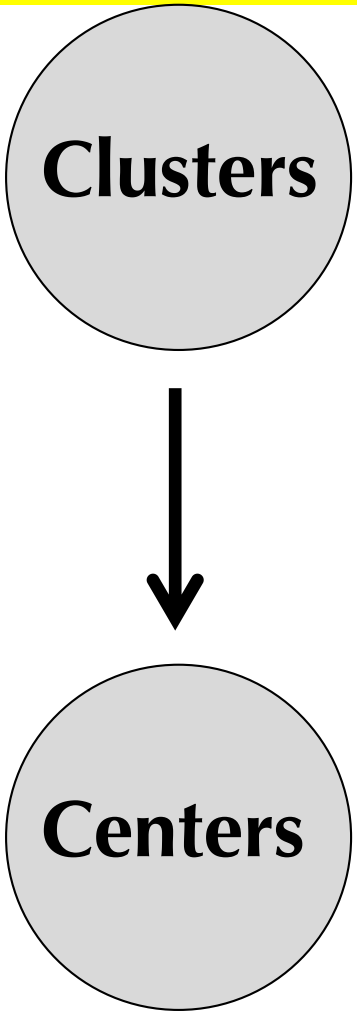
Clusters

Centers

assign each data point to its nearest center

The Lloyd Algorithm in Action



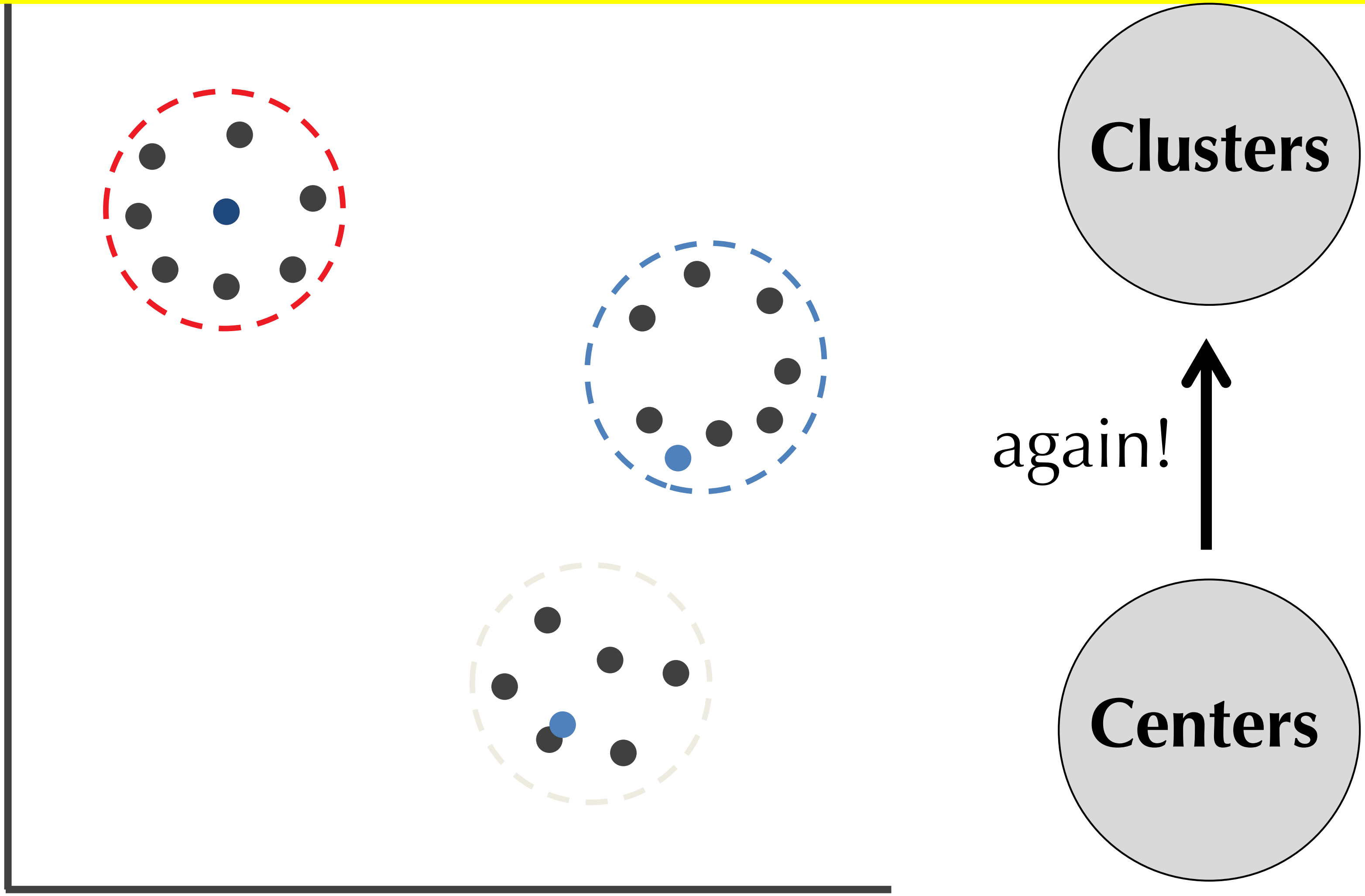


Clusters

Centers

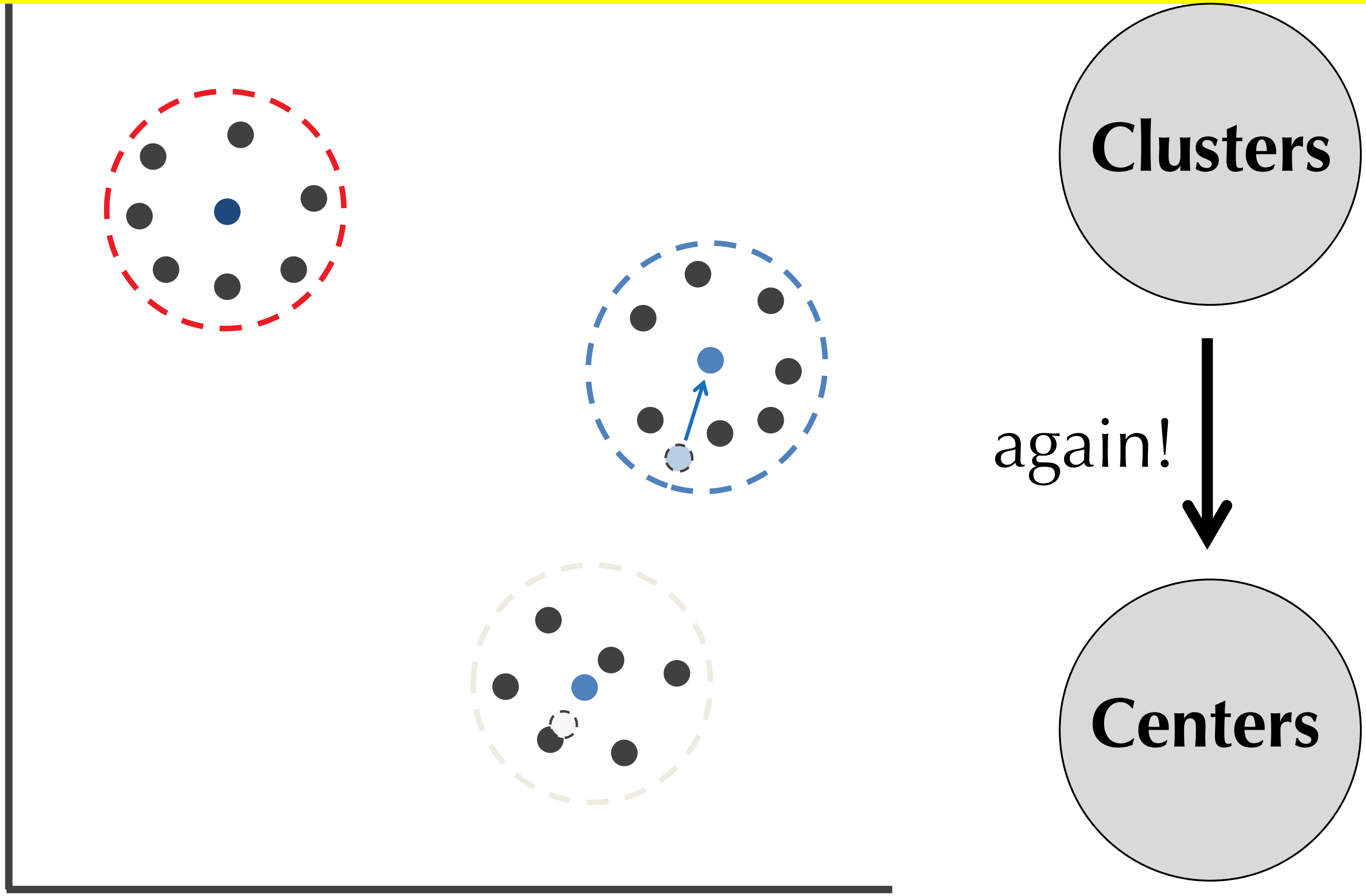
new centers ← clusters’ centers of gravity

The Lloyd Algorithm in Action



assign each data point to its nearest center

The Lloyd Algorithm in Action



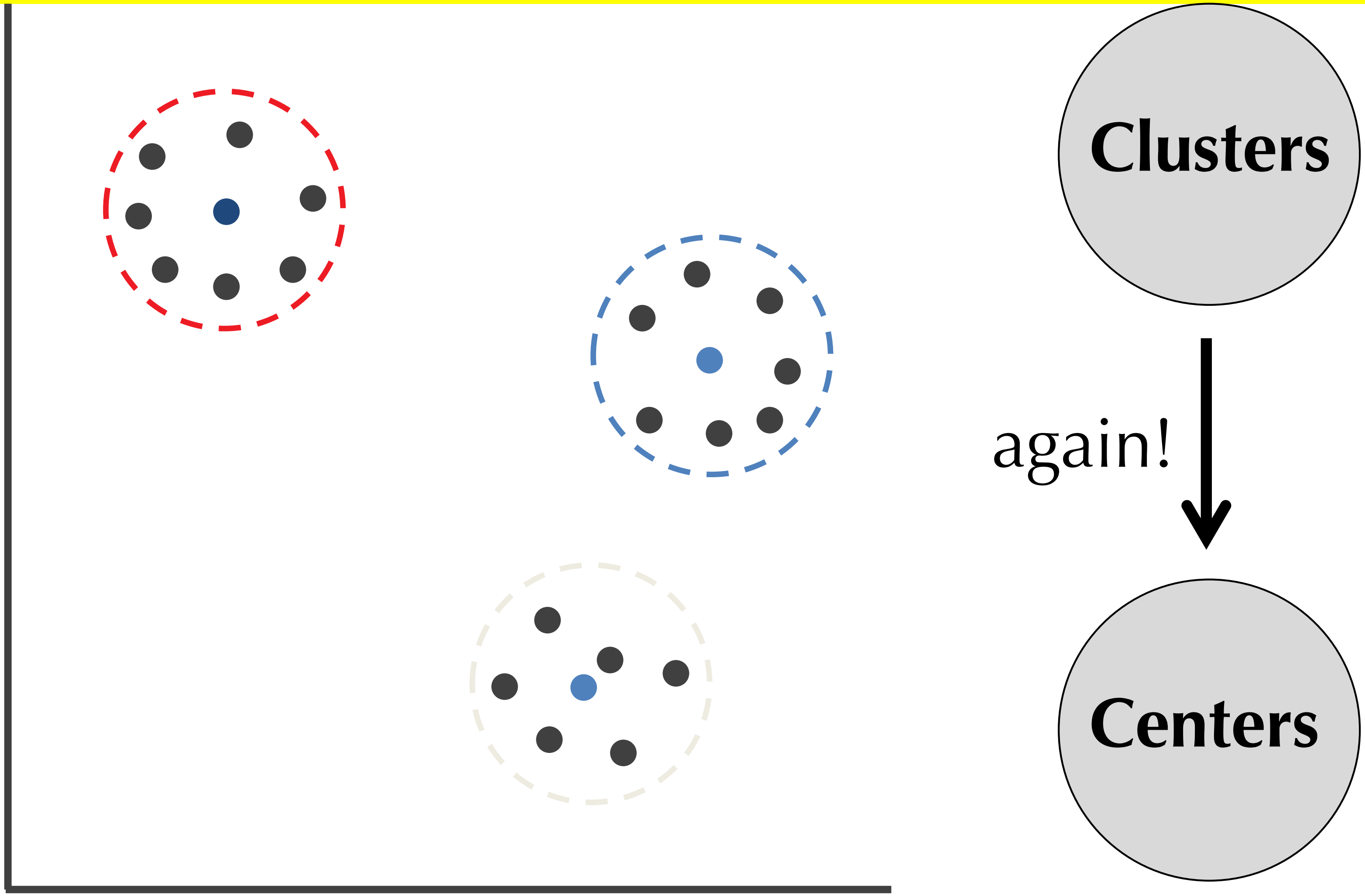
Clusters

again!

Centers

new centers ← clusters’ centers of gravity

The Lloyd Algorithm in Action



Clusters

again!

Centers

assign each data point to its nearest center

The Lloyd Algorithm

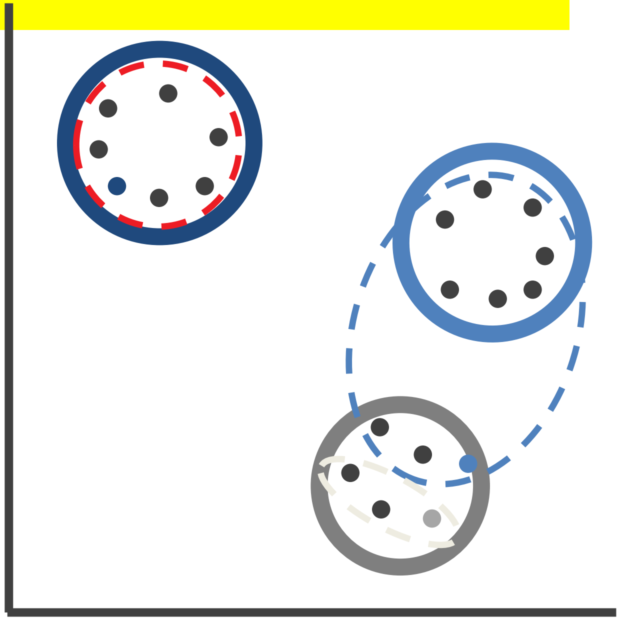
Select *k* arbitrary data points as *Centers* and then iteratively performs the following two steps:

- **Centers to Clusters**: Assign each data point to the cluster corresponding to its nearest center (ties are broken arbitrarily).
- **Clusters to Centers**: After the assignment of data points to *k* clusters, compute new centers as clusters' center of gravity.

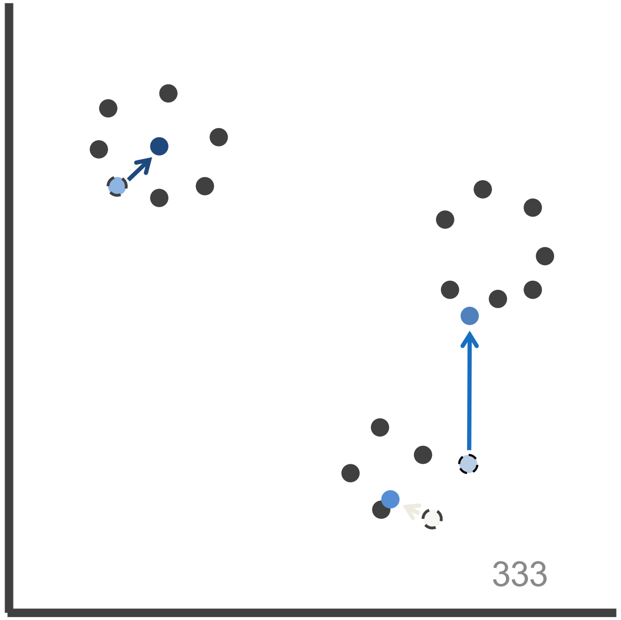
The Lloyd algorithm terminates when the centers stop moving (**convergence**).

Must the Lloyd Algorithm Converge?

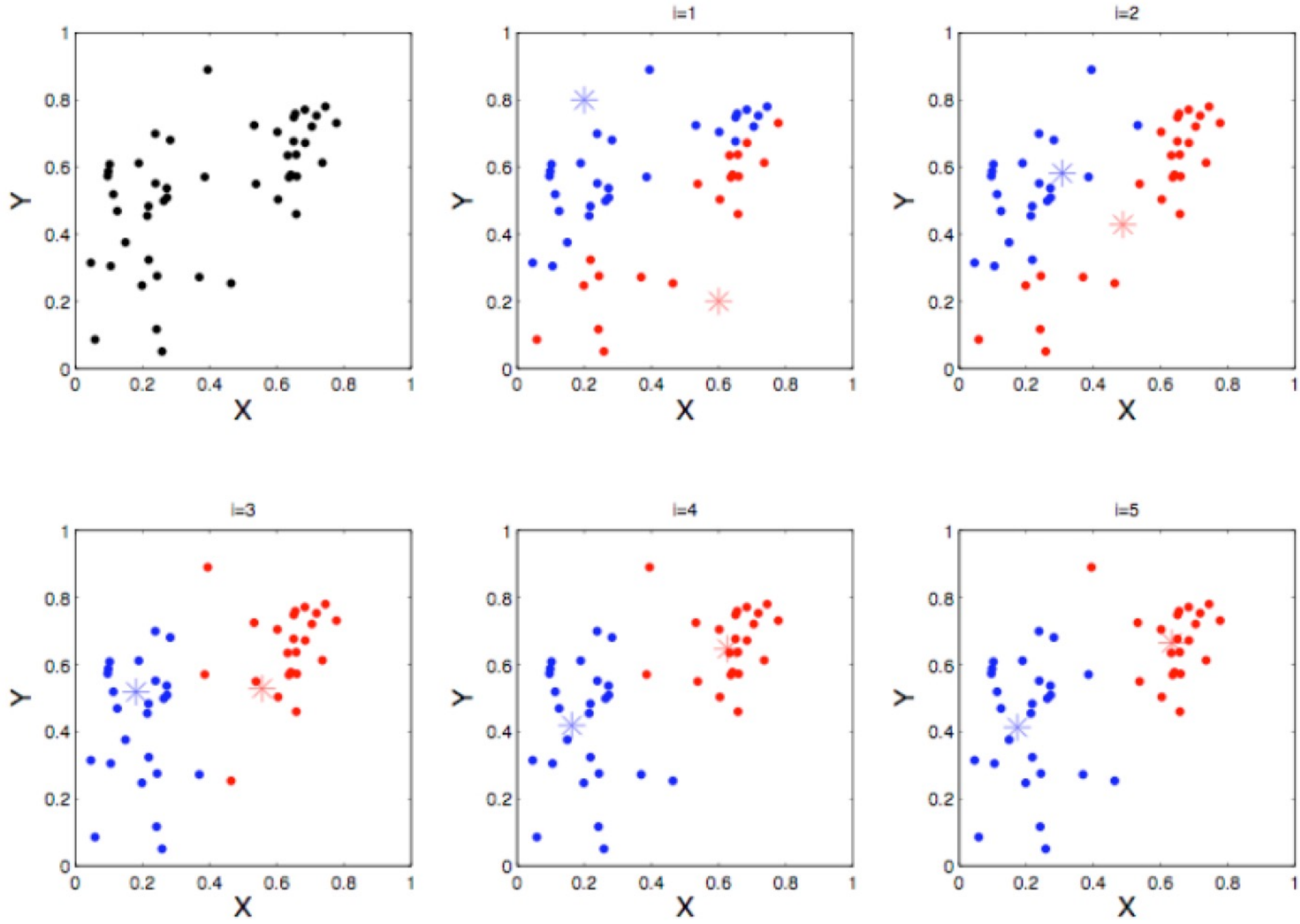
- If a data point is assigned to a new center during the **Centers to Clusters** step:
  - the squared error distortion is reduced because this center must be closer to the point than the previous center was.



- If a center is moved during the **Clusters to Centers** step:
  - the squared error distortion is reduced since the center of gravity is the *only point* minimising the distortion (the Center of Gravity Theorem).

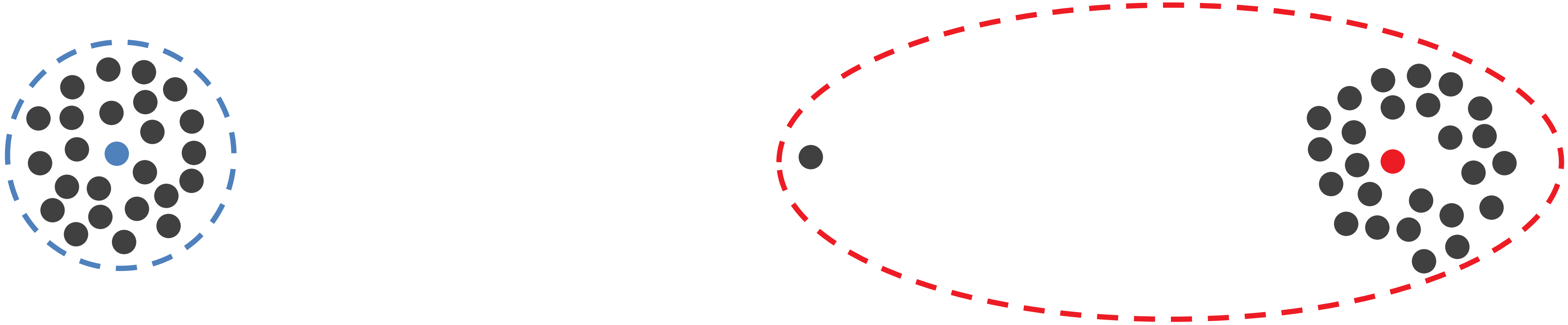


RECAP



Soft vs. Hard Clustering

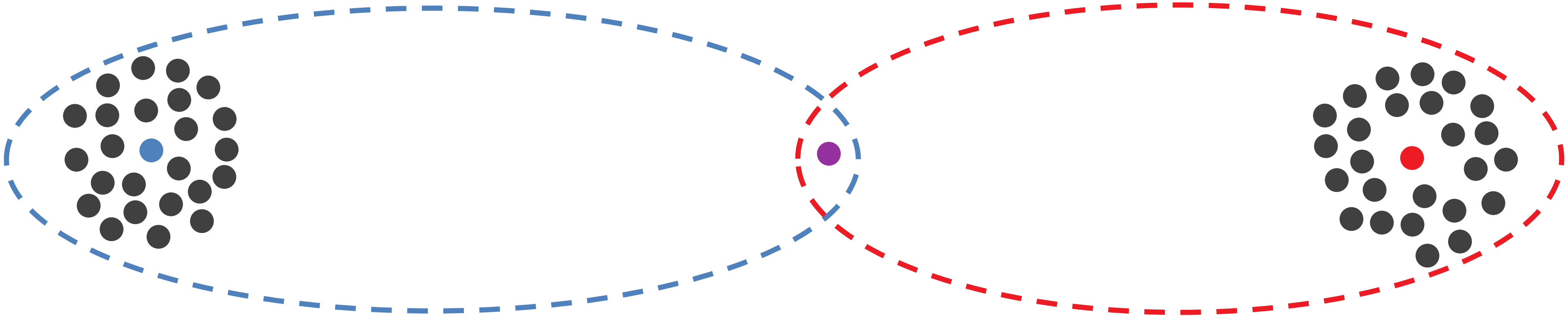
- The Lloyd algorithm assigns the midpoint either to the red or to the blue cluster.
  - "**hard**" assignment of data points to clusters.



Midpoint: A point approximately halfway between two clusters.

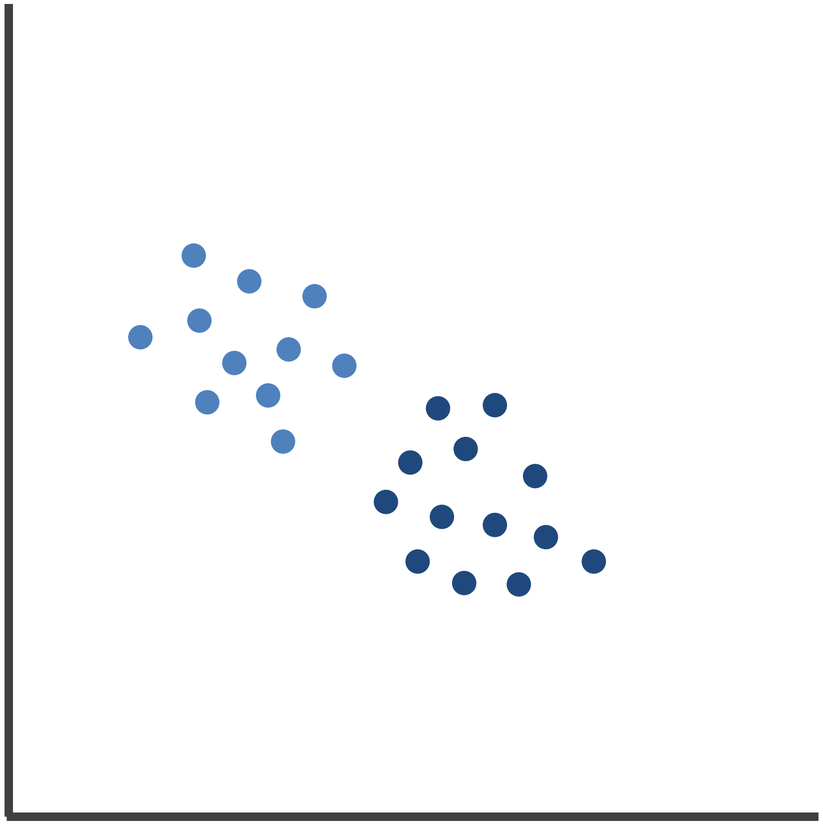
Soft vs. Hard Clustering

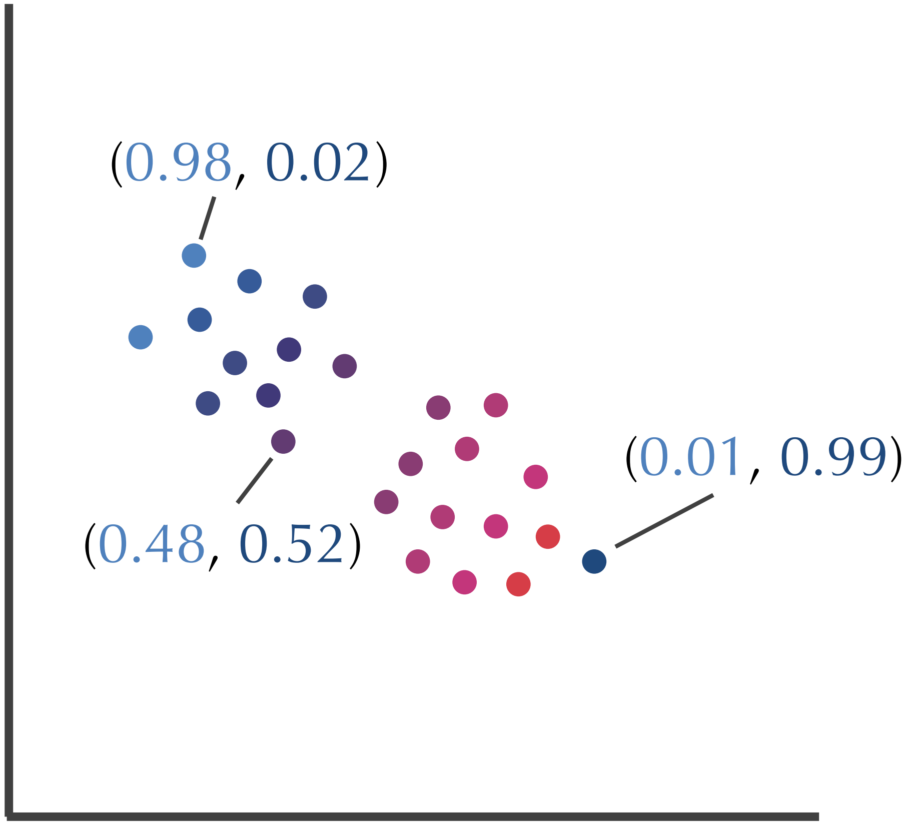
- The Lloyd algorithm assigns the midpoint either to the red or to the blue cluster.
  - "**hard**" assignment of data points to clusters.



- Can we color the midpoint half-red and half-blue?
  - "**soft**" assignment of data points to clusters.

Soft vs. Hard Clustering





Hard choices: points are colored red or blue depending on their cluster membership.

Soft choices: points are assigned "red" and "blue" *responsibilities* r_{blue} and r_{red} $(r_{\text{blue}} + r_{\text{red}} = 1)$

Flipping One Biased Coin

- We flip a loaded coin with an **unknown bias θ** (probability that the coin lands on heads).
- The coin lands on heads ***i* out of *n*** times.
- For each bias, we can compute the probability of the resulting sequence of flips.

Probability of generating the given sequence of flips is

$$\Pr(\text{sequence}|\theta) = \theta^{i} * (1-\theta)^{n-i}$$

This expression is maximised at **θ= *i*/*n*** (most likely bias)

A

Flipping Two Biased Coins

B

Data

HTTTHTTHTH	0.4
HHHHTHHHHH	0.9
HTHHHHHTHH	0.8
HTTTTTHHTT	0.3
THHHTHHHTH	0.7

Goal: estimate the probabilities θ_A and θ_B

If We Knew Which Coin Was Used in Each Sequence…

	Data	*HiddenVector*
HTTTHTTHTH	0.4	1
HHHHTHHHHH	0.9	0
HTHHHHHTHH	0.8	0
HTTTTTHHTT	0.3	1
THHHTHHHTH	0.7	0

Goal: estimate *Parameters* = (θ_A ,θ_B) when *HiddenVector* is given

If We Knew Which Coin Was Used in Each Sequence…

	Data	*HiddenVector*
HTTTHTTHTH	0.4	1
HHHHTHHHHH	0.9	0
HTHHHHHTHH	0.8	0
HTTTTTHHTT	0.3	1
THHHTHHHTH	0.7	0

θ_A = fraction of heads generated in all flips with coin *A* = (4+3) / (10+10) = (0.4+0.3) / 2 = 0.35

θ_B = fraction of heads generated in all flips with coin *B* = (9+8+7) / (10+10+10) = (0.9+0.8+0.7) / (1+1+1) = 0.80

Parameters as a Dot-Product

	Data		***HiddenVector***	***Parameters***=(θ_A, θ_B)
HTTTHTTHTH	**0.4**	*	**1**	
HHHHTHHHHH	**0.9**	*	**0**	
HTHHHHHTHH	**0.8**	*	**0**	(0.35, 0.80)
HTTTTTHHTT	**0.3**	*	**1**	
THHHTHHHTH	**0.7**	*	**0**	

θ_A = fraction of heads generated in all flips with coin A =
= (4+3) / (10+10) = (0.4+0.3) / 2 = 0.35

(0.4*1+0.9*0+0.8*0+0.3*1+0.7*0)/ (1+0+0+1+0) = 0.35

$\sum_{\text{all data points } i} Data_i * HiddenVector_i \;/\; \sum_{\text{all data points } i} HiddenVector_i = 0.35$

Data * *HiddenVector* / (1,**1**,…,*1)**HiddenVector* =0.35

1 refers to a vector (1,1, … ,1) consisting of all 1's

Parameters as a Dot-Product

	Data		***HiddenVector***	***Parameters***=(θ_A, θ_B)
HTTT**H**TT**H**T**H**	0.4	*	1	
HHHHT**HHHHH**	0.9	*	0	
HT**HHHHHH**T**HH**	0.8	*	0	(0.35, 0.80)
HTTTTT**HH**TT	0.3	*	1	
T**HHH**T**HHH**T**H**	0.7	*	0	

θ_B = fraction of heads generated in all flips with coin *B*
= (9+8+7) / (10+10+10) = (0.9+0.8+0.7) /(1+1+1) = 0.80

(0.5*0+0.9*1+0.8*1+0.4*0+0.7*1) / (0+1+1+0+1) = 0.80

$\sum_{\text{all points } i}$ *Data*$_\text{i}$ * (1- *HiddenVector*$_\text{i}$) / $\sum_{\text{all points } i}$ (1- *HiddenVector*$_\text{i}$)=

Data * (1-*HiddenVector*) / **1** * (1 - *HiddenVector*)

Parameters as a Dot-Product

	Data		***HiddenVector***	***Parameters***=(θ_A, θ_B)
HTTT**H**TT**H**T**H**	**0.4**	*	**1**	
HHHHT**HHHHH**	**0.9**	*	**0**	
HT**HHHHHH**T**HH**	**0.8**	*	**0**	(0.35, 0.80)
HTTTTT**HH**TT	**0.3**	*	**1**	
T**HHH**T**HHHH**T**H**	**0.7**	*	**0**	

θ_A = fraction of heads generated in all flips with coin *A*

= (0.4+0.3)/2=0.35

= *Data* * *HiddenVector* / **1** * *HiddenVector*

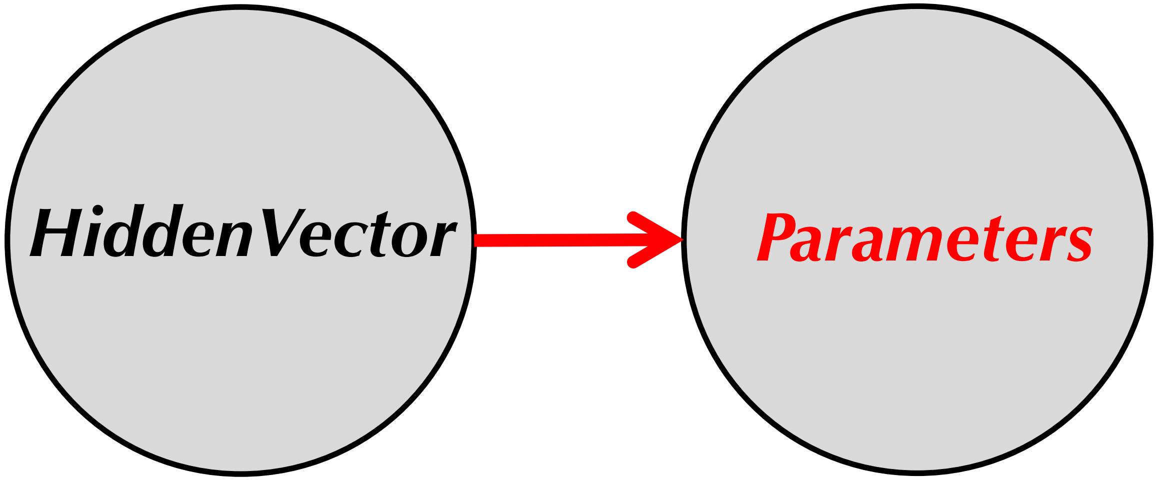
θ_B = fraction of heads generated in all flips with coin *B*

= (0.9+0.8+0.7)/3=0.80

= *Data* * (**1**-*HiddenVector*) / **1** * (**1** - *HiddenVector*)

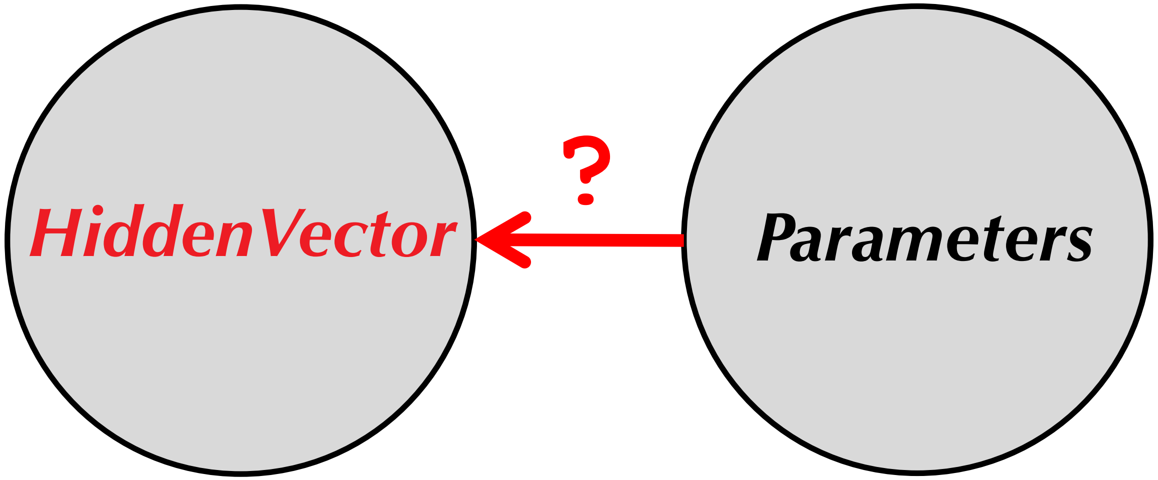
Data, HiddenVector, Parameters

Data	*HiddenVector*	*Parameters*=(θ_A, θ_B)
0.4	1	
0.9	0	
0.8	0	→ (0.35, 0.80)
0.3	1	
0.7	0	



Data, HiddenVector, Parameters

Data	*HiddenVector*	*Parameters*=(θ_A, θ_B)
0.4	?	
0.9	?	
0.8	?	← (0.35, 0.80)
0.3	?	
0.7	?	



From *Data & Parameters to HiddenVector*

Data	***HiddenVector***	***Parameters***=(θ_A, θ_B)
0.4	**?**	
0.9	**?**	
0.8	**?** ←	(0.35, 0.80)
0.3	**?**	
0.7	**?**	

Which coin is more likely to generate the 1st sequence (with 4 H)?

$\Pr(1^{st}\ \text{sequence}|\theta_A) = \theta_A^4\,(1-\theta_A)^6 = 0.35^4 \bullet 0.65^6 \approx 0.00113$ >

$\Pr(1^{st}\ \text{sequence}|\theta_B) = \theta_B^4(1-\theta_B)^6 = 0.80^4 \bullet 0.20^6 \approx 0.00003$

From *Data & Parameters to HiddenVector*

Data	***HiddenVector***		***Parameters***=(θ_A, θ_B)
0.4	1		
0.9	?		
0.8	?	←	(0.35, 0.80)
0.3	?		
0.7	?		

Which coin is more likely to generate the 1st sequence (with 4 H)?

$\Pr(1^{st}\text{ sequence}|\theta_A)=\theta_A{}^4\,(1\text{-}\theta_A)^6 = 0.35^4 \bullet 0.65^6 \approx 0.00113$ >

$\Pr(1^{st}\text{ sequence}|\theta_B\,)=\theta_B{}^4(1\text{-}\theta_B)^6 = 0.80^4 \bullet 0.20^6 \approx 0.00003$

From *Data & Parameters* to *HiddenVector*

Data	***HiddenVector***		***Parameters***=(θ_A, θ_B)
0.4	**1**		
0.9	**?**		
0.8	**?**	←	(0.35, 0.80)
0.3	**?**		
0.7	**?**		

Which coin is more likely to generate the 2nd sequence (with 9 H)?

$\Pr(2^{nd}\text{ sequence}|\theta_A)= \theta_A{}^{9}\,(1\text{-}\theta_A)^{1}=0.35^{9}\bullet 0.65^{1} \approx 0.00005$ $<$

$\Pr(2^{nd}\text{ sequence}|\theta_B)= \theta_B{}^{9}\,(1\text{-}\theta_B)^{1}=0.80^{9}\bullet 0.20^{1} \approx 0.02684$

From *Data & Parameters* to *HiddenVector*

Data	***HiddenVector***		***Parameters***=(θ_A, θ_B)
0.4	1		
0.9	0		
0.8	?	←	(0.35, 0.80)
0.3	?		
0.7	?		

Which coin is more likely to generate the 2nd sequence (with 9 H)?

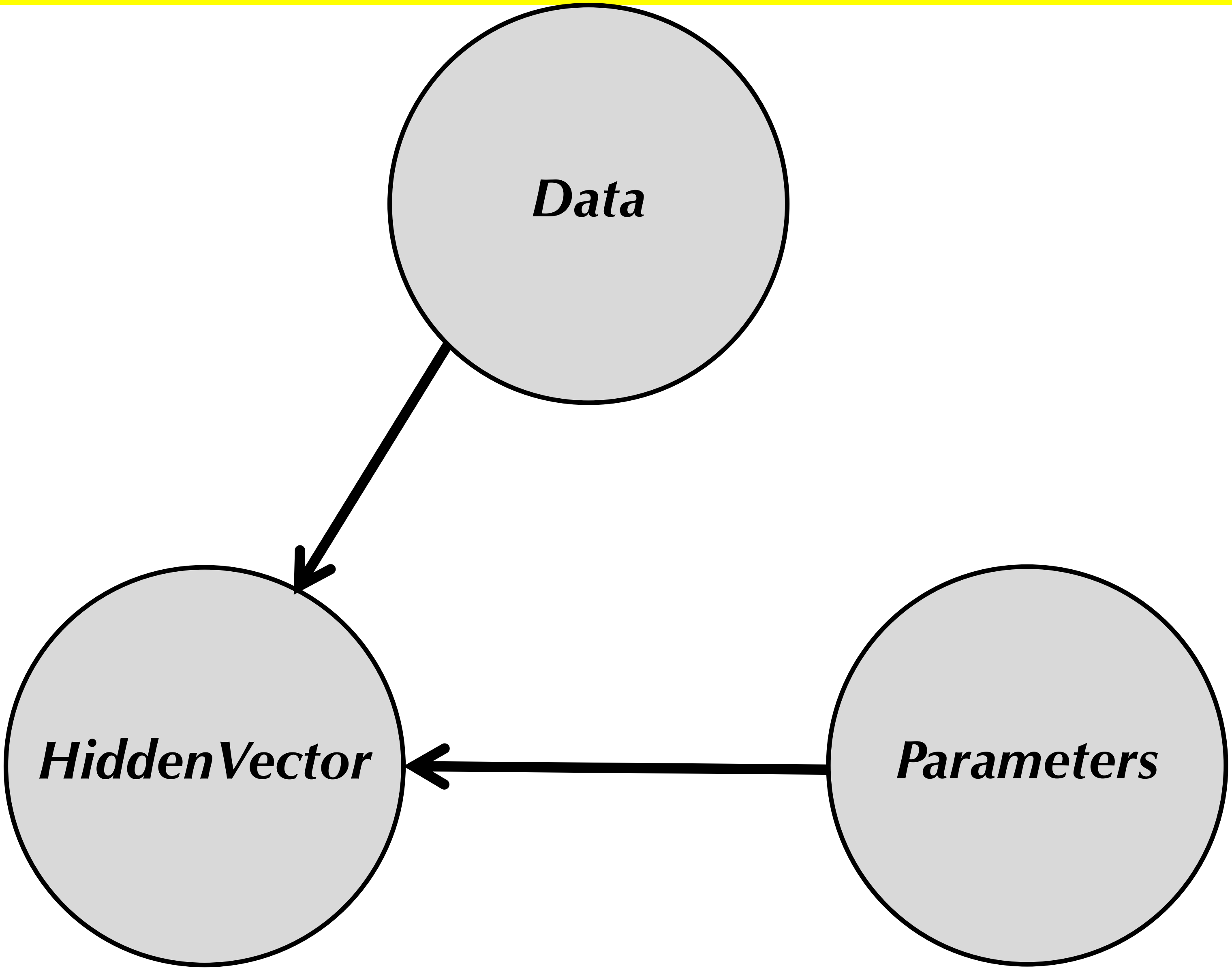
$\Pr(2^{nd}\text{ sequence}|\theta_A) = \theta_A^{\,9}\,(1\text{-}\theta_A)^1 = 0.35^9 \bullet 0.65^1 \approx\ 0.00005\ <$

$\Pr(2^{nd}\text{ sequence}|\theta_B) = \theta_B^{\,9}\,(1\text{-}\theta_B)^1 = 0.80^9 \bullet 0.20^1 \approx 0.02684$

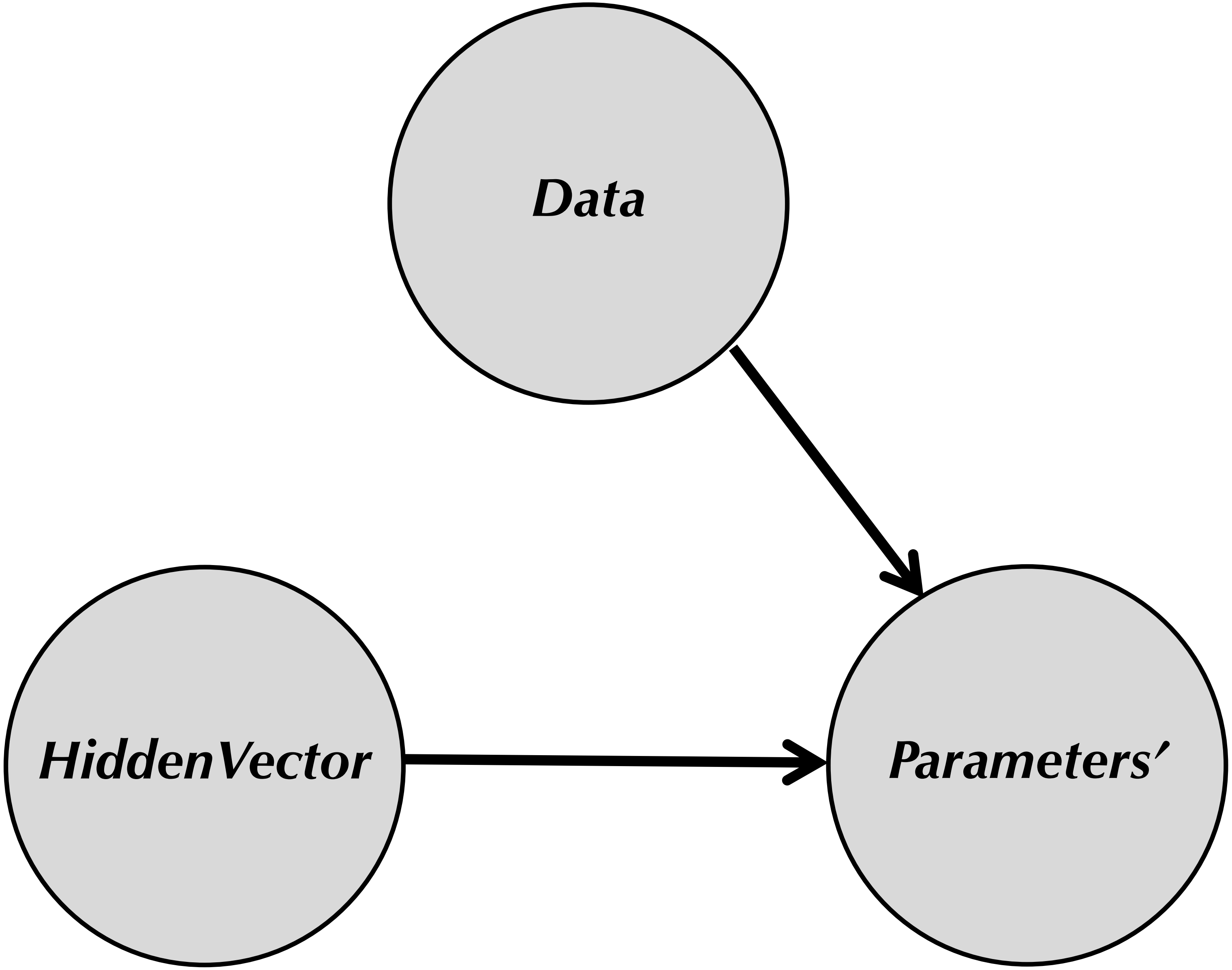
HiddenVector Reconstructed!

Data	*HiddenVector*	*Parameters*=(θ_A, θ_B)
0.4	1	
0.9	0	
0.8	0	← (0.35, 0.80)
0.3	1	
0.7	0	

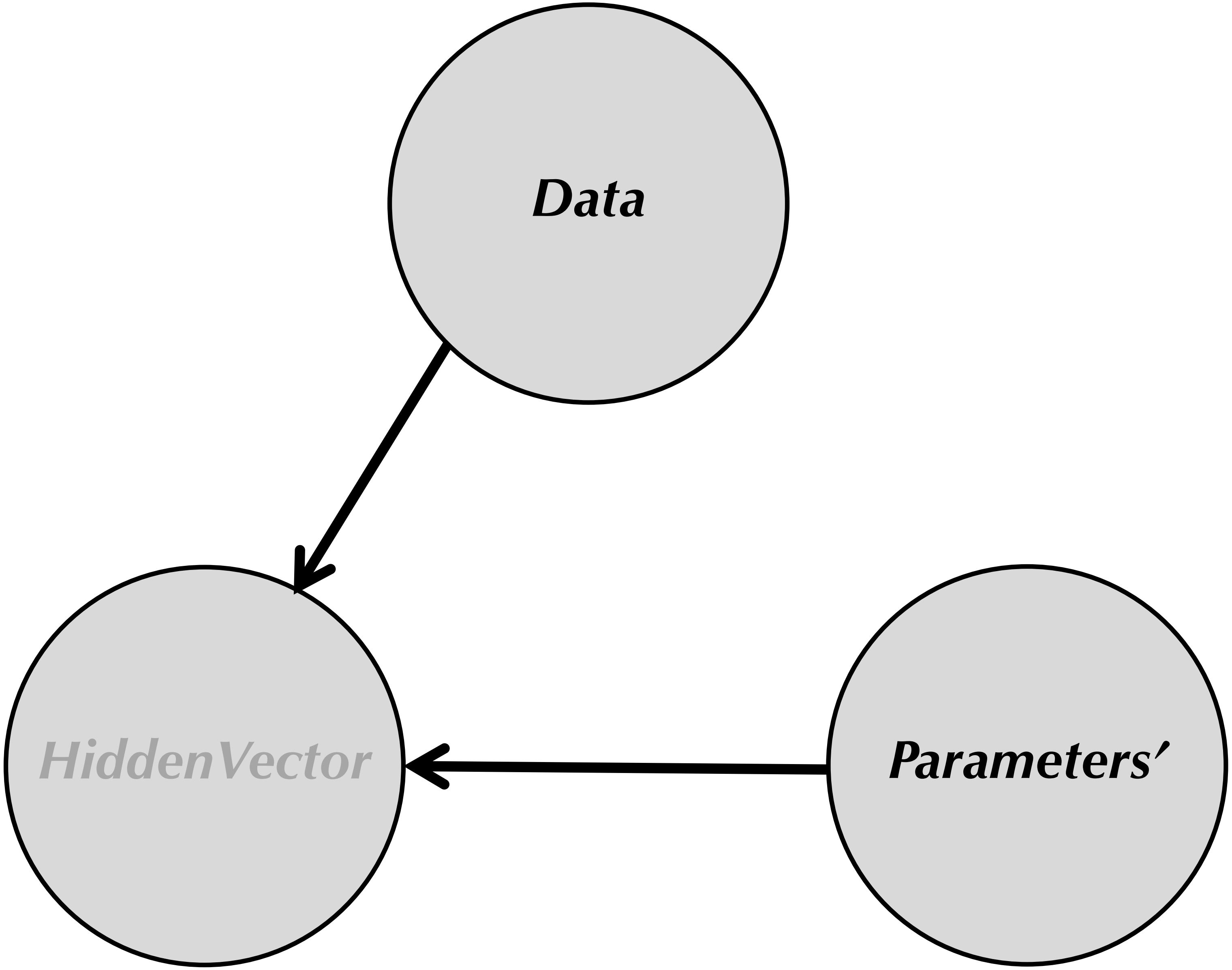
Reconstructing *HiddenVector* and *Parameters*



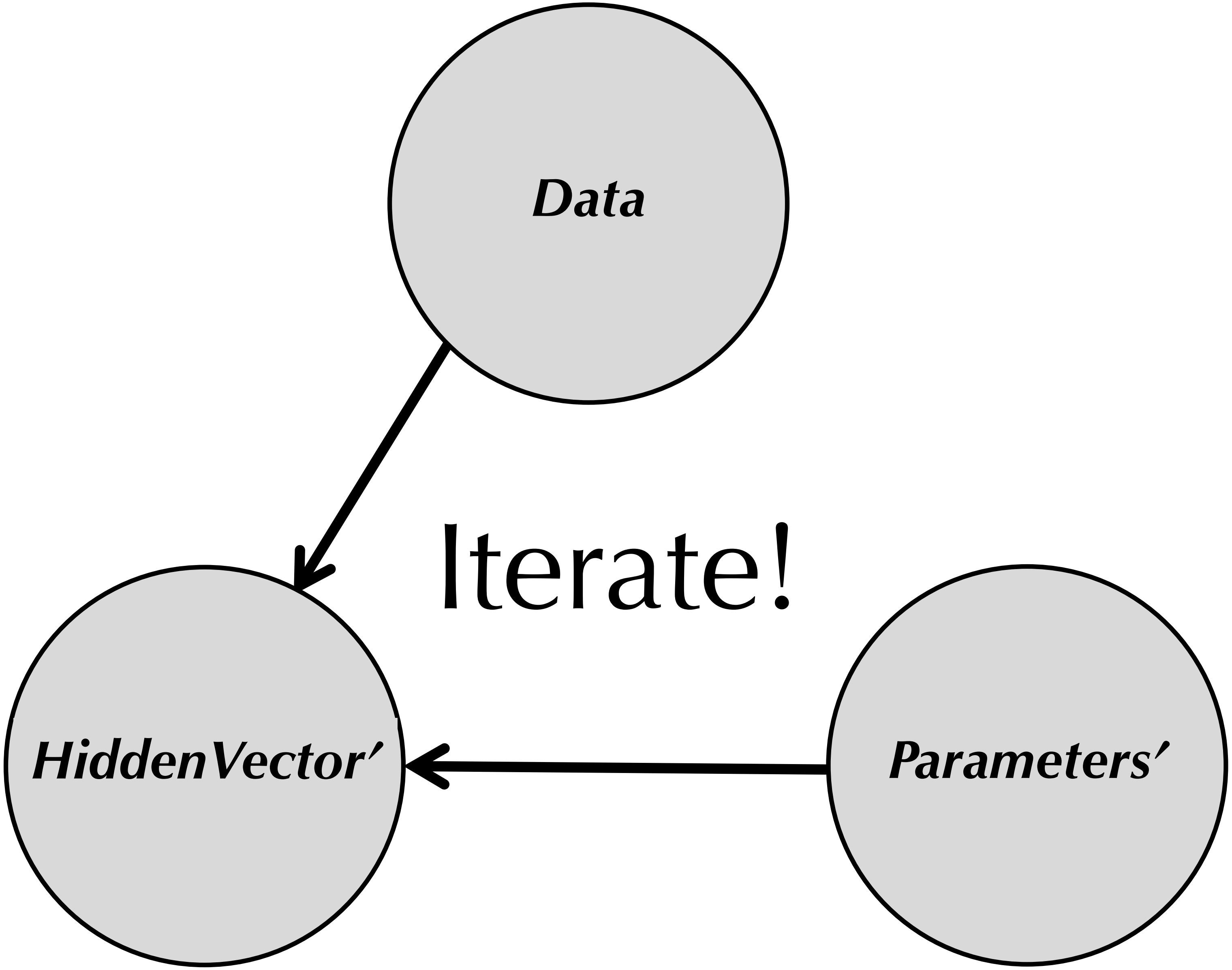
Reconstructing *HiddenVector* and *Parameters*



Reconstructing *HiddenVector* and *Parameters*



Reconstructing *HiddenVector* and *Parameters*

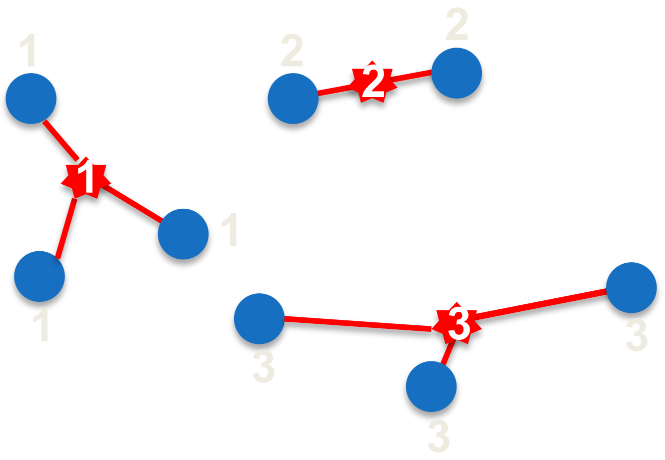


From Coin Flipping to k-means Clustering: Where Are *Data*, *HiddenVector*, and *Parameters*?

Data: data points $Data = (Data_1,\dots,Data_n)$

Parameters: $Centers = (Center_1,\dots,Center_k)$

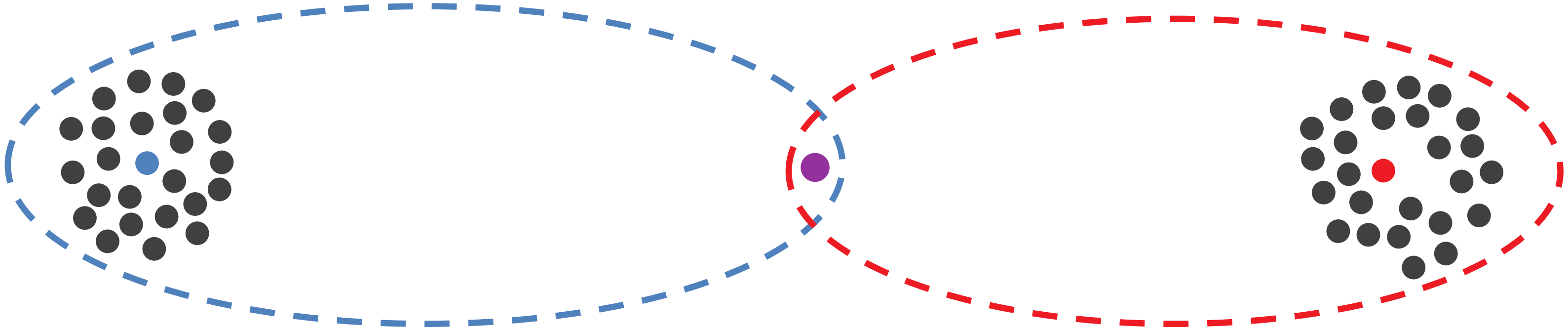
HiddenVector: assignments of data points to k centers (n-dimensional vector with coordinates varying from 1 to k).



Coin Flipping and Soft Clustering

- **Coin flipping**: how would you select between coins A and B if $\Pr(\text{sequence}|\theta_A) = \Pr(\text{sequence}|\theta_B)$?

- ***k*-means clustering:** what cluster would you assign a data point it to if it is a midpoint of centers C_1 and C_2?



Soft assignments: assigning C_1 and C_2 "responsibility" ≈0.5 for a midpoint.

From *Data & Parameters to HiddenVector*

Data	***HiddenVector***	***Parameters*** = (θ_A, θ_B)
0.4	?	
0.9	?	
0.8	?	← (0.60, 0.82)
0.3	?	
0.7	?	

Which coin is more likely to have generated the first sequence (with 4 H)?

$\Pr(1^{st}\text{ sequence}|\theta_A)=\theta_A{}^5\,(1-\theta_A)^5 = 0.60^4 \bullet 0.40^6 \approx 0.000531$ >
$\Pr(1^{st}\text{ sequence}|\theta_B\,)= \theta_B{}^5(1-\theta_B)^5 = 0.82^4 \bullet 0.18^6 \approx 0.000015$

Memory Flash:
From *Data & Parameters to HiddenVector*

Data	***HiddenVector***	***Parameters* = (θ_A, θ_B)**
0.4	**1**	
0.9	**?**	
0.8	**?**	← (0.60, 0.82)
0.3	**?**	
0.7	**?**	

Which coin is more likely to have generated the first sequence (with 4 H)?

$\Pr(1^{st} \text{ sequence}|\theta_A) = \theta_A{}^5\ (1-\theta_A)^5 = 0.60^4 \bullet 0.40^6 \approx 0.000531$ **>**
$\Pr(1^{st} \text{ sequence}|\theta_B\ ) = \theta_B{}^5 (1-\theta_B)^5 = 0.82^4 \bullet 0.18^6 \approx 0.000015$

From *Data & Parameters* to ***HiddenMatrix***

Data	***HiddenMatrix***		***Parameters*** = (θ_A, θ_B)
0.4	0.97 0.03		
0.9	?		
0.8	?	←	(0.60, 0.82)
0.3	?		
0.7	?		

What are the **responsibilities** of coins for this sequence?

$\Pr(1^{st} \text{ sequence} | \theta_A) \approx 0.000531 >$

$\Pr(1^{st} \text{ sequence} | \theta_B) \approx 0.000015$

$0.000531 / (0.000531 + 0.000015) \approx 0.97$

$0.000015 / (0.000531 + 0.000015) \approx 0.03$

From *Data & Parameters* to ***HiddenMatrix***

Data	***HiddenMatrix***		***Parameters*** = (θ_A, θ_B)
0.4	0.97	0.03	
0.9	0.12	0.88	
0.8	?		← (0.60, 0.82)
0.3	?		
0.7	?		

What are the responsibilities of coins for the 2nd sequence?

$\Pr(2^{nd}\text{ sequence}|\theta_A) \approx 0.0040 <$

$\Pr(2^{nd}\text{ sequence}|\theta_B\ ) \approx 0.0302$

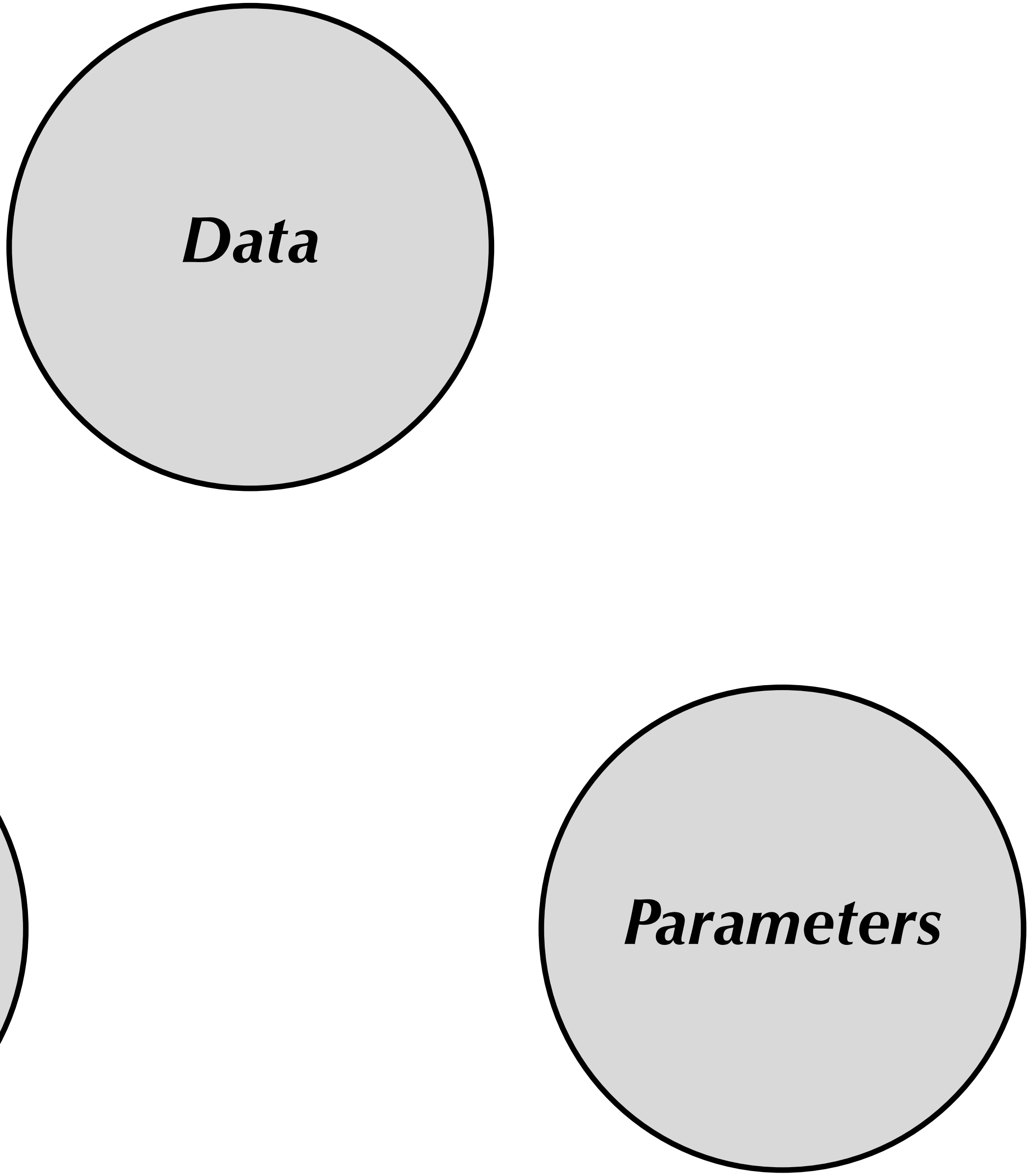
$0.0040\ /\ (0.0040 + 0.0302) = 0.12$

$0.0342\ /\ (0.0040 + 0.0342) = 0.88$

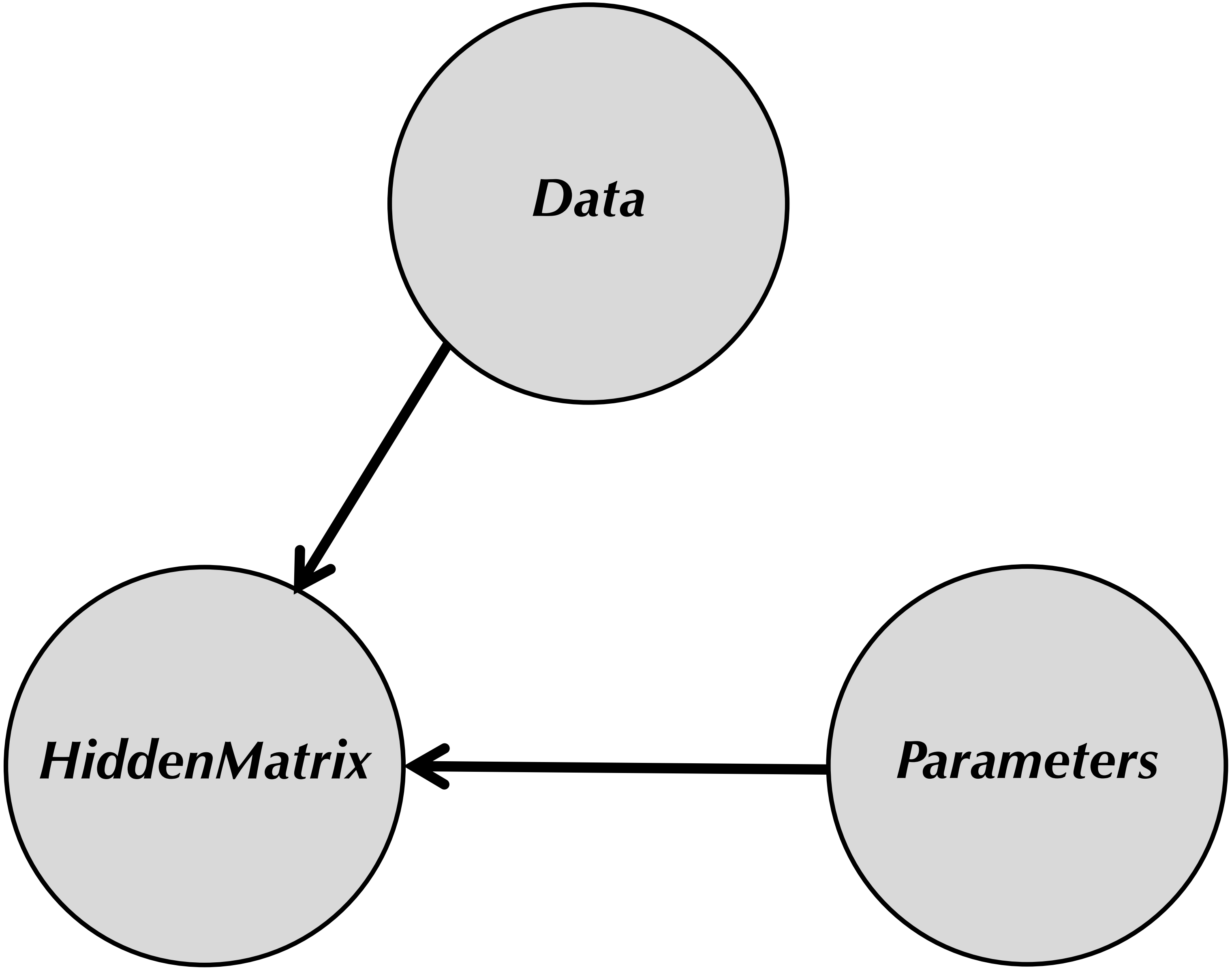
HiddenMatrix Reconstructed!

Data	***HiddenMatrix***		***Parameters*** = (θ_A, θ_B)
0.4	**0.97**	**0.03**	
0.9	**0.12**	**0.88**	
0.8	**0.29**	**0.71**	← (0.60, 0.82)
0.3	**0.99**	**0.01**	
0.7	**0.55**	**0.45**	

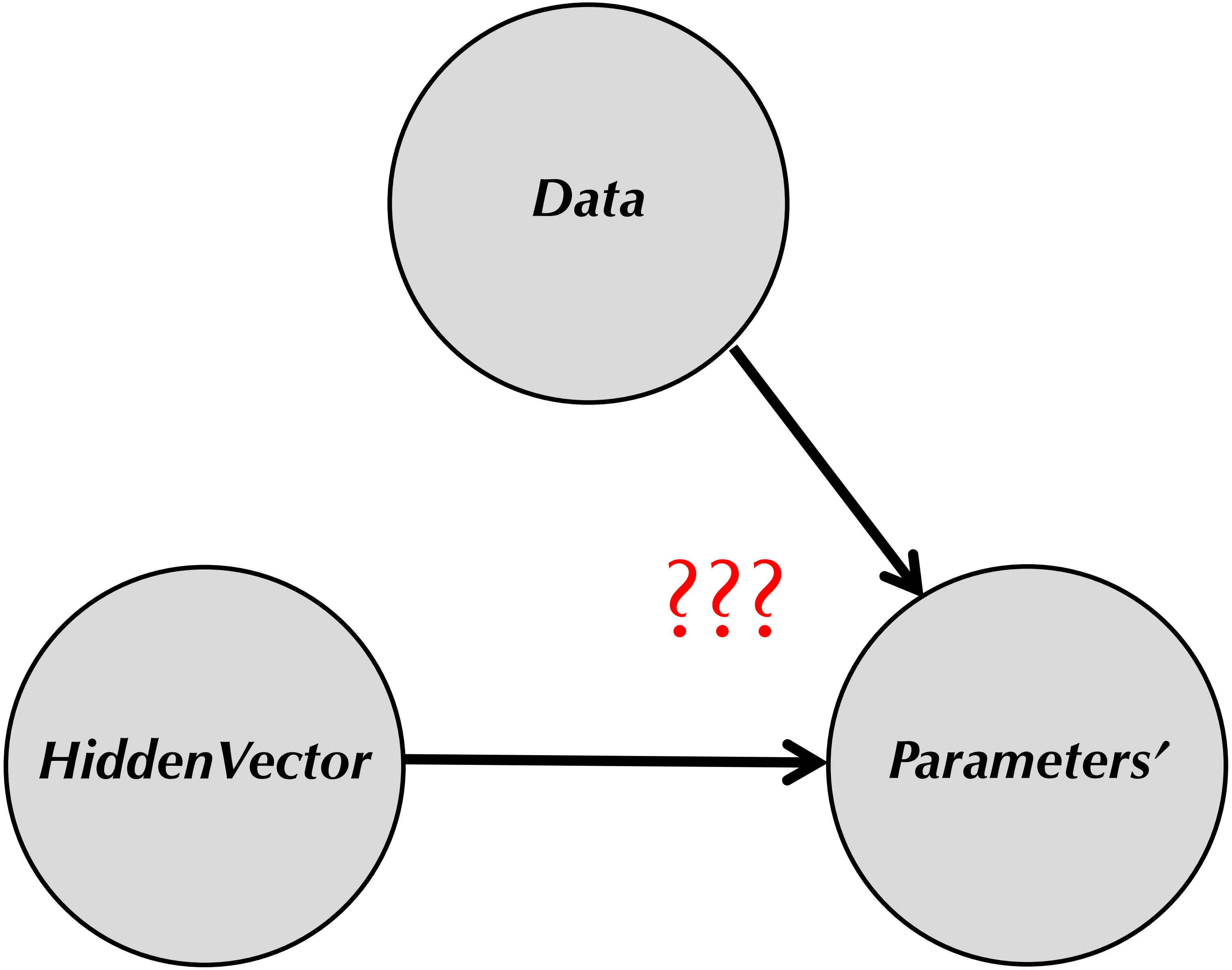
Expectation Maximization Algorithm



E-step



M-step



Memory Flash: Dot Product

	Data		*HiddenVector*	*Parameters*=(θ_A, θ_B)
HTTTHTTHTH	0.4	*	1	
HHHHTHHHHH	0.9	*	0	
HTHHHHHTHH	0.8	*	0	???
HTTTTTHHTT	0.3	*	1	
THHHTHHHTH	0.7	*	0	

θ_A = *Data* * *HiddenVector* / **1** * *HiddenVector*

θ_B = *Data* * (**1**-*HiddenVector*) / **1** * (**1**-*HiddenVector*)

From *Data* & *HiddenMatrix* to ***Parameters***

	Data	***HiddenVector***	***Parameters***=(θ_A, θ_B)
HTTTHTTHTH	0.4	1	
HHHHTHHHHH	0.9	0	
HTHHHHHTHH	0.8	0	
HTTTTTHHTT	0.3	1	
THHHTHHHTH	0.7	0	

$$\theta_A = \mathit{Data} \quad * \quad \mathit{HiddenVector} \quad / \quad \mathbf{1} \quad * \quad \mathit{HiddenVector}$$

$$\theta_B = \mathit{Data} \quad * \; (\mathbf{1}\text{-}\mathit{HiddenVector}) \quad / \quad \mathbf{1} \quad * \quad (\mathbf{1}\text{-}\mathit{HiddenVector})$$

HiddenVector = (1 0 0 1 0)

What is *HiddenMatrix* corresponding to this *HiddenVector*?

From *Data* & *HiddenMatrix* to ***Parameters***

	Data	***HiddenVector***	***Parameters***=(θ_A,θ_B)
HTTT**H**TT**H**T**H**	0.4	1	
HHHHT**HHHHH**	0.9	0	
HT**HHHHH**T**HH**	0.8	0	
HTTTTT**HH**TT	0.3	1	
T**HHH**T**HHH**T**H**	0.7	0	

θ_A = *Data* * *HiddenVector* / **1** * *HiddenVector*

θ_A = *Data* * 1st row of *HiddenMatrix* / **1***1st row of *HiddenMatrix*

θ_B = *Data* * (**1**-*HiddenVector*) / **1** * (**1**-*HiddenVector*)

θ_B = *Data* * 2nd row of *HiddenMatrix* / **1***2nd row of *HiddenMatrix*

HiddenVector = (1 0 0 1 0)

$$\textit{Hidden Matrix} = \begin{array}{ccccc} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array} \begin{array}{l} = \textit{HiddenVector} \\ = \mathbf{1} - \textit{HiddenVector} \end{array}$$

From *Data* & *HiddenMatrix* to ***Parameters***

	Data	***HiddenMatrix***		***Parameters***=(θ_A,θ_B)
HTTTHTTHTH	0.4	0.97	0.03	
HHHHTHHHHH	0.9	0.12	0.88	
HTHHHHHTHH	0.8	0.29	0.71	
HTTTTTHHTT	0.3	0.99	0.01	
THHHTHHHTH	0.7	0.55	0.45	

$\theta_A = \textit{Data} * \textit{HiddenVector} \;/\; \mathbf{1} * \textit{HiddenVector}$

$\theta_A = \textit{Data} * 1^{st}\text{ row of } \textit{HiddenMatrix} \;/\; \mathbf{1} * 1^{st}\text{ row of } \textit{HiddenMatrix}$

$\theta_B = \textit{Data} * (\mathbf{1}\text{-}\textit{HiddenVector}) \;/\; \mathbf{1} * (\mathbf{1}\text{-}\textit{HiddenVector})$

$\theta_B = \textit{Data} * 2^{nd}\text{ row of } \textit{HiddenMatrix} \;/\; \mathbf{1} * 2^{nd}\text{ row of } \textit{HiddenMatrix}$

$\textit{HiddenVector} = (\,1 \quad 0 \quad 0 \quad 1 \quad 0\,)$

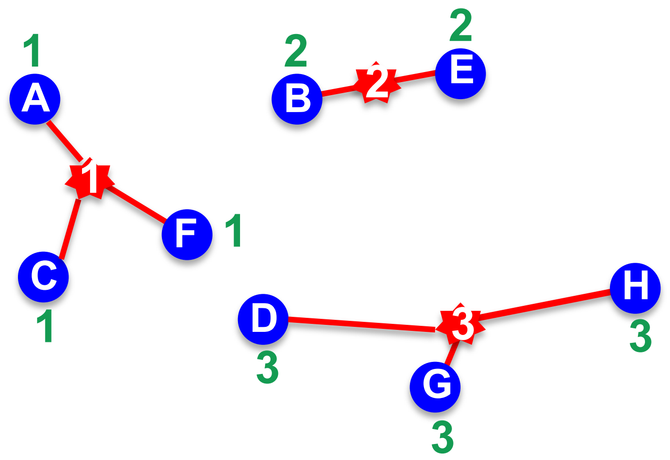
$$\textbf{\textit{Hidden Matrix}} = \begin{array}{ccccc} .97 & .03 & .29 & .99 & .55 \\ .03 & .97 & .71 & .01 & .45 \end{array}$$

From *HiddenVector* to *HiddenMatrix*

Data: data points *Data* = {*Data*$_1$, … ,*Data*$_n$}
Parameters: *Centers* = {*Center*$_1$, … ,*Center*$_k$}
HiddenVector: assignments of data points to centers

	A	B	C	D	E	F	G	H
HiddenVector	**1**	**2**	**1**	**3**	**2**	**1**	**3**	**3**

HiddenMatrix	A	B	C	D	E	F	G	H
1	**1**	0	**1**	0	0	**1**	0	0
2	0	**1**	0	0	**1**	0	0	0
3	0	0	0	**1**	0	0	**1**	**1**



From *HiddenVector* to *HiddenMatrix*

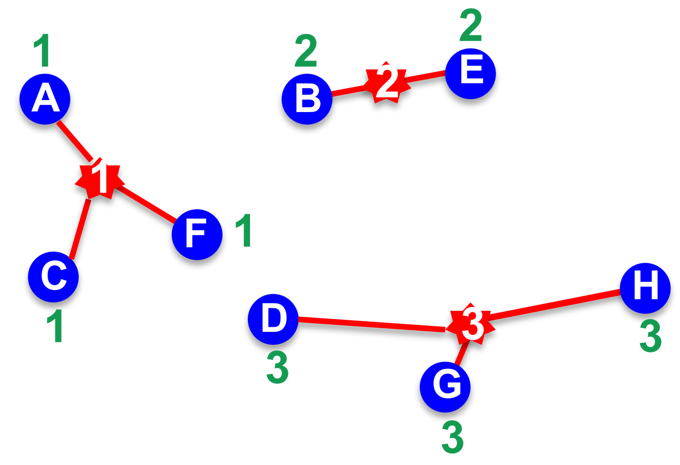
Data: data points *Data* = {$Data_1$, … ,$Data_n$}

Parameters: *Centers* = {$Center_1$, … ,$Center_k$}

HiddenMatrix$_{i,j}$: responsibility of center *i* for data point *j*

HiddenMatrix

	A	B	C	D	E	F	G	H
1	0.7	0	1	0	0	1	0	0
2	0.2	1	0	0	1	0	0	0
3	0.1	0	0	1	0	0	1	1



From *HiddenVector* to *HiddenMatrix*

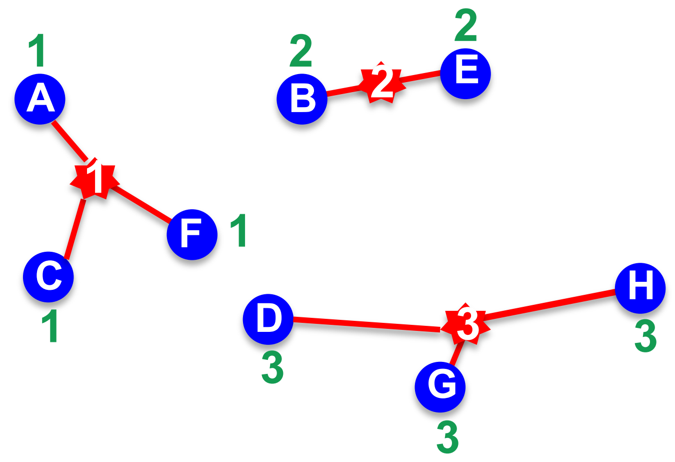
Data: data points $Data = \{Data_1, \ldots, Data_n\}$

Parameters: $Centers = \{Center_1, \ldots, Center_k\}$

$HiddenMatrix_{i,j}$: responsibility of center *i* for data point *j*

HiddenMatrix

	A	B	C	D	E	F	G	H
1	0.70	0.15	0.73	0.40	0.15	0.80	0.05	0.05
2	0.20	0.80	0.17	0.20	0.80	0.10	0.05	0.20
3	0.10	0.05	0.10	0.40	0.05	0.10	0.90	0.75



Responsibilities and the Law of Gravitation

planets

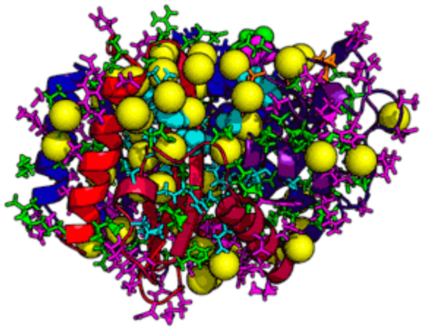
stars	0.70	0.15	0.73	0.40	0.15	0.80	0.05	0.05
	0.20	0.80	0.17	0.20	0.80	0.10	0.05	0.20
	0.10	0.05	0.10	0.40	0.05	0.10	0.90	0.75

responsibility of star *i* for a planet *j* is proportional to the pull (Newtonian law of gravitation):

$$Force_{i,j} = 1/distance(Data_j, Center_i)^2$$

$$HiddenMatrix_{ij} := Force_{i,j} \,/\, \sum_{\text{all centers } j} Force_{i,j}$$

Responsibilities and Statistical Mechanics



data points

	0.70	0.15	0.73	0.40	0.15	0.80	0.05	0.05
centers	0.20	0.80	0.17	0.20	0.80	0.10	0.05	0.20
	0.10	0.05	0.10	0.40	0.05	0.10	0.90	0.75

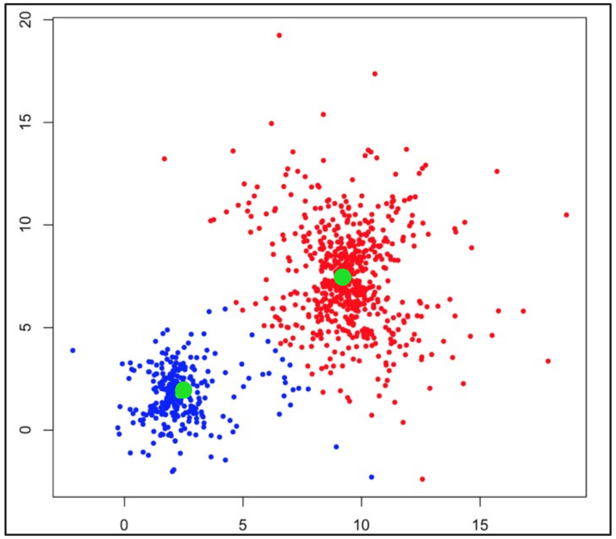
responsibility of center *i* for a data point *j* is proportional to

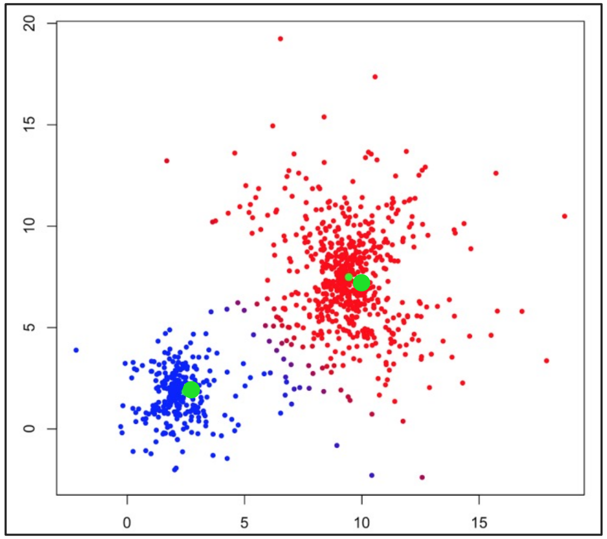
$$Force_{i,j} = e^{-\beta \cdot \text{distance}(Data\,j,\ Center\,i)}$$

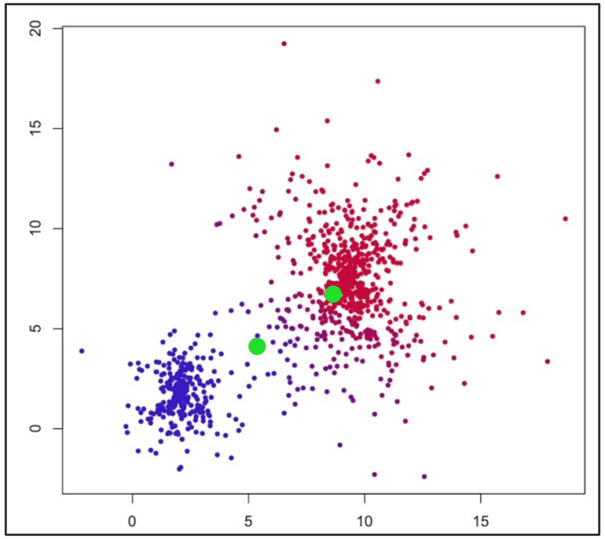
where β is a **stiffness parameter**.

$$HiddenMatrix_{ij} := Force_{i,j} \,/\, \sum_{\text{all centers } j} Force_{i,j}$$

How Does Stiffness Affect Clustering?







Hard *k*-means clustering

Soft *k*-means clustering (stiffness β=1)

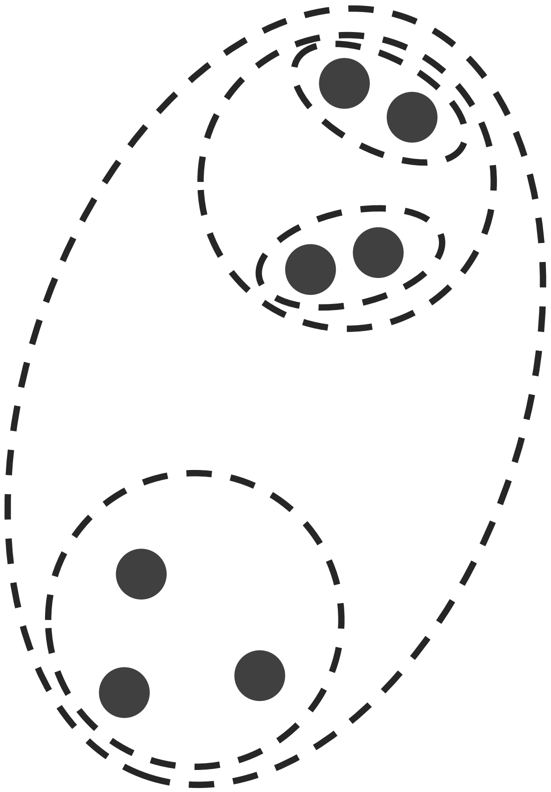
Soft *k*-means clustering (stiffness β= 0.3)

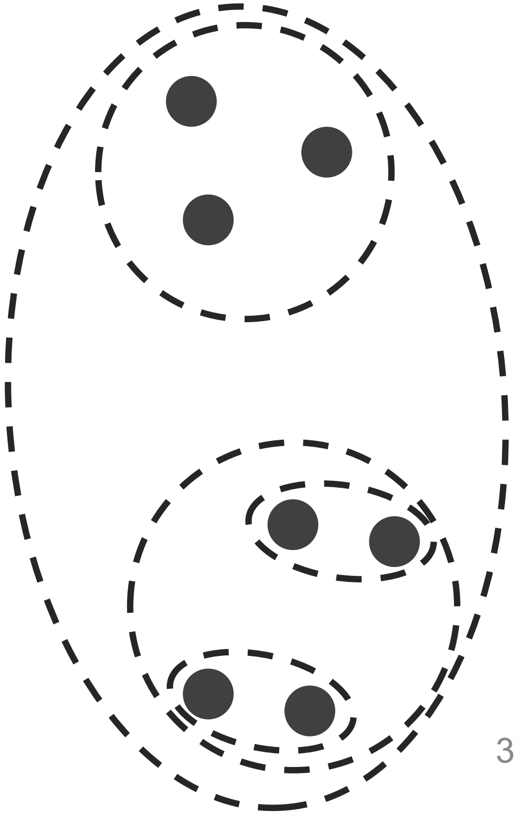
Stratification of Clusters

Clusters often have **subclusters**, which have subsubclusters, and so on.

Stratification of Clusters

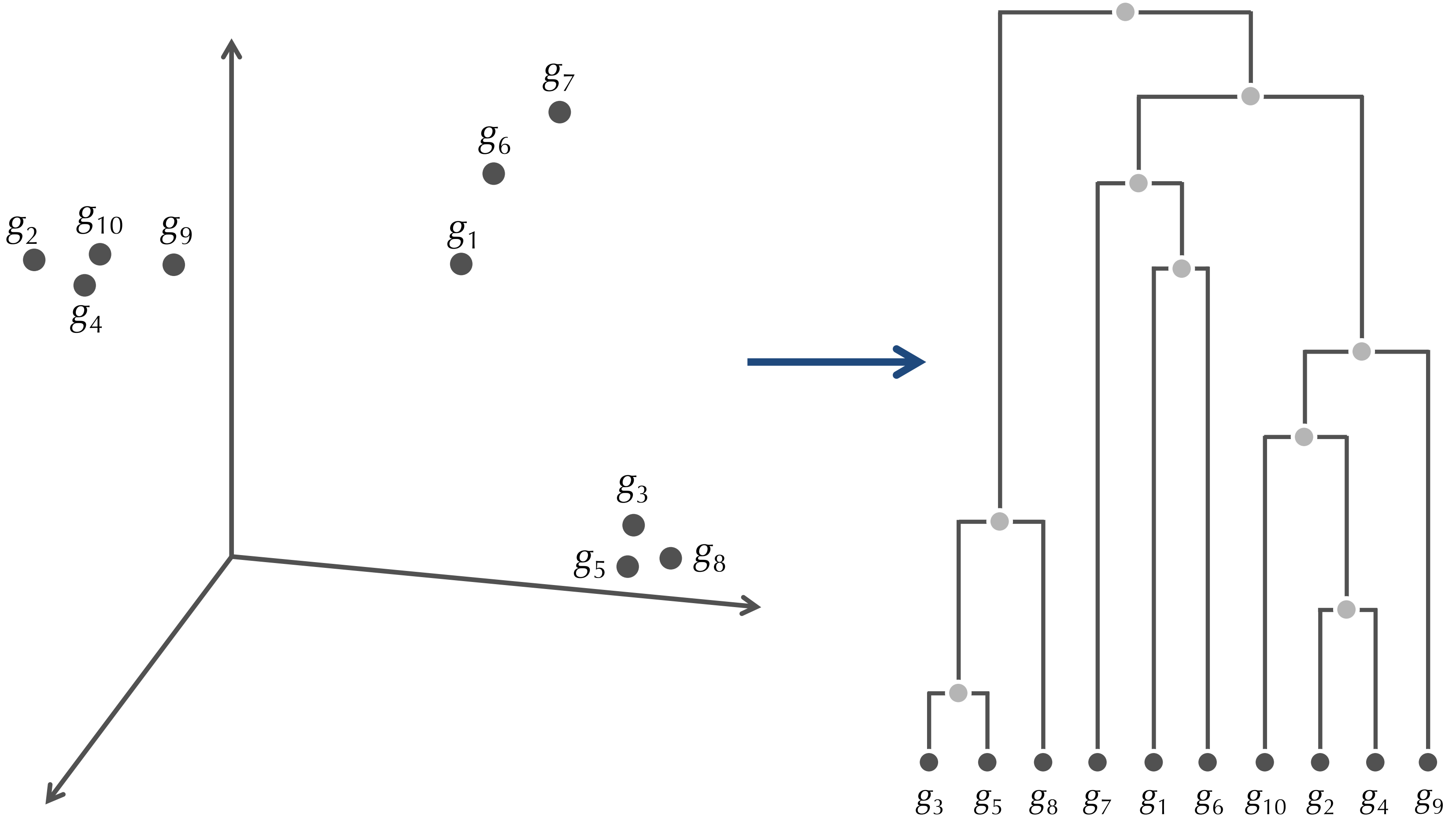
Clusters often have **subclusters**, which have sub-subclusters, and so on.





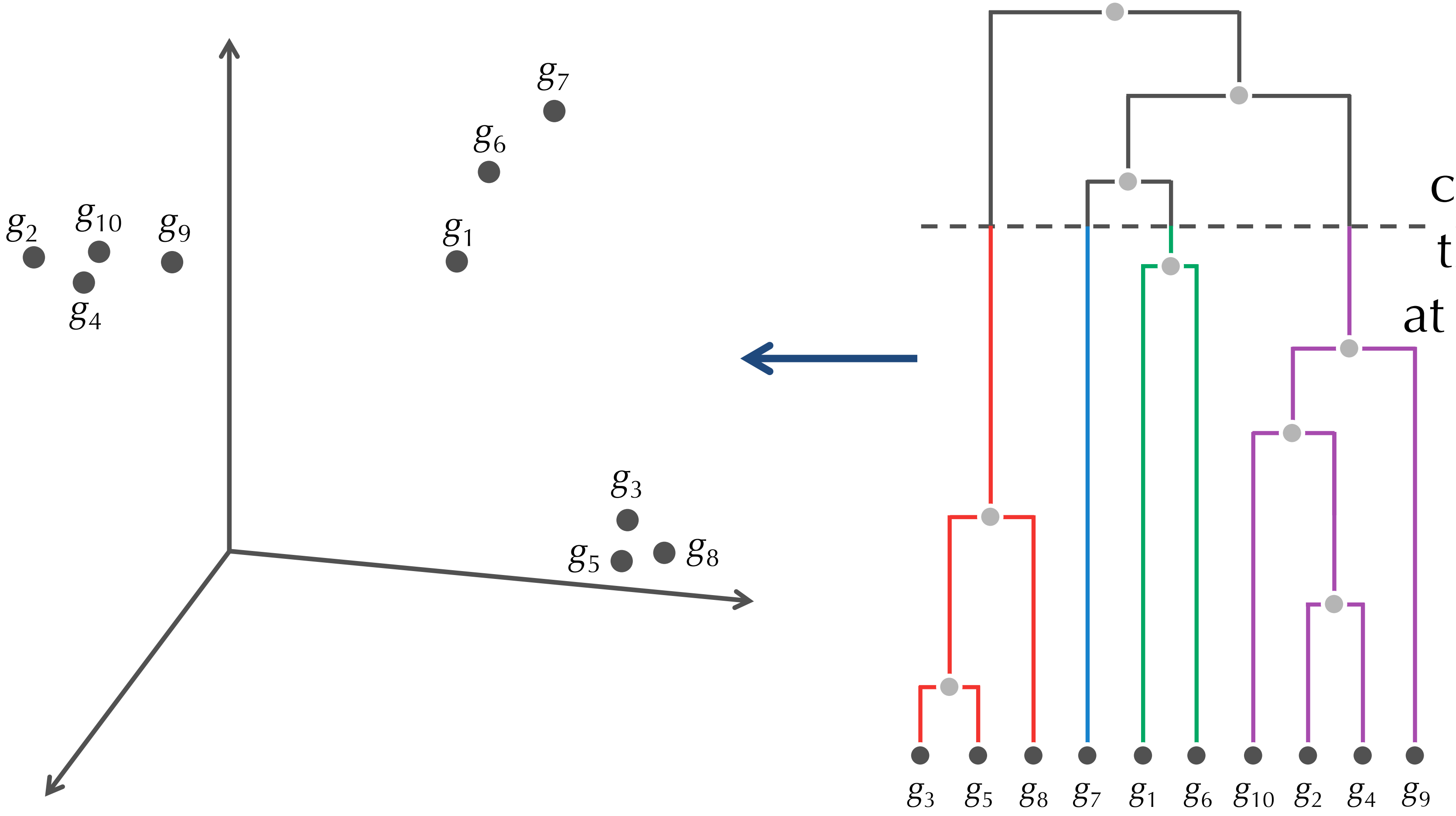
From Data to a Tree

To capture stratification, the **hierarchical clustering** algorithm organises *n* data points into a tree.


g_7
g_6
g_2
g_{10}
g_9
g_1
g_4
g_3
g_5
g_8
g_3 g_5 g_8 g_7 g_1 g_6 g_{10} g_2 g_4 g_9

From a Tree to a Partition into 4 Clusters

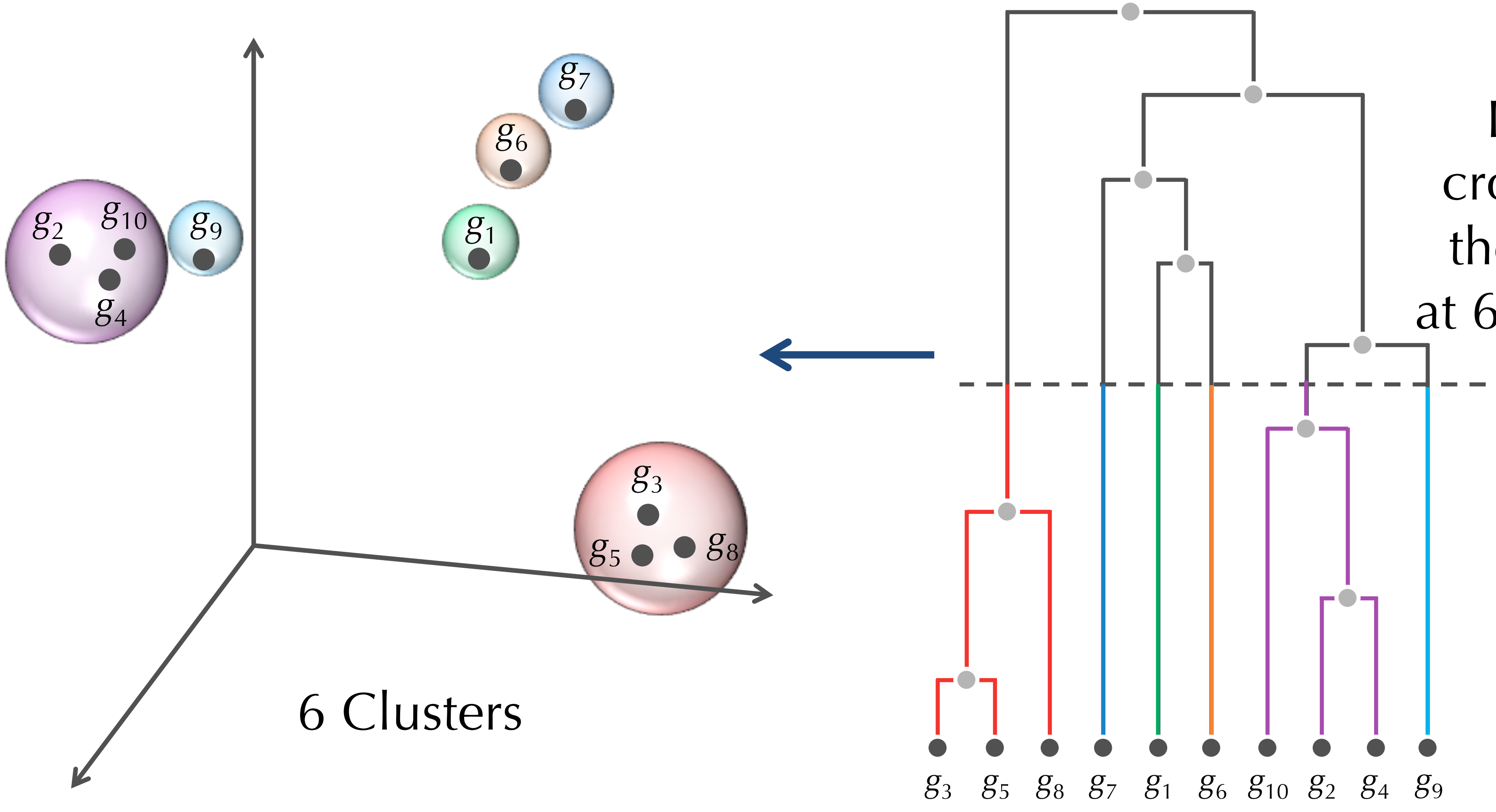
To capture stratification, the **hierarchical clustering** algorithm organises *n* data points into a tree.


g_7
g_6
g_2
g_{10}
g_9
g_1
g_4
g_3
g_5
g_8
g_3 g_5 g_8 g_7 g_1 g_6 g_{10} g_2 g_4 g_9

Line crossing the tree at 4 points

From a Tree to a Partition into 6 Clusters

To capture stratification, the **hierarchical clustering** algorithm first organises *n* data points into a tree.

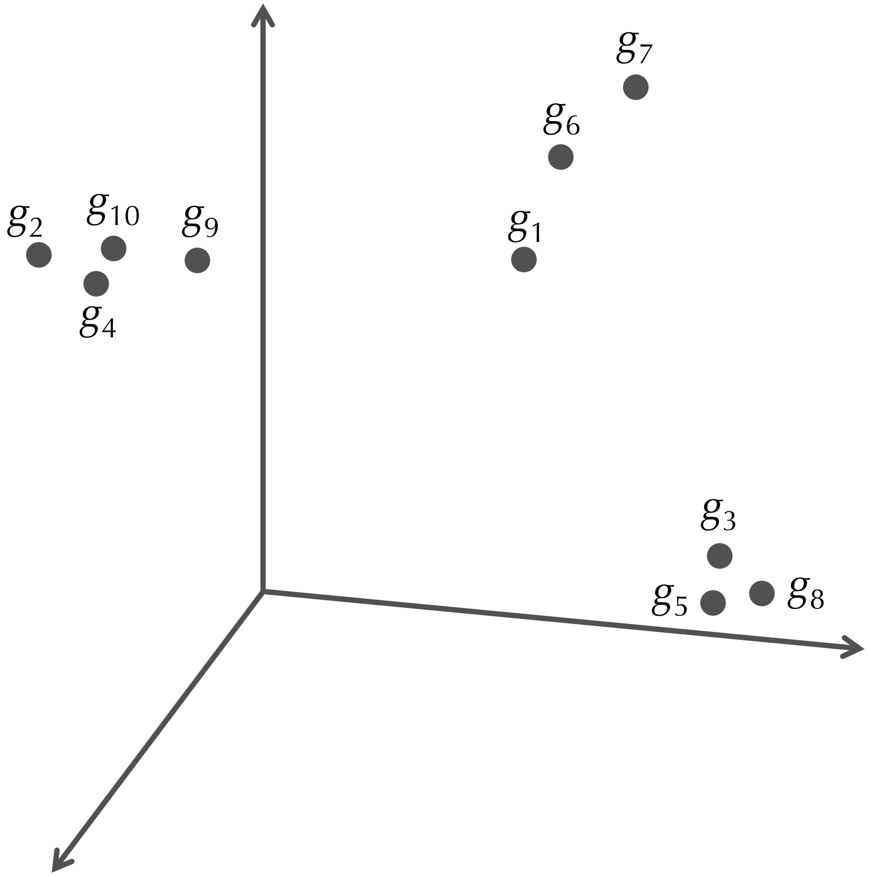

g_7
g_6
g_1
g_2
g_{10}
g_9
g_4
g_3
g_5
g_8
g_3 g_5 g_8 g_7 g_1 g_6 g_{10} g_2 g_4 g_9

6 Clusters

Line crossing the tree at 6 points

Constructing the Tree

Hierarchical clustering starts from a transformation of *n* x *m* expression matrix into *n* x *n* **similarity matrix** or **distance matrix.**

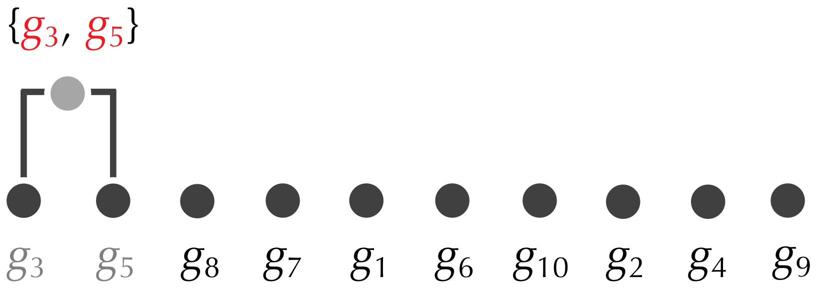


Distance Matrix

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	9.3	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	0.9	12.0	9.5	10.1	12.8	2.0	1.0
g_3	9.2	12.0	0.0	11.2	0.7	11.1	8.1	1.1	10.5	11.5
g_4	7.7	0.9	11.2	0.0	11.2	9.2	9.5	12.0	1.6	1.1
g_5	9.3	12.0	0.7	11.2	0.0	11.2	8.5	1.0	10.6	11.6
g_6	2.3	9.5	11.1	9.2	11.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	8.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.1	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	10.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	11.6	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

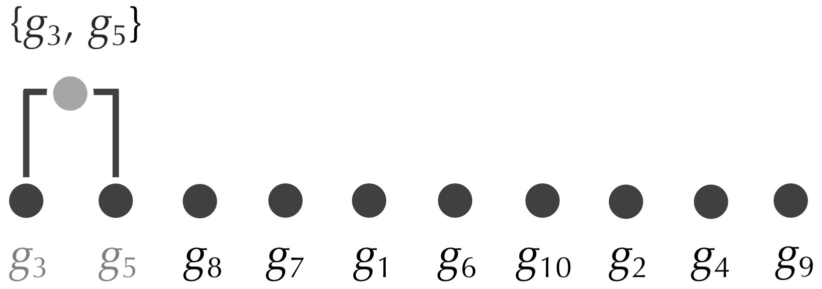
Identify the two closest clusters and merge them.



	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	9.3	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	0.9	12.0	9.5	10.1	12.8	2.0	1.0
g_3	9.2	12.0	0.0	11.2	**0.7**	11.1	8.1	1.1	10.5	11.5
g_4	7.7	0.9	11.2	0.0	11.2	9.2	9.5	12.0	1.6	1.1
g_5	9.3	12.0	0.7	11.2	0.0	11.2	8.5	1.0	10.6	11.6
g_6	2.3	9.5	11.1	9.2	11.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	8.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.1	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	10.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	11.6	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

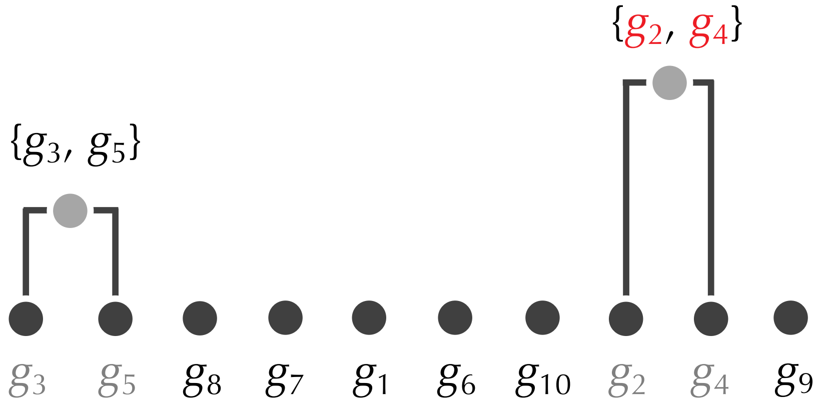
Recompute the distance between two clusters as average distance between elements in the cluster.



	g_1	g_2	g_3, g_5	g_4	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	0.9	9.5	10.1	12.8	2.0	1.0
g_3, g_5	9.2	12.0	0.0	11.2	11.1	8.1	1.0	10.5	11.5
g_4	7.7	0.9	11.2	0.0	9.2	9.5	12.0	1.6	1.1
g_6	2.3	9.5	11.1	9.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.0	12.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

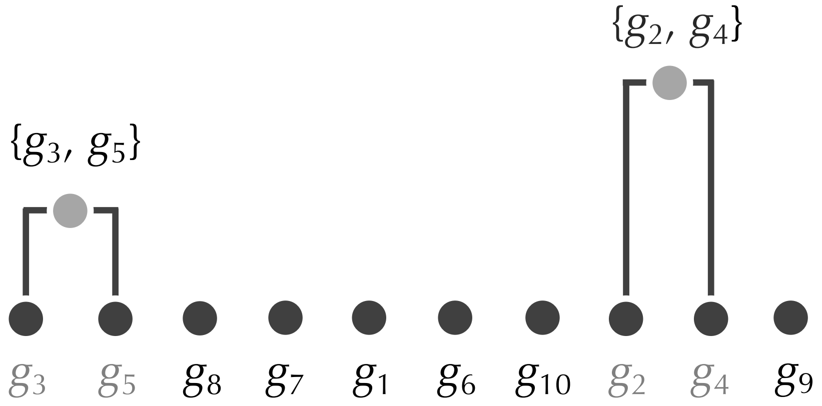
Identify the two closest clusters and merge them.



	g_1	g_2	g_3, g_5	g_4	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	**0.9**	9.5	10.1	12.8	2.0	1.0
g_3, g_5	9.2	12.0	0.0	11.2	11.1	8.1	1.0	10.5	11.5
g_4	7.7	0.9	11.2	0.0	9.2	9.5	12.0	1.6	1.1
g_6	2.3	9.5	11.1	9.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.0	12.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

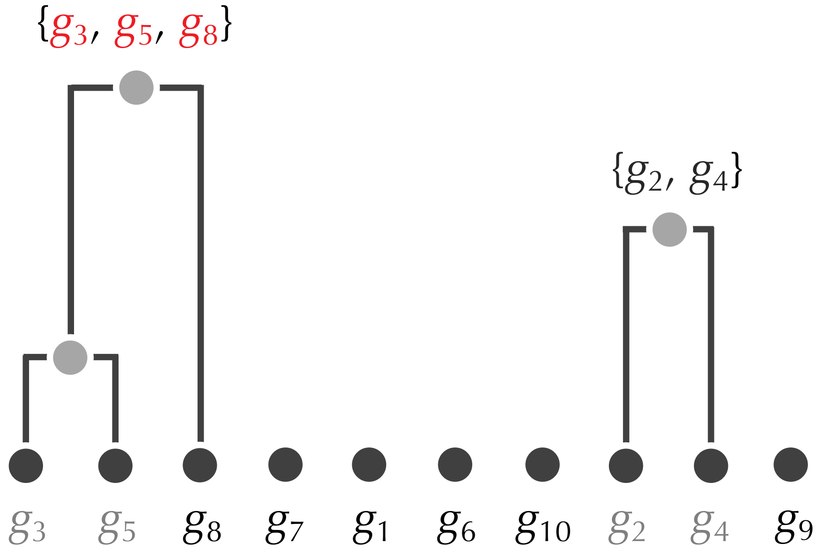
Recompute the distance between two clusters (as average distance between elements in the cluster).



	g_1	g_2, g_4	g_3, g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	7.7	9.2	2.3	5.1	10.2	6.1	7.0
g_2, g_4	7.7	0.0	11.2	9.2	9.5	12.0	1.6	1.0
g_3, g_5	9.2	11.2	0.0	11.1	8.1	1.0	10.5	11.5
g_6	2.3	9.2	11.1	0.0	5.6	12.1	7.7	8.5
g_7	5.1	9.5	8.1	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	1.6	10.5	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	8.5	9.3	12.4	1.1	0.0

Constructing the Tree

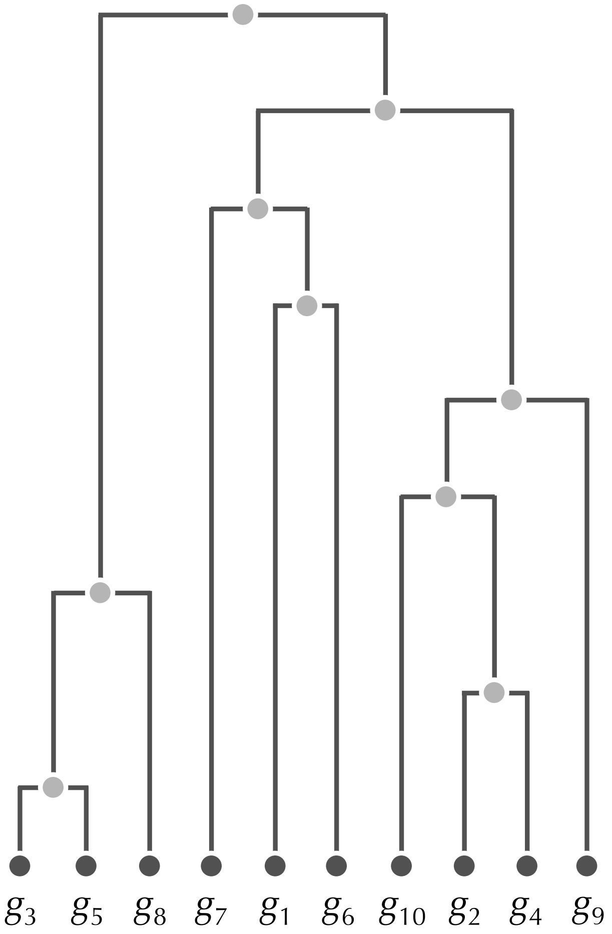
Identify the two closest clusters and merge them.



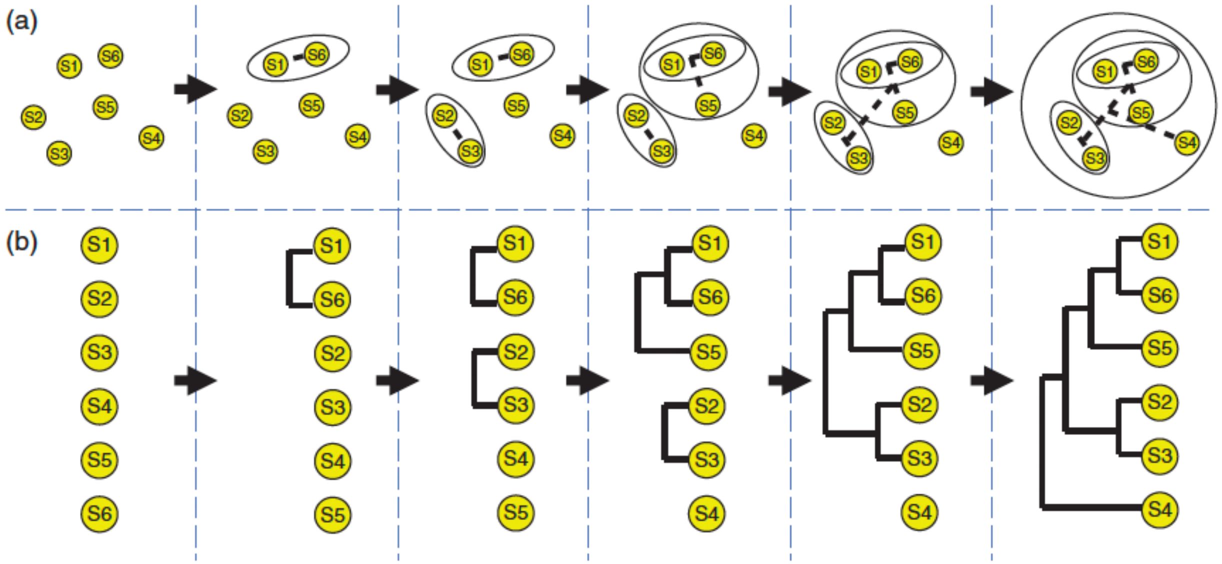
	g_1	g_2, g_4	g_3, g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	7.7	9.2	2.3	5.1	10.2	6.1	7.0
g_2, g_4	7.7	0.0	11.2	9.2	9.5	12.0	1.6	1.0
g_3, g_5	9.2	11.2	0.0	11.1	8.1	**1.0**	10.5	11.5
g_6	2.3	9.2	11.1	0.0	5.6	12.1	7.7	8.5
g_7	5.1	9.5	8.1	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	1.6	10.5	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	8.5	9.3	12.4	1.1	0.0

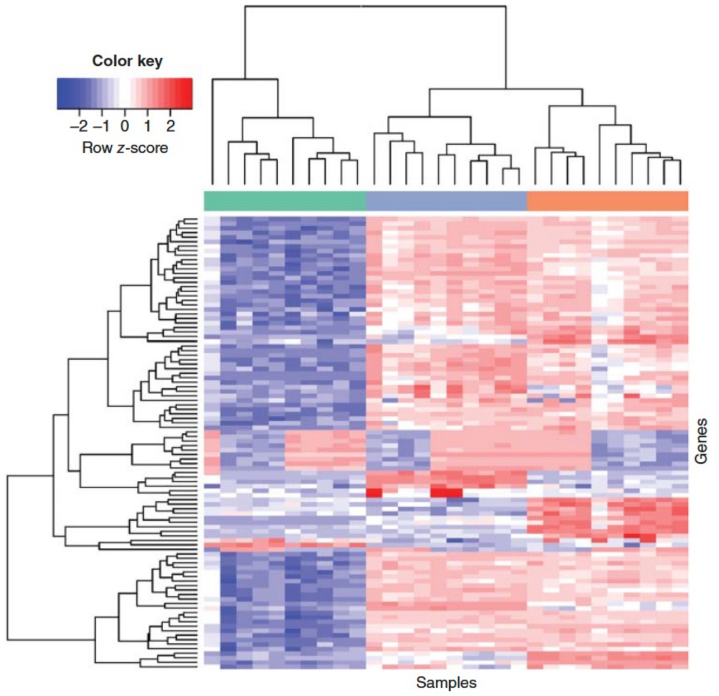
Constructing the Tree

Iterate until all elements form a single cluster (root).



hierarchical clustering: examples





Y axis: genes; x axis: conditions
(disease vs control; disease progression, etc)

Constructing a Tree from a Distance Matrix *D*

```
HierarchicalClustering ($D$, $n$)
  $Clusters$ ← $n$ single-element clusters labeled 1 to $n$
  $T$ ← a graph with the $n$ isolated nodes labeled 1 to $n$
  while there is more than one cluster
    find the two closest clusters $C_i$ and $C_j$
    merge $C_i$ and $C_j$ into a new cluster $C_{new}$ with $|C_i| + |C_j|$ elements
    add a new node labeled by cluster $C_{new}$ to $T$
    connect node $C_{new}$ to $C_i$ and $C_j$ by directed edges
    remove the rows and columns of $D$ corresponding to $C_i$ and $C_j$
    remove $C_i$ and $C_j$ from $Clusters$
    add a row and column to $D$ for the cluster $C_{new}$ by computing
      $D(C_{new}, C)$ for each cluster $C$ in $Clusters$
    add $C_{new}$ to $Clusters$
  assign root in $T$ as a node with no incoming edges
  return $T$
```

Different Distance Functions Result in Different Trees

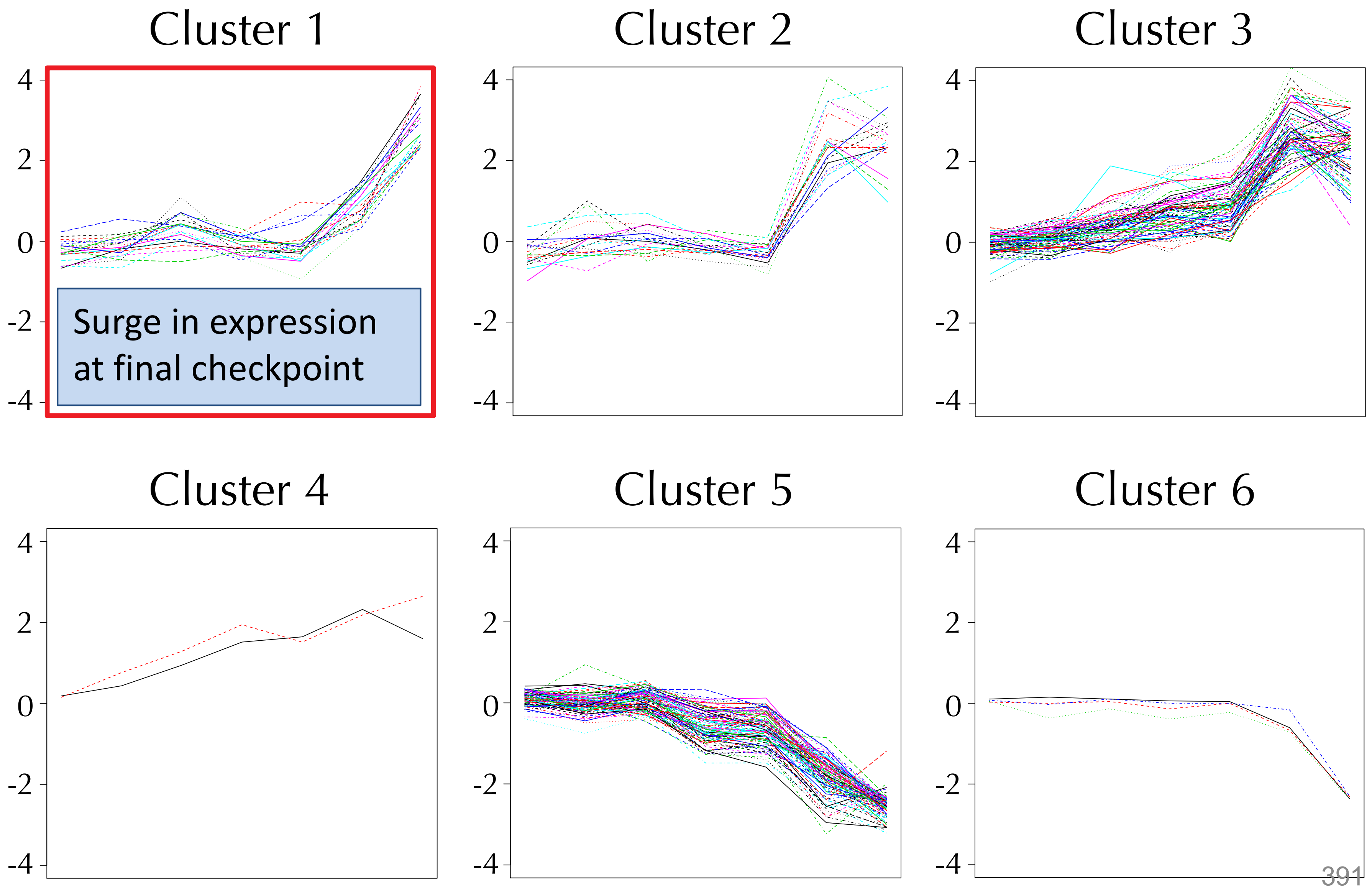
Average distance between elements of two clusters:

$$D_{\mathrm{avg}}(C_1, C_2) = (\textstyle\sum_{\text{all points } i \text{ and } j \text{ in clusters } C1 \text{ and } C2\text{, respectively}} D_{i,j})/\ (|C_1|*|C_2|)$$

Minimum distance between elements of two clusters:

$$D_{\min}(C_1, C_2) = \min_{\text{all points } i \text{ and } j \text{ in clusters } C1 \text{ and } C2\text{, respectively}} D_{i,j}$$

Clusters Constructed by **HierarchicalClustering**



Surge in expression at final checkpoint

Markov Clustering Algorithm (MCL)

MCL is unsupervised cluster algorithm for graphs derived by Stijn van Dongen during his Ph.D. (at the link below there is also his thesis).

Unlike most clustering algorithms, the MCL does not require the number of expected clusters to be specified beforehand. The basic idea underlying the algorithm is that dense clusters correspond to regions with a larger number of paths (" A random walk that visits a dense cluster will likely not leave the cluster until many of its vertices have been visited."). The algorithm works well with within a highly connected graphs. You can find the code for many programming languages at micans.org/mcl

Markov Clustering Algorithm

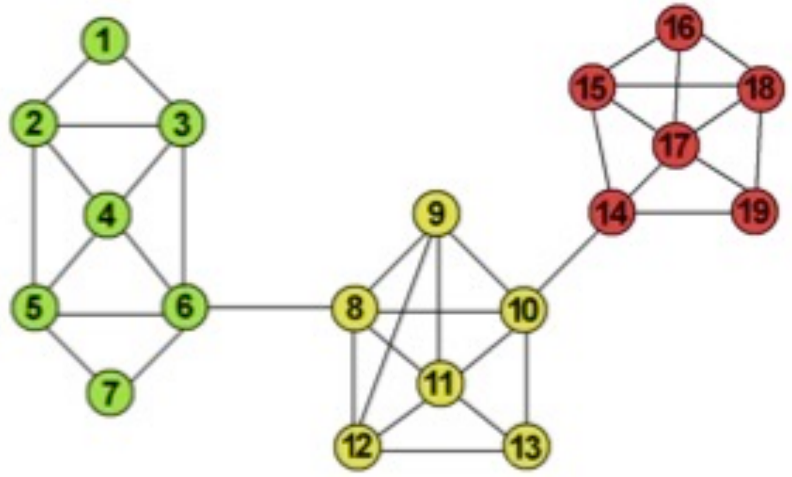
We take a random walk on the graph described by the similarity matrix, but after each step we weaken the links between distant nodes and strengthen the links between nearby nodes.

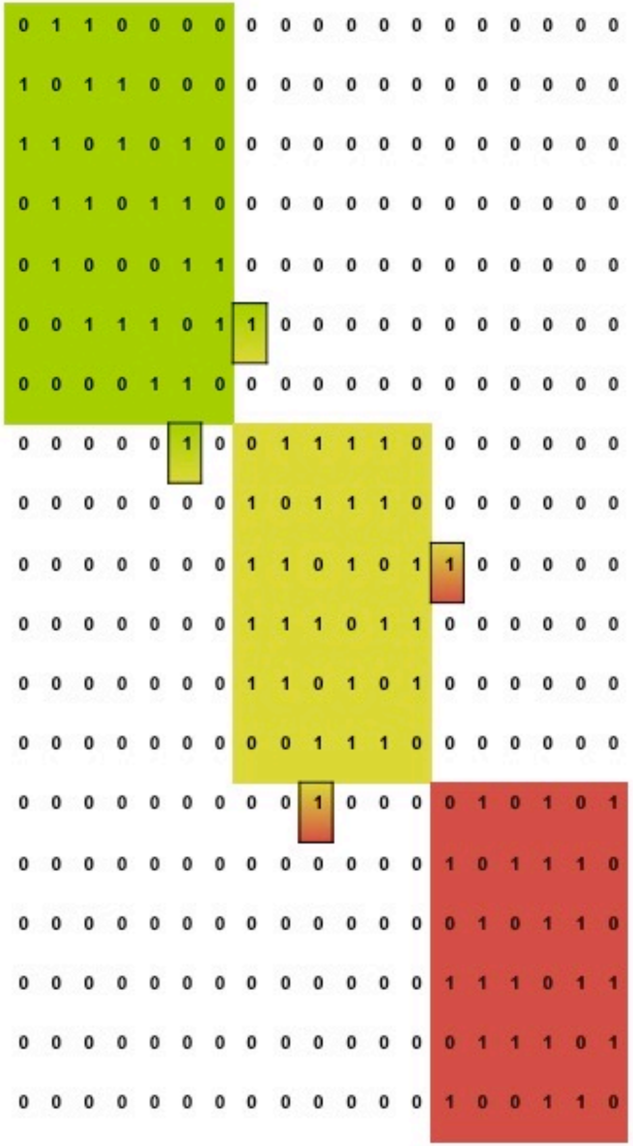
A random walk has a higher probability to stay inside the cluster than to leave it soon. **The crucial point lies in boosting this effect by an iterative alternation of expansion and inflation steps. An inflation parameter is responsible for both strengthening and weakening of current, i.e.**

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.

Markov Clustering Algorithm

Matrix representation



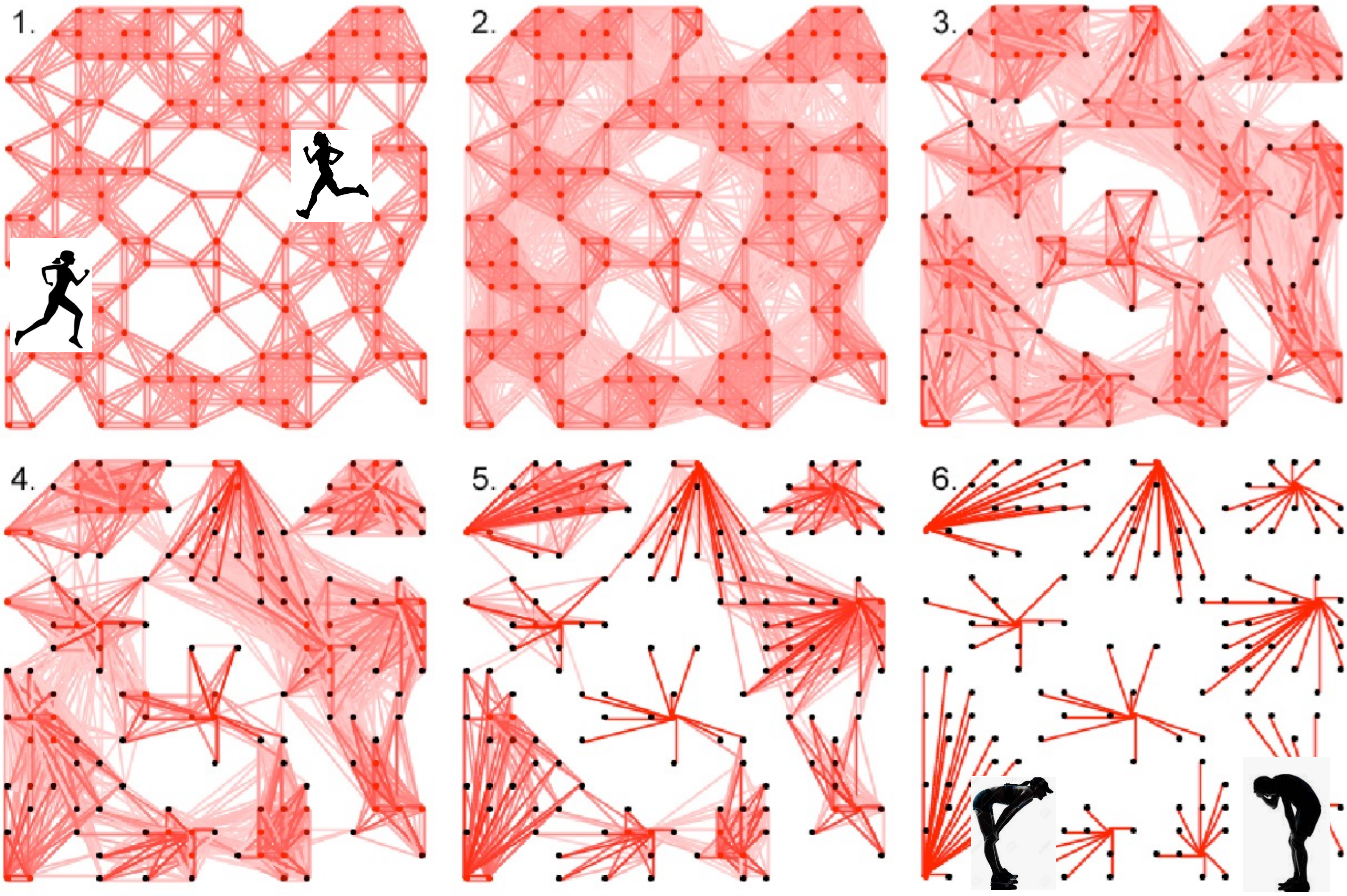


0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0

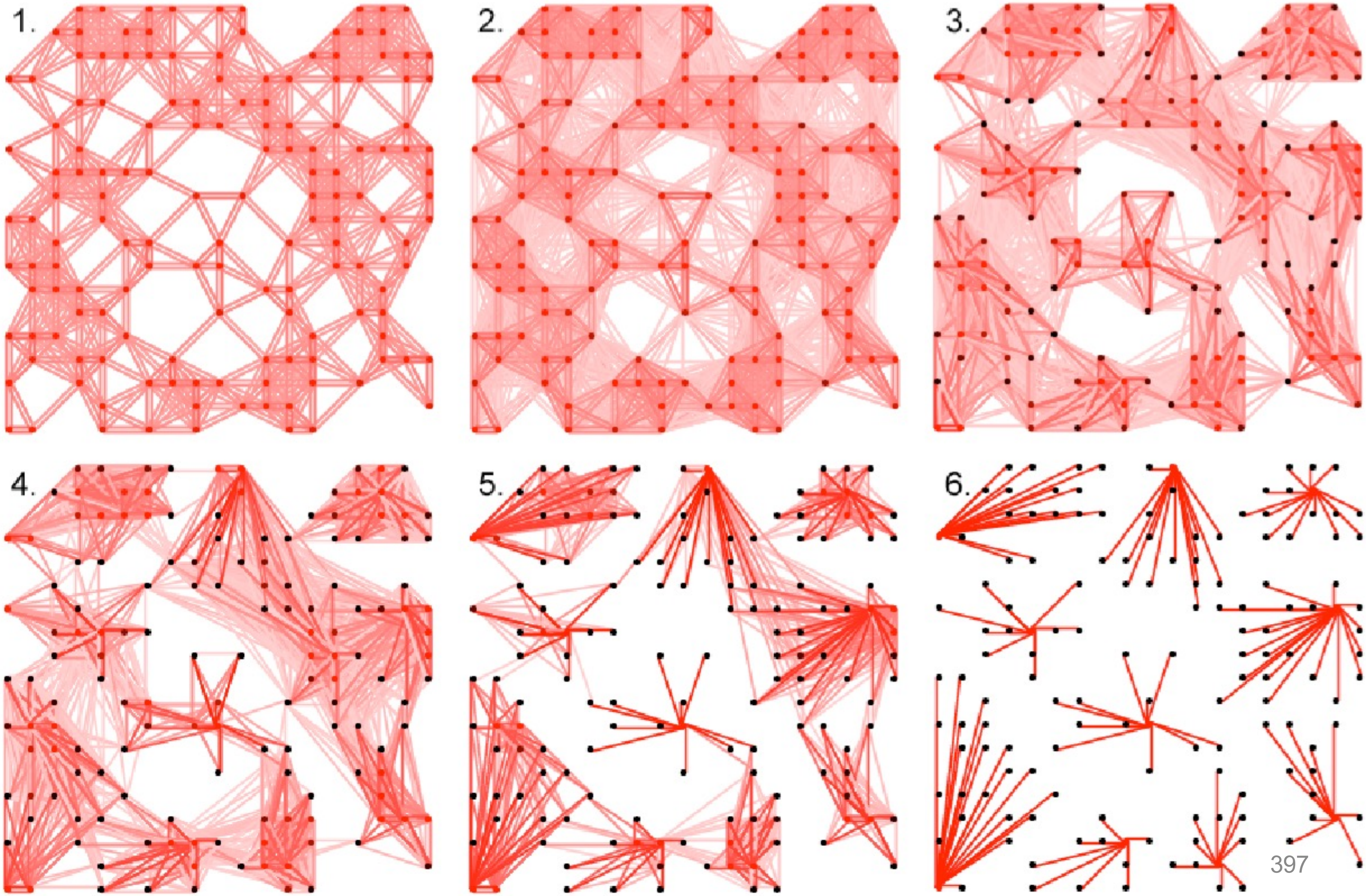
Markov Clustering Algorithm

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).

Markov Clustering Algorithm: example



Markov Clustering Algorithm: example

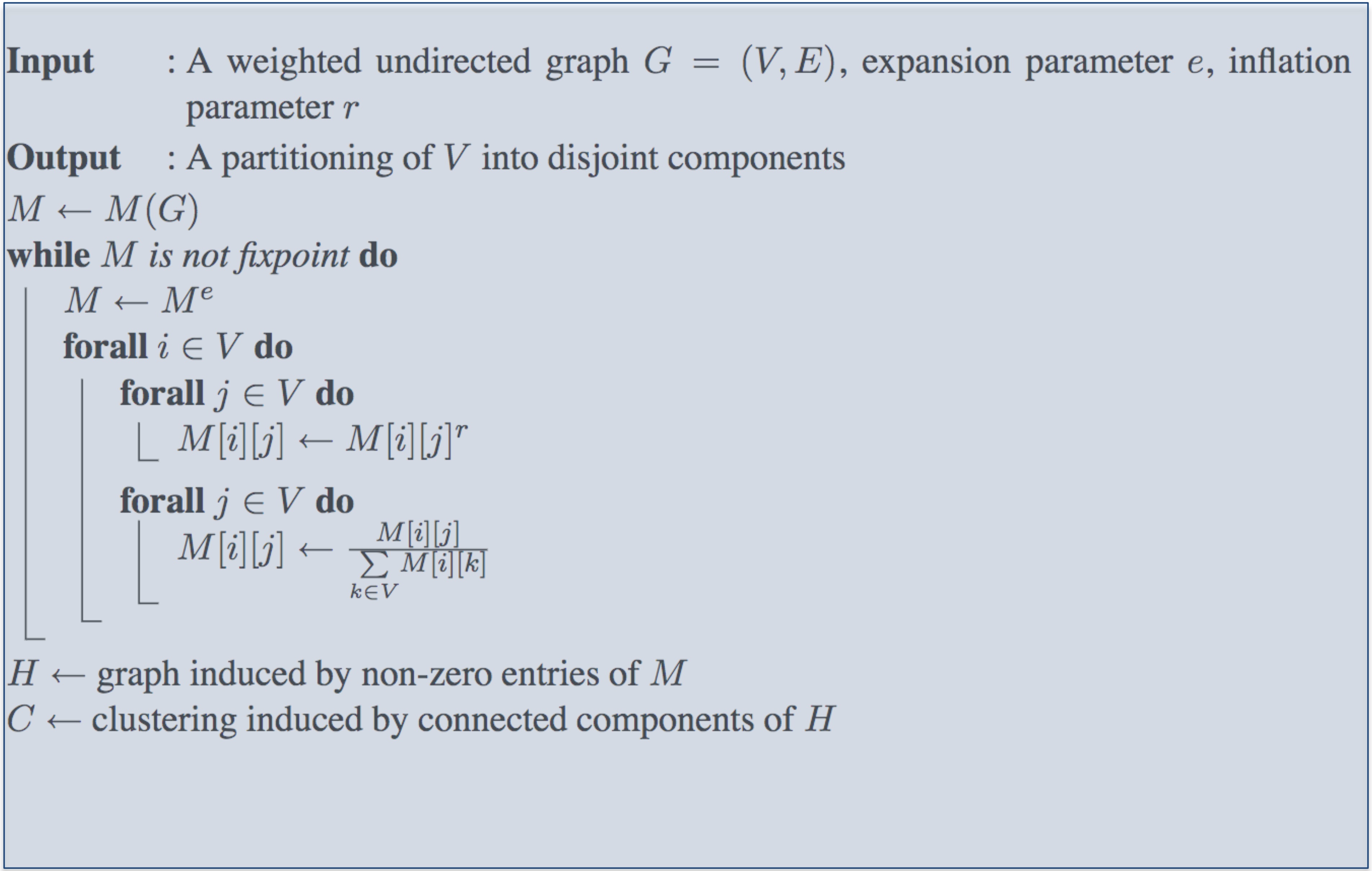


Markov Clustering Algorithm

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.

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.

Markov Clustering Algorithm



```
Input   : A weighted undirected graph $G = (V, E)$, expansion parameter $e$, inflation
          parameter $r$
Output  : A partitioning of $V$ into disjoint components
$M \leftarrow M(G)$
while $M$ is not fixpoint do
    $M \leftarrow M^e$
    forall $i \in V$ do
        forall $j \in V$ do
            $M[i][j] \leftarrow M[i][j]^r$
        forall $j \in V$ do
            $M[i][j] \leftarrow \frac{M[i][j]}{\sum_{k \in V} M[i][k]}$
$H \leftarrow$ graph induced by non-zero entries of $M$
$C \leftarrow$ clustering induced by connected components of $H$
```

Louvain Algorithm: optimising modularity

Community (communities are maximally coherent subnetworks) finding algorithm in two phases: Modularity Optimization (local moving of nodes) and Community Aggregation. In the local moving phase, individual nodes are moved to the community that yields the largest increase in the quality function. In the aggregation phase, an aggregate network is created based on the partition obtained in the local moving phase. Each community in this partition becomes a node in the aggregate network. After the first step is completed, the second follows. Both will be executed until there are no more changes in the network and maximum modularity is achieved.

Louvain Algorithm: optimising modularity.

The modularity of a partition is a scalar value between −1 and 1 that measures the density of links inside communities as compared to links between Communities and is an objective function to optimise :

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j),$$

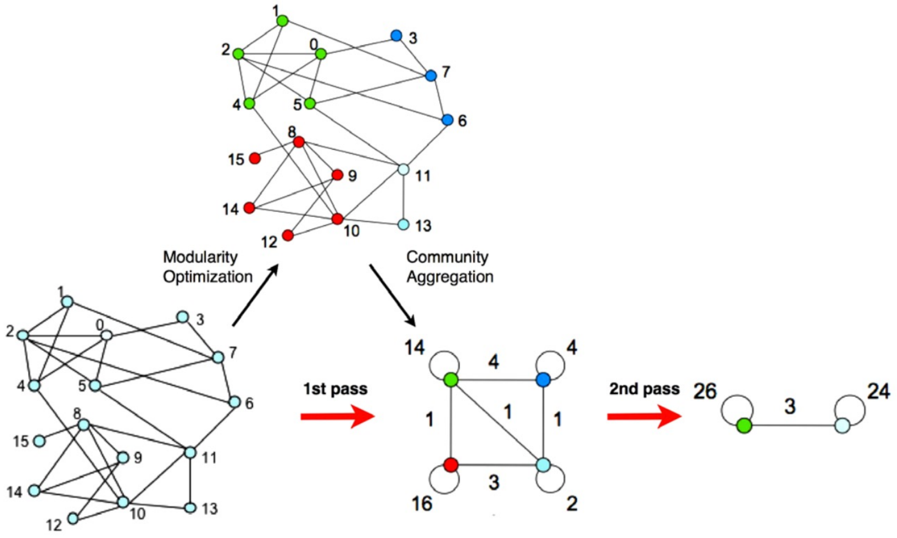
where A_{ij} represents the weight of the edge between i and j, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges attached to vertex i, c_i is the community to which vertex i is assigned, the δ function δ(u, v) is 1 if u = v and 0 otherwise and m = $\frac{1}{2}\sum_{ij} A_{ij}$.

Q>0 when the #edges within groups > # edges within groups in arandomly rewired graph.

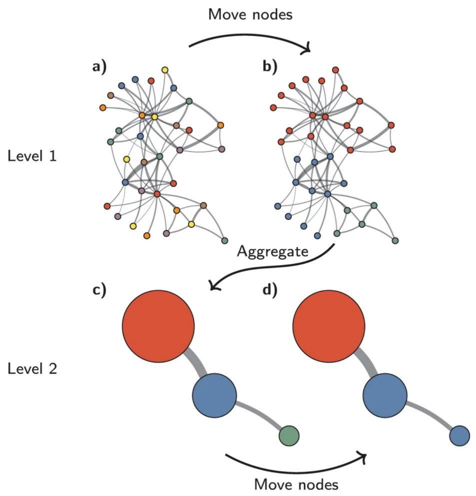
Exact modularity optimisation is a problem that is computationally hard and so approximation algorithms are necessary when dealing with large networks.

Vincent D Blondel et al J. Stat. Mech. (2008) P10008 /all authors from Lovain, Belgium

Louvain Algorithm: step 1



Each pass is made of two phases: (1) modularity is optimised by allowing only local changes of communities; (2) the communities found are aggregated in order to build a new network of communities. The passes are repeated iteratively until no increase of modularity is possible (from https://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008/pdf).



The Louvain algorithm starts from a singleton partition in which each node is in its own community (a). The algorithm moves individual nodes from one community to another to find a partition (b). Based on this partition, an aggregate network is created (c). The algorithm then moves individual nodes in the aggregate network (d). These steps are repeated until the quality cannot be increased further.

Louvain Algorithm: step 1

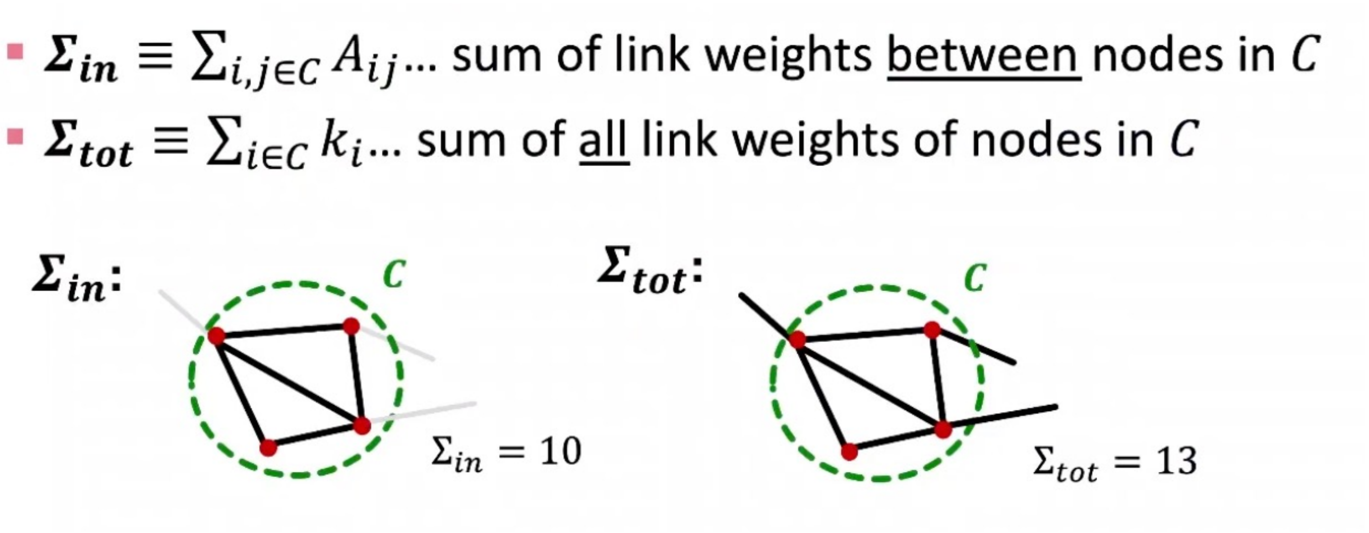
The gain in modularity ΔQ obtained by moving an isolated node i into a community C can easily be computed by

$$\Delta Q = \left[\frac{\sum_{in} + 2k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

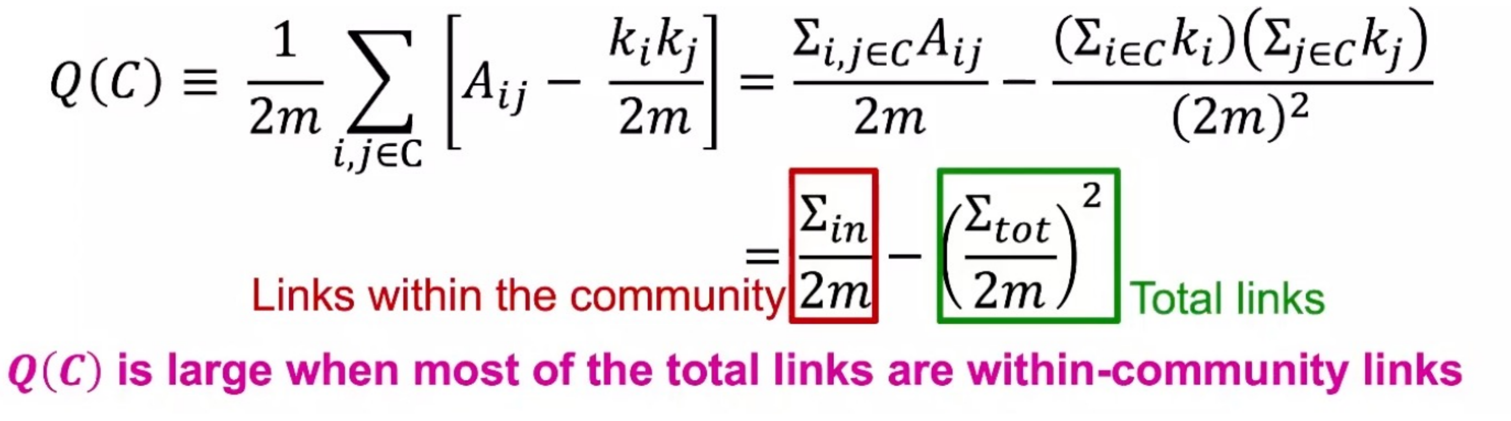
Where $\sum_{in}$ is the sum of the weights of the links inside C, $\sum_{tot}$ is the sum of the weights of the links incident to nodes in C, k_i is the sum of the weights of the links incident to node i, $k_{i,in}$ is the sum of the weights of the links from i to nodes in C and m is the sum of the weights of all the links in the network.

A similar expression is used in order to evaluate the change of modularity when i is removed from its community. In practice, one therefore evaluates the change of modularity by removing i from its community and then by moving it into a neighbouring community.

Louvain Algorithm: step 1, details



- $\boldsymbol{\Sigma_{in}} \equiv \sum_{i,j \in C} A_{ij}$... sum of link weights between nodes in C
- $\boldsymbol{\Sigma_{tot}} \equiv \sum_{i \in C} k_i$... sum of all link weights of nodes in C



$$Q(C) \equiv \frac{1}{2m} \sum_{i,j \in C} \left[A_{ij} - \frac{k_i k_j}{2m} \right] = \frac{\sum_{i,j \in C} A_{ij}}{2m} - \frac{\left(\sum_{i \in C} k_i\right)\left(\sum_{j \in C} k_j\right)}{(2m)^2}$$

$$= \boxed{\frac{\Sigma_{in}}{2m}} - \boxed{\left(\frac{\Sigma_{tot}}{2m}\right)^2}$$

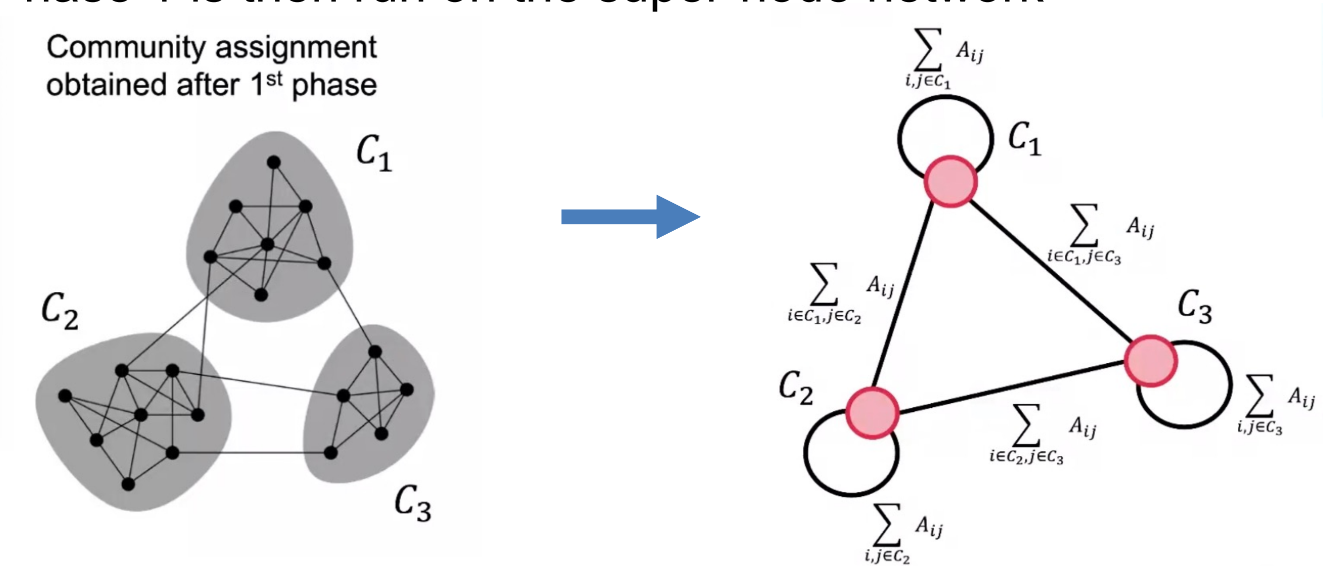
Links within the community

Total links

$Q(C)$ is large when most of the total links are within-community links

Louvain Algorithm: step 2, details

Super-nodes are obtained by merging nodes in the same community.
The communities obtained in the first step are contracted into super-nodes and the network is created accordingly:
Super-nodes are connected if there is at least one edge between the nodes of the different communities.
The weight of the edge between the two super nodes is the sum of the weights from all the edges between their corresponding communities.
Phase 1 is then run on the super-node network



Community assignment obtained after 1st phase
C_1
C_2
C_3
$\sum_{i,j \in C_1} A_{ij}$
C_1
$\sum_{i \in C_1, j \in C_3} A_{ij}$
$\sum_{i \in C_1, j \in C_2} A_{ij}$
C_3
C_2
$\sum_{i,j \in C_3} A_{ij}$
$\sum_{i \in C_2, j \in C_3} A_{ij}$
$\sum_{i,j \in C_2} A_{ij}$

https://www.youtube.com/watch?v=0zuiLBOIcsw&t=214s

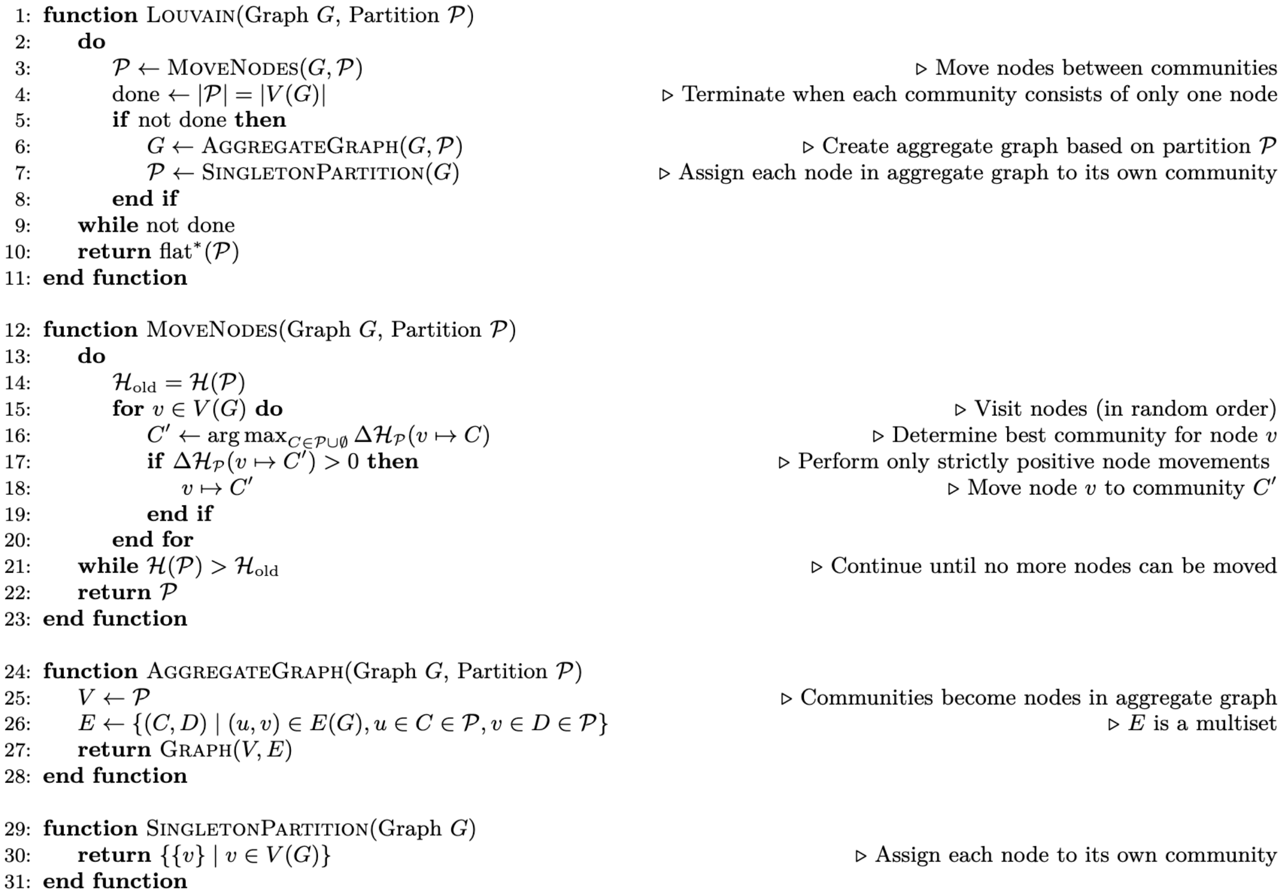
Louvain Algorithm: steps 1,2

- Start with every node in its own community
- Phase 1: Modularity optimization
  - Order the nodes and do the following for each I
    - Move i to the community of neighbor j that leads to maximus ΔQ.
    - If all $\Delta Q<0$ then I remains in its current community
  - Repeatedly cycle through all node until $\Delta Q=0$
- Phase 2: Community aggregation
  - Create a weighted network of communities from Phase 1
    - Nodes= communities from Phase 1
    - Edge weights = sum of weights of edges between communities
    - Edges within a community become 2 self-loops
- Repeat: apply phase 1/phase 2 to resulting network, and so until $\Delta Q=0$

Louvain Algorithm: limitations

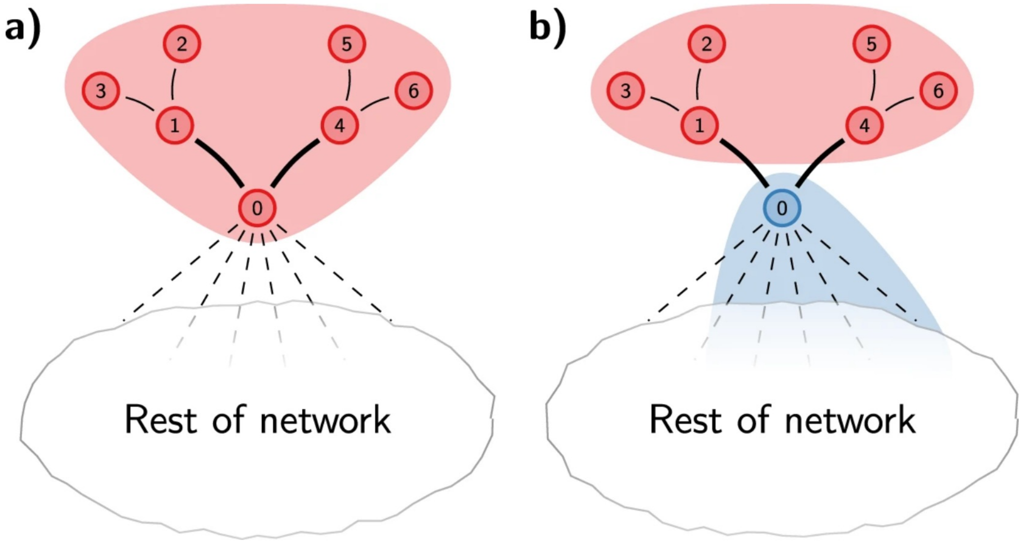
- Modularity has a resolution limit:
  - Cannot identify community of size O($m^{1/2}$)
  - Due to the normalizing factor 1/(2m) which means that modularity depends on the size of the network, not just local properties.
- The Louvain method is an approximation algorithm due to randomness and resolution limit.

Lovain algorithm

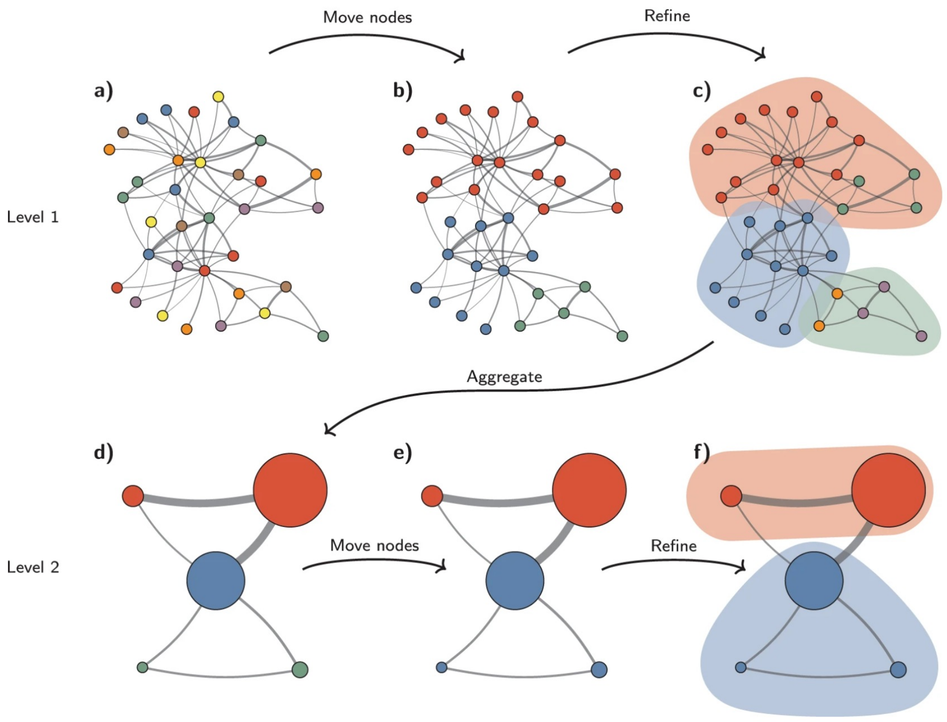


```
function LOUVAIN(Graph $G$, Partition $\mathcal{P}$)
    do
        $\mathcal{P} \leftarrow$ MOVENODES($G, \mathcal{P}$)    ▷ Move nodes between communities
        done $\leftarrow |\mathcal{P}| = |V(G)|$    ▷ Terminate when each community consists of only one node
        if not done then
            $G \leftarrow$ AGGREGATEGRAPH($G, \mathcal{P}$)    ▷ Create aggregate graph based on partition $\mathcal{P}$
            $\mathcal{P} \leftarrow$ SINGLETONPARTITION($G$)    ▷ Assign each node in aggregate graph to its own community
        end if
    while not done
    return flat$^*(\mathcal{P})$
end function

function MOVENODES(Graph $G$, Partition $\mathcal{P}$)
    do
        $\mathcal{H}_{\text{old}} = \mathcal{H}(\mathcal{P})$
        for $v \in V(G)$ do    ▷ Visit nodes (in random order)
            $C' \leftarrow \arg\max_{C \in \mathcal{P} \cup \emptyset} \Delta\mathcal{H}_{\mathcal{P}}(v \mapsto C)$    ▷ Determine best community for node $v$
            if $\Delta\mathcal{H}_{\mathcal{P}}(v \mapsto C') > 0$ then    ▷ Perform only strictly positive node movements
                $v \mapsto C'$    ▷ Move node $v$ to community $C'$
            end if
        end for
    while $\mathcal{H}(\mathcal{P}) > \mathcal{H}_{\text{old}}$    ▷ Continue until no more nodes can be moved
    return $\mathcal{P}$
end function

function AGGREGATEGRAPH(Graph $G$, Partition $\mathcal{P}$)
    $V \leftarrow \mathcal{P}$    ▷ Communities become nodes in aggregate graph
    $E \leftarrow \{(C, D) \mid (u, v) \in E(G), u \in C \in \mathcal{P}, v \in D \in \mathcal{P}\}$    ▷ $E$ is a multiset
    return GRAPH($V, E$)
end function

function SINGLETONPARTITION(Graph $G$)
    return $\{\{v\} \mid v \in V(G)\}$    ▷ Assign each node to its own community
end function
```

From Louvain to Leiden clustering (Non examinable)



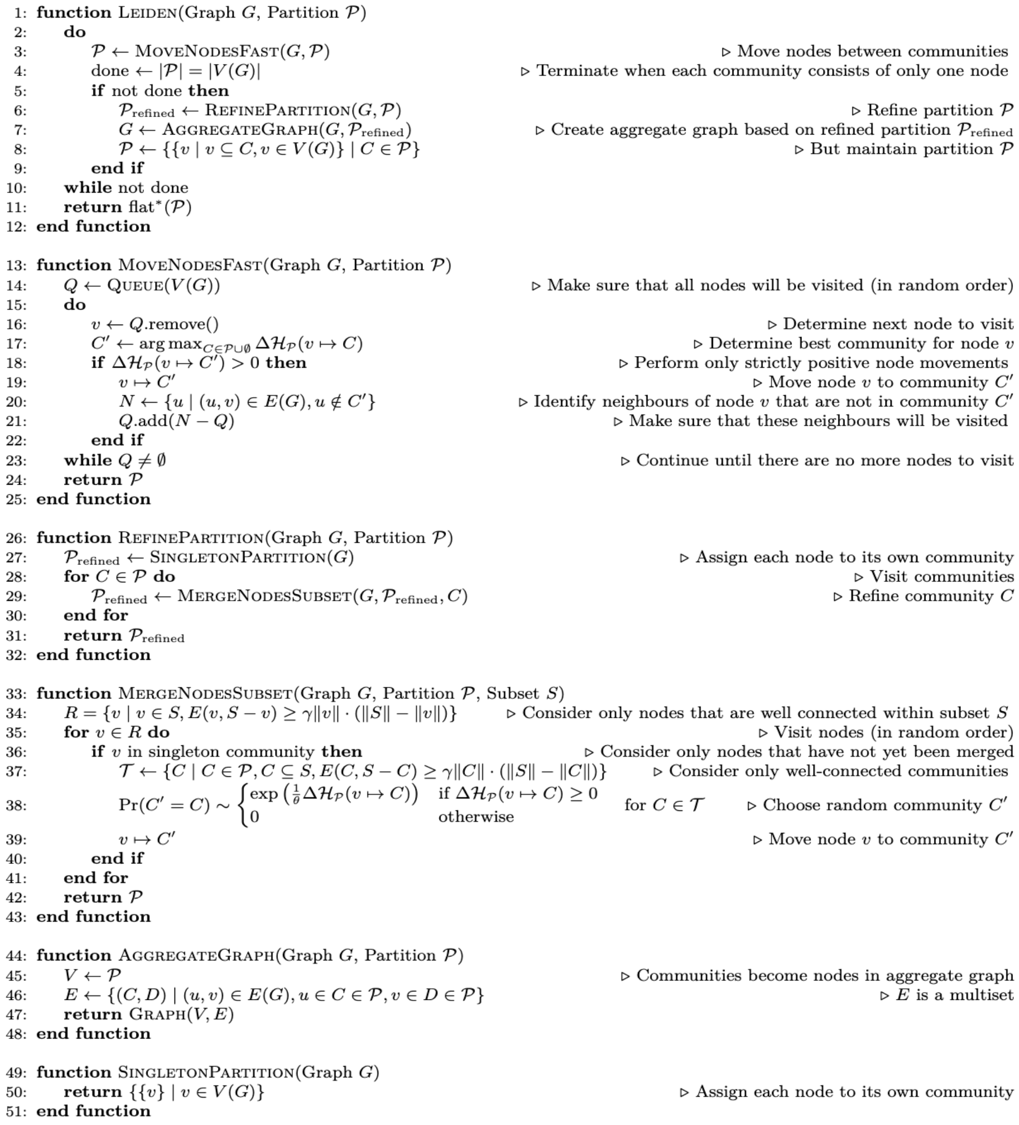
Disconnected community. Consider the partition shown in (**a**). When node 0 is moved to a different community, the red community becomes internally disconnected, as shown in (**b**). However, nodes 1–6 are still locally optimally assigned, and therefore these nodes will stay in the red community.



Leiden algorithm

The Leiden algorithm starts from a singleton partition (a). The algorithm moves individual nodes from one community to another to find a partition (b), which is then refined (c).

An aggregate network (d) is created based on the refined partition, using the non-refined partition to create an initial partition for the aggregate network. For example, the red community in (b) is refined into two subcommunities in (c), which after aggregation become two separate nodes in (d), both belonging to the same community. The algorithm then moves individual nodes in the aggregate network (e). In this case, refinement does not change the partition (f). These steps are repeated until no further improvements can be made.

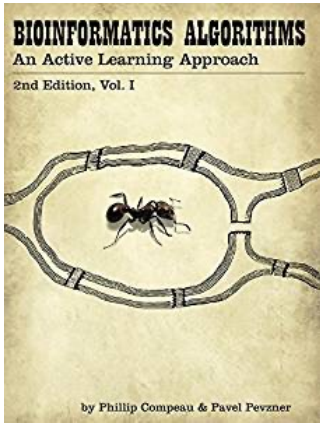


```
function Leiden(Graph $G$, Partition $\mathcal{P}$)
    do
        $\mathcal{P} \leftarrow$ MoveNodesFast($G, \mathcal{P}$)    ▷ Move nodes between communities
        done $\leftarrow |\mathcal{P}| = |V(G)|$    ▷ Terminate when each community consists of only one node
        if not done then
            $\mathcal{P}_{\text{refined}} \leftarrow$ RefinePartition($G, \mathcal{P}$)    ▷ Refine partition $\mathcal{P}$
            $G \leftarrow$ AggregateGraph($G, \mathcal{P}_{\text{refined}}$)    ▷ Create aggregate graph based on refined partition $\mathcal{P}_{\text{refined}}$
            $\mathcal{P} \leftarrow \{\{v \mid v \subseteq C, v \in V(G)\} \mid C \in \mathcal{P}\}$    ▷ But maintain partition $\mathcal{P}$
        end if
    while not done
    return flat$^*(\mathcal{P})$
end function

function MoveNodesFast(Graph $G$, Partition $\mathcal{P}$)
    $Q \leftarrow$ Queue($V(G)$)    ▷ Make sure that all nodes will be visited (in random order)
    do
        $v \leftarrow Q$.remove()    ▷ Determine next node to visit
        $C' \leftarrow \arg\max_{C \in \mathcal{P} \cup \emptyset} \Delta\mathcal{H}_{\mathcal{P}}(v \mapsto C)$    ▷ Determine best community for node $v$
        if $\Delta\mathcal{H}_{\mathcal{P}}(v \mapsto C') > 0$ then    ▷ Perform only strictly positive node movements
            $v \mapsto C'$    ▷ Move node $v$ to community $C'$
            $N \leftarrow \{u \mid (u, v) \in E(G), u \notin C'\}$    ▷ Identify neighbours of node $v$ that are not in community $C'$
            $Q$.add($N - Q$)    ▷ Make sure that these neighbours will be visited
        end if
    while $Q \neq \emptyset$    ▷ Continue until there are no more nodes to visit
    return $\mathcal{P}$
end function

function RefinePartition(Graph $G$, Partition $\mathcal{P}$)
    $\mathcal{P}_{\text{refined}} \leftarrow$ SingletonPartition($G$)    ▷ Assign each node to its own community
    for $C \in \mathcal{P}$ do    ▷ Visit communities
        $\mathcal{P}_{\text{refined}} \leftarrow$ MergeNodesSubset($G, \mathcal{P}_{\text{refined}}, C$)    ▷ Refine community $C$
    end for
    return $\mathcal{P}_{\text{refined}}$
end function

function MergeNodesSubset(Graph $G$, Partition $\mathcal{P}$, Subset $S$)
    $R = \{v \mid v \in S, E(v, S - v) \geq \gamma\|v\| \cdot (\|S\| - \|v\|)\}$    ▷ Consider only nodes that are well connected within subset $S$
    for $v \in R$ do    ▷ Visit nodes (in random order)
        if $v$ in singleton community then    ▷ Consider only nodes that have not yet been merged
            $\mathcal{T} \leftarrow \{C \mid C \in \mathcal{P}, C \subseteq S, E(C, S - C) \geq \gamma\|C\| \cdot (\|S\| - \|C\|)\}$    ▷ Consider only well-connected communities
            $\Pr(C' = C) \sim \begin{cases} \exp\left(\frac{1}{\theta}\Delta\mathcal{H}_{\mathcal{P}}(v \mapsto C)\right) & \text{if } \Delta\mathcal{H}_{\mathcal{P}}(v \mapsto C) \geq 0 \\ 0 & \text{otherwise} \end{cases}$ for $C \in \mathcal{T}$    ▷ Choose random community $C'$
            $v \mapsto C'$    ▷ Move node $v$ to community $C'$
        end if
    end for
    return $\mathcal{P}$
end function

function AggregateGraph(Graph $G$, Partition $\mathcal{P}$)
    $V \leftarrow \mathcal{P}$    ▷ Communities become nodes in aggregate graph
    $E \leftarrow \{(C, D) \mid (u, v) \in E(G), u \in C \in \mathcal{P}, v \in D \in \mathcal{P}\}$    ▷ $E$ is a multiset
    return Graph($V, E$)
end function

function SingletonPartition(Graph $G$)
    return $\{\{v\} \mid v \in V(G)\}$    ▷ Assign each node to its own community
end function
```

From numerical experiments, both seem to run in near-linear time in the number of edges. However, the constant factor of the Louvain algorithm is larger than the constant factor of the Leiden algorithm, i.e. it is slower overall. Implementation: https://github.com/vtraag/leidenalg

The student could make experiments to test the complexity

Reference for this section

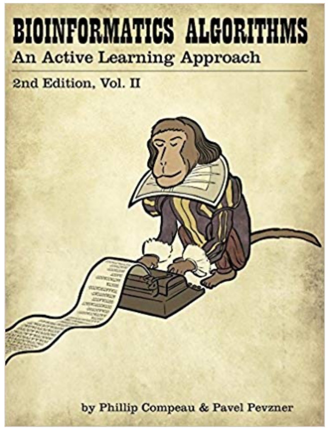


BIOINFORMATICS ALGORITHMS

An Active Learning Approach

2nd Edition, Vol. I

by Phillip Compeau & Pavel Pevzner



BIOINFORMATICS ALGORITHMS

An Active Learning Approach

2nd Edition, Vol. II

by Phillip Compeau & Pavel Pevzner

- Chapter 8 Vol 2 (pag 72-109)

Markov Clustering Algorithm: https://micans.org/mcl/
https://towardsdatascience.com/markov-clustering-algorithm-577168dad475

Louvain/Leiden:
Vincent D Blondel et al J. Stat. Mech. (2008) P10008
https://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008/pdf

Section 5

Genome Sequencing

Biologists need algorithms for personal genome sequencing

- **2010:** Nicholas Volker became the first human being to be saved by genome sequencing.
  - Doctors could not diagnose his condition; he went through dozens of surgeries.
  - Sequencing revealed a rare mutation in a *XIAP* gene linked to a defect in his immune system.
  - This led doctors to use immunotherapy, which saved the child.

- Different people have slightly different genomes: on average, roughly 1 mutation in 1000 nucleotides.

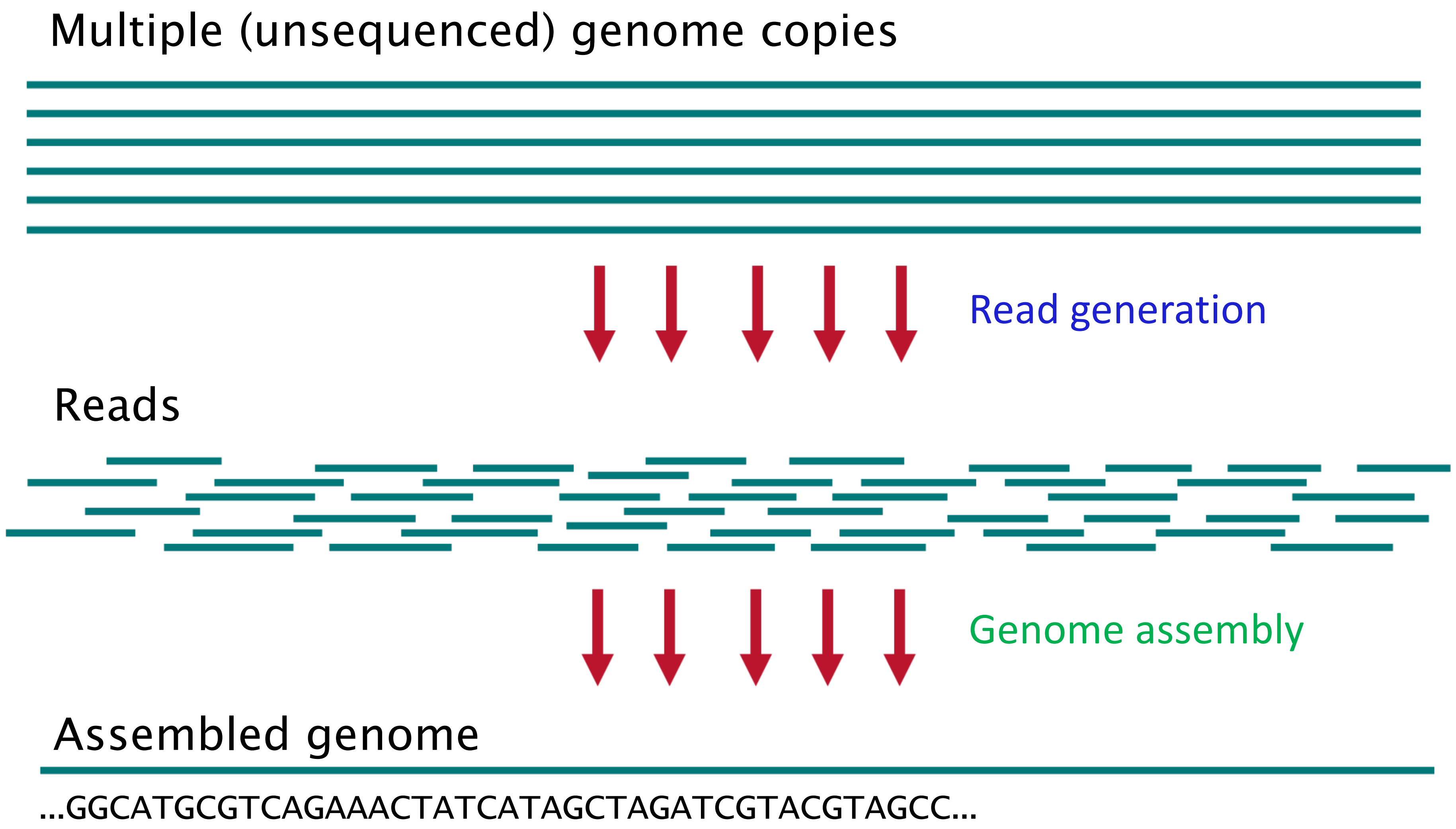
What Makes Genome Sequencing Difficult?

- Modern sequencing machines cannot read an entire genome one nucleotide at a time from beginning to end (like we read a book)
- They can only shred the genome and generate short **reads**.
- The genome assembly is not the same as a jigsaw puzzle: we must use *overlapping* reads to reconstruct the genome, a giant **overlap puzzle**!

Genome Sequencing Problem. Reconstruct a genome from reads.

- Input. A collection of strings Reads.
- Output. A string Genome reconstructed from Reads.

From Experimental to Computational Challenges



Multiple (unsequenced) genome copies

Read generation

Reads

Genome assembly

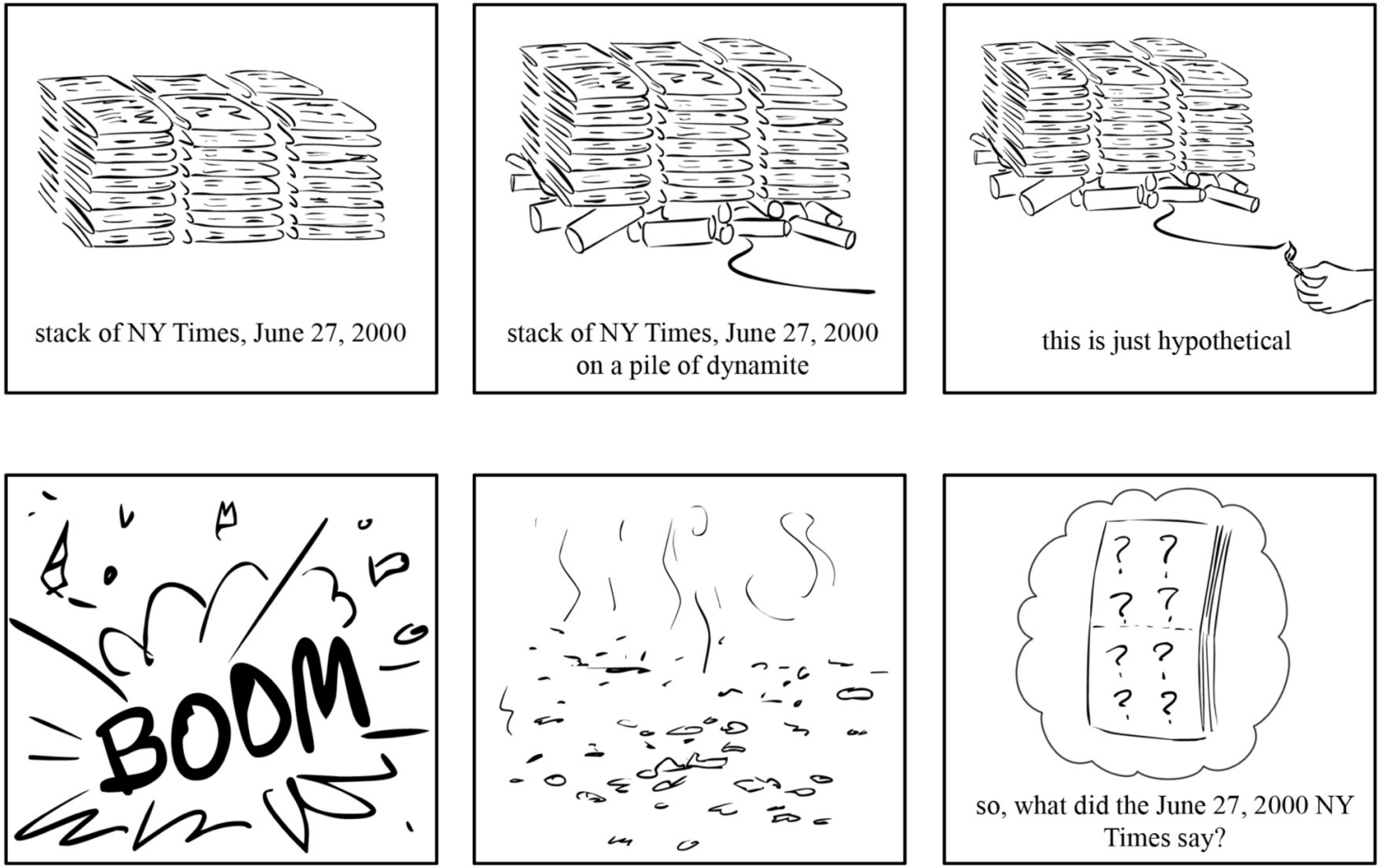
Assembled genome

...GGCATGCGTCAGAAACTATCATAGCTAGATCGTACGTAGCC...

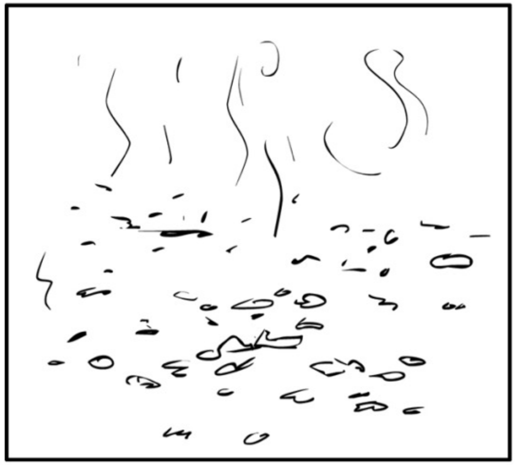
Computational topics in this lecture

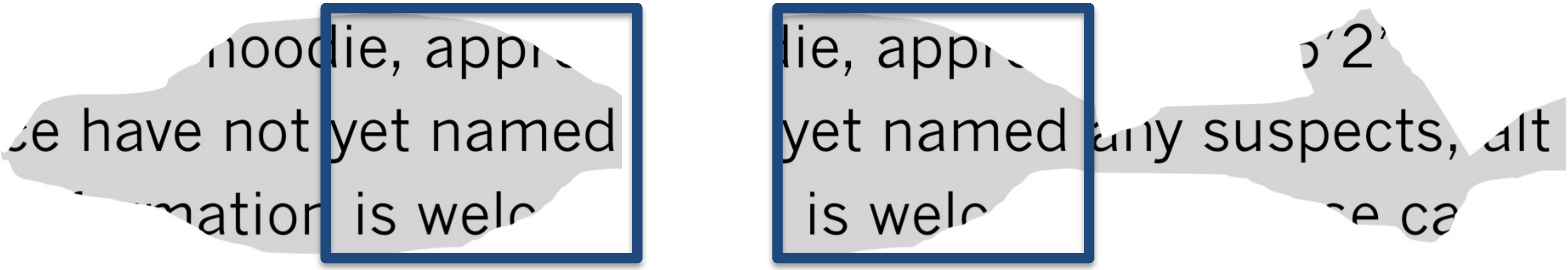
- What Is Genome Sequencing: Exploding Newspapers analogy
- The String Reconstruction Problem
- String Reconstruction as a Hamiltonian Path Problem
- String Reconstruction as an Eulerian Path Problem
- De Bruijn Graphs
- Euler’s Theorem
- Assembling Read-Pairs
- De Bruijn Graphs Face Harsh Realities of Assembly

The Newspaper Problem


stack of NY Times, June 27, 2000
stack of NY Times, June 27, 2000 on a pile of dynamite
this is just hypothetical
BOOM
? ? ? ? ? ? ? ?
so, what did the June 27, 2000 NY Times say?

The newspaper problem as an overlapping puzzle





The Newspaper Problem as an Overlapping Puzzle

The New York Times

Genetic Code of Human Life Is Cracked by Scientists

JUSTICES REAFFIRM MIRANDA RULE, 7-2; A PART OF 'CULTURE'

A SHARED SUCCESS

A Pearl and a Hodgepodge: Human DNA

Multiple-Party Ballot Rejected in Primaries

Clinton Raises Estimate of Surplus And the Stakes on How to Use It

Sharansky in Eyes of Israelis: A Hero or Betrayer of Peace?

Vote in Zimbabwe Shows Opposition Making Big Gains

Buildings' Savior Now a Troubled Landlord

Multiple Copies of a Genome (Millions of them)

stack of NY Times, June 27, 2000

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC

Breaking the Genomes at Random Positions

CTGATG★TGGACTACGC★ACTACTGCT★GCTGTATTA★GATCAGCTACC★CATCGTAGCTA★GATGCATTAGC★AGCTATCG★ATCAGCTAC★ACATCGTAGC

CTGA★GATGGACT★CGCTACTACT★CTAGCTGTAT★ACGATCAGC★ACCACATCGT★GCTACGATGC★TTAGCAAGC★ATCGGATCA★CTACCACAT★GTAGC

CTGATG★TGGACTACG★TACTACTGCTA★CTGTATTAC★ATCAGCTA★CACATCGTAGC★ACGATGCATTA★CAAGCTAT★GGATCAGCT★CCACATCGTAGC

CTGATGATGG★CTACGCTAC★ACTGCTAGCT★TATTACGAT★AGCTACCACA★CGTAGCTACGA★GCATTAGCAA★CTATCGGA★CAGCTACCAC★TCGTAGC

Generating “Reads”

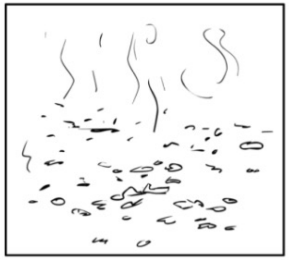
CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC

CTGATGATG GACTACGCT ACTACTGCTA GCTGTATTACG ATCAGCTACC ACATCGTAGCT ACGATGCATTA GCAAGCTATC GGATCAGCTAC CACATCGTAGC

CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCAC ATCGTAGCTACG ATGCATTAGCA AGCTATCGG A TCAGCTACCA CATCGTAGC

CTGATGATGGACT ACGCTACTACT GCTAGCTGTAT TACGATCAGC TACCACATCGT AGCTACGATGCA TTAGCAAGCT ATCGGATCA GCTACCACATC GTAGC

“Burning” Some Reads



CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC

CTGATGATG GACTACGCT ACTACTGCTA GCTGTATTACG ATCAGCTACC ACATCGTAGCT ACGATGCATTA GCAAGCTATC GGATCAGCTAC CACATCGTAGC

CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCAC ATCGTAGCTACG ATGCATTAGCA AGCTATCGG A TCAGCTACCA CATCGTAGC

CTGATGATGGACT ACGCTACTACT GCTAGCTGTAT TACGATCAGC TACCACATCGT AGCTACGATGCA TTAGCAAGCT ATCGGATCA GCTACCACATC GTAGC

No Idea What Position Every Read Comes From

ATCAGCTACCA

CTGATGATGGACT

TGGACTACGCTAC

AGCTATCGG

AGCTACGATGCA

ATGCATTAGCA

TACTGCTAG

ATCAGCTACC

TTAGCAAGCT

TCGTAGCTACG

CATCGTAGC

CTGATGA

GCTGTATTACG

CTGATGATGG

TACTGCTAGCT

TCAGCTACCA

ATGCATTAGCA

GCAAGCTATC

CTGTATTACG

GCTACCACATC

GCTATCGGA

ATGCATTAGCAA

CTGATGATG

GACTACGCT

CATCGTAGC

ATCAGCTACCACA

ACGATGCATTA

CACATCGTAGC

ACGCTACTACT

ATCGGATCA

ACTACTGCTA

TACGATCAGC

TACCACATCGT

ACATCGTAGCT

GGATCAGCTAC

GCAAGCTATC

AGCTACCAC

CTGATGATGG

ACTACGCTAC

ATCGTAGCTACG

TACTGCTAGCT

ATCGTAGCTACG

GCTAGCTGTAT

GTATTACGATC

What Is k-mer Composition?

***Composition*$_3$(TAATGCCATGGGATGTT)=**

TAA
AAT
ATG
TGC
GCC
CCA
CAT
ATG
TGG
GGG
GGA
GAT
ATG
TGT
GTT

k-mer Composition

$\mathit{Composition}_3$(TAATGCCATGGGATGTT)=

TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT

=

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

e.g., lexicographic order (like in a dictionary)

Reconstructing a String from its Composition

String Reconstruction Problem. Reconstruct a string from its k-mer composition.

- Input. A collection of k-mers.
- Output. A Genome such that Composition_k(Genome) is equal to the collection of k-mers.

A Naive String Reconstruction Approach

ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TGC TGG TGT

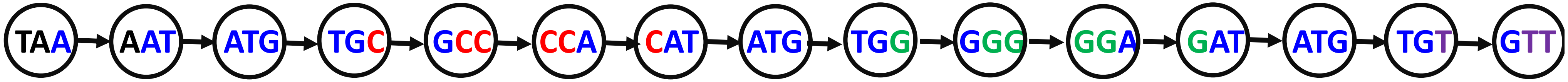
TAA
AAT

ATG ATG CAT CCA GAT GCC GGA GGG TGC TGG

TAA
AAT
ATG
TGT
GTT

Representing a Genome as a Path

Composition$_3$(TAATGCCATGGGATGTT)=

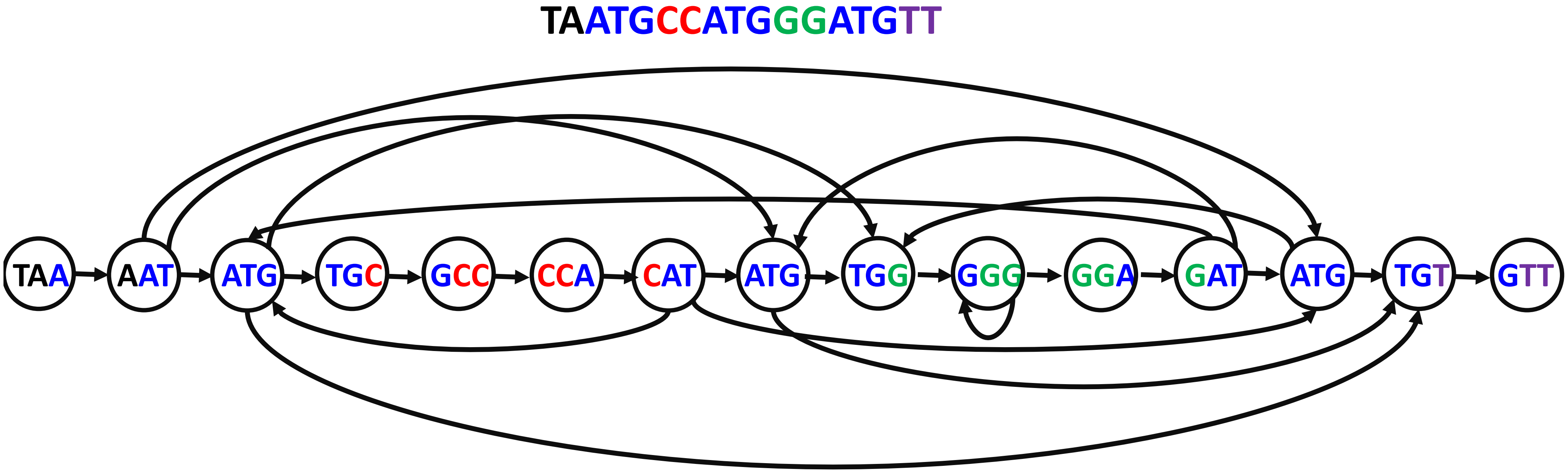


Can we construct this genome path without knowing the genome TAATGCCATGGGATGTT, only from its composition?

Yes. We simply need to connect k-mer$_1$ with k-mer$_2$ if suffix(k-mer$_1$)=prefix(k-mer$_2$).
E.g. TAA → AAT

□ □

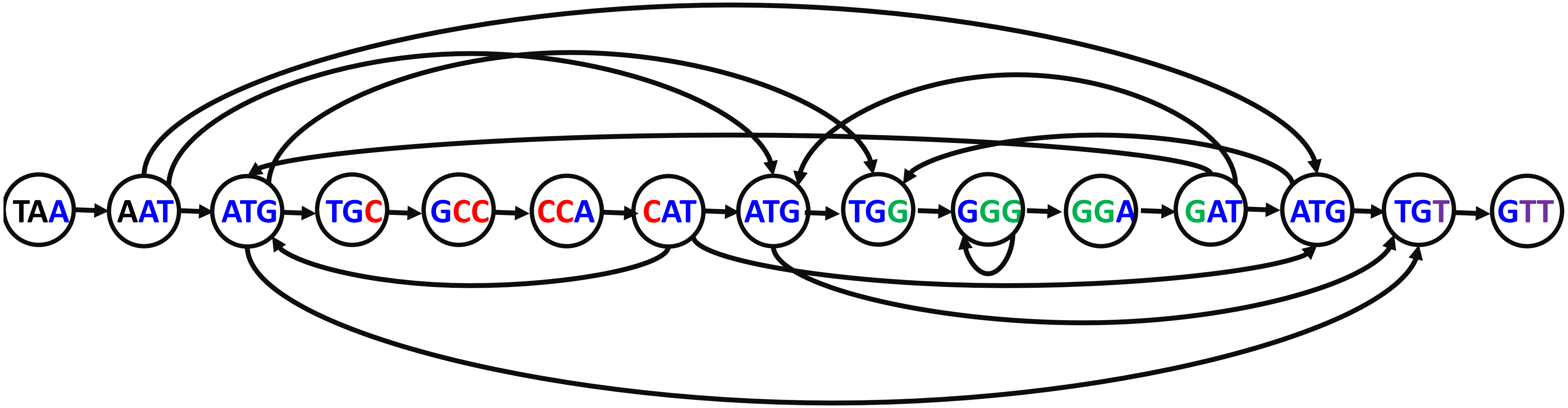
A Path Turns into a Graph



Yes. We simply need to connect k-mer$_1$ with k-mer$_2$ if suffix(k-mer$_1$)=prefix(k-mer$_2$).
E.g. TAA → AAT

A Path Turns into a Graph

TAATGCCATGGGATGTT

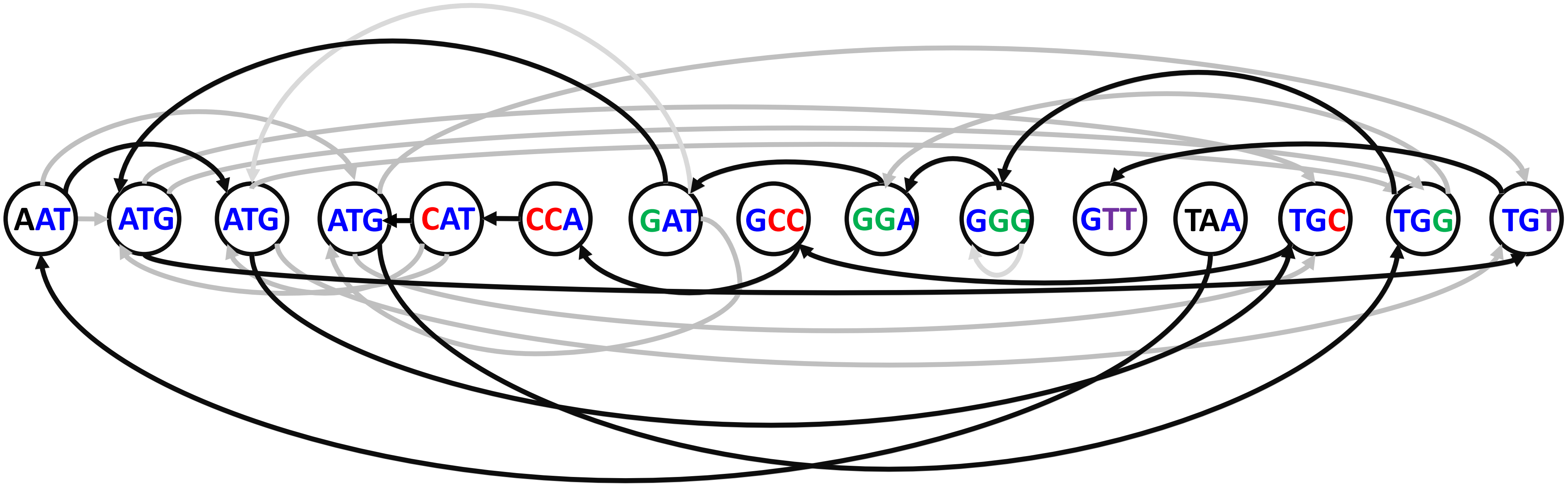


Can we still find the genome path in this graph?

Where Is the Genomic Path?

A **Hamiltonian path:** a path that visits each node in a graph exactly once.

TAATGCCATGGGATGTT



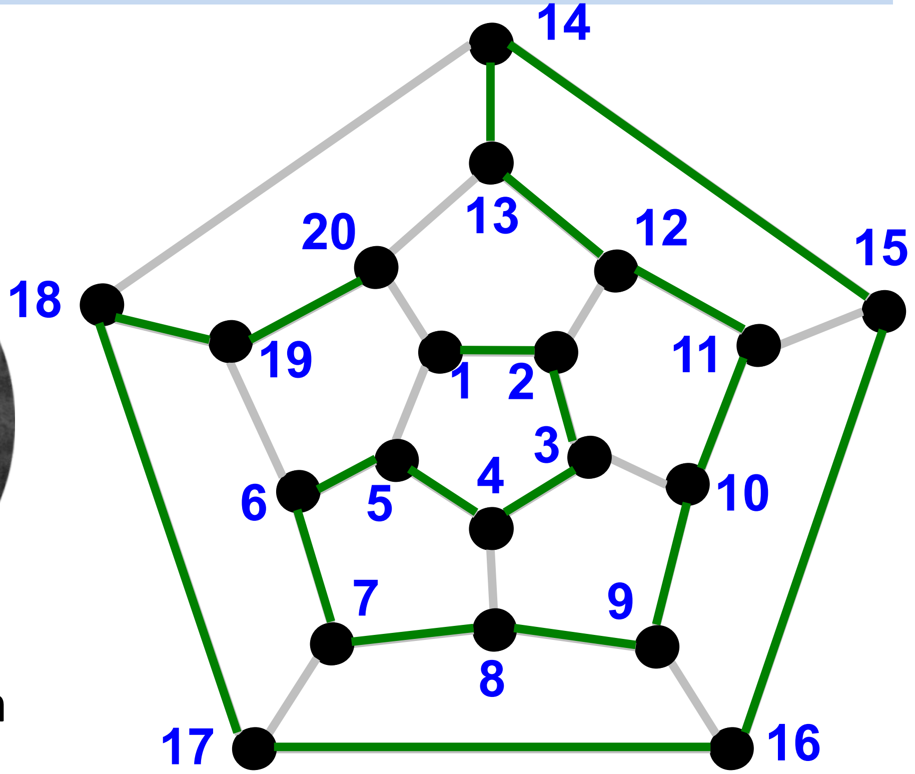
What are we trying to find in this graph?

Does This Graph Have a Hamiltonian Path?

Hamiltonian Path Problem. Find a Hamiltonian path in a graph.
Input. A graph.
Output. A path visiting every node in the graph exactly once.

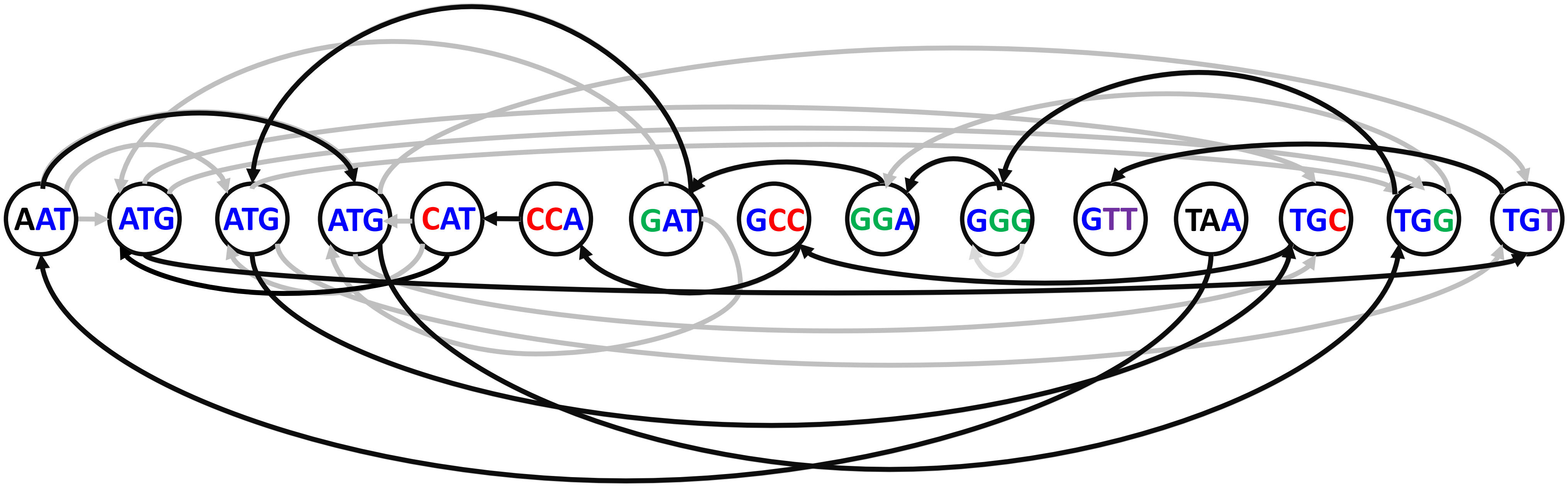
Icosian game (1857)

William
Hamilton

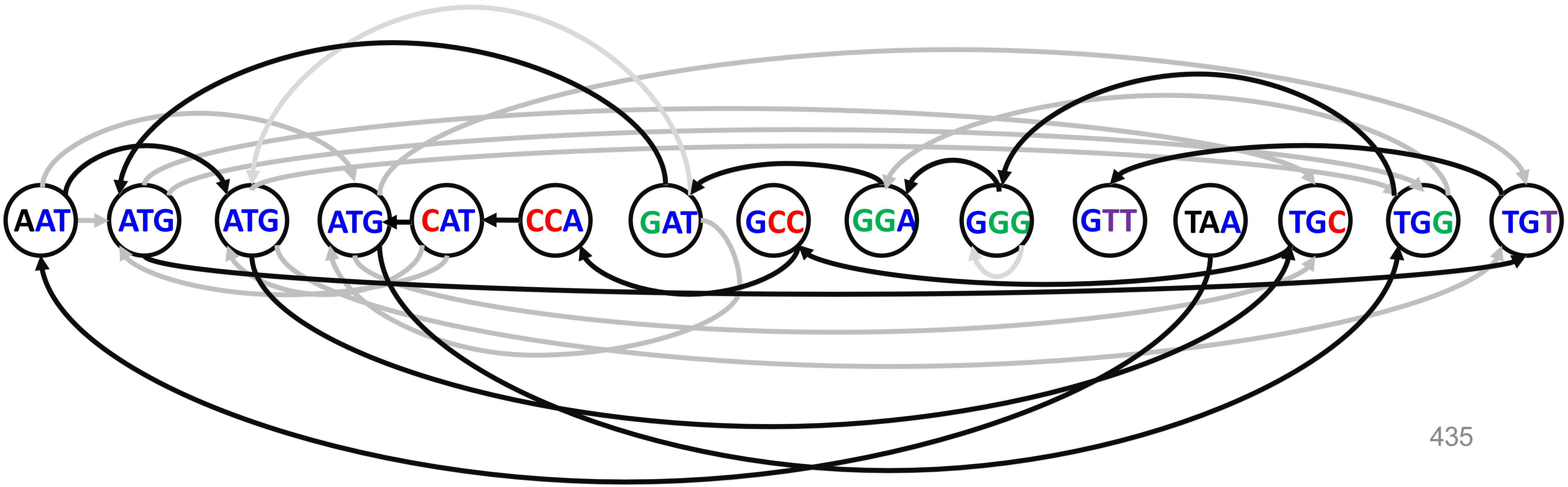


Undirected graph

TAATGGGATGCCATGTT

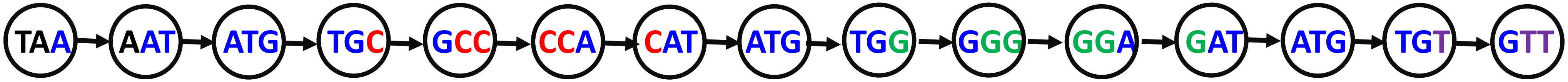


TAATGCCATGGGATGTT

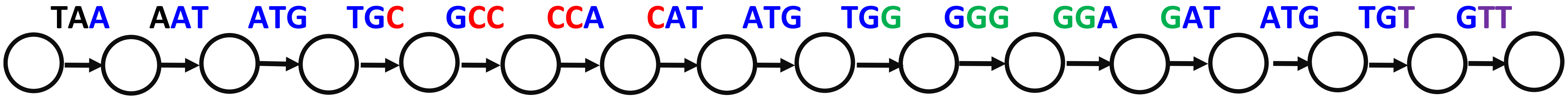


A Slightly Different Path

TAATGCCATGGGATGTT



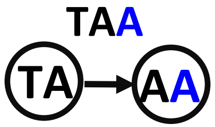
3-mers as nodes



3-mers as edges

How do we label the starting and ending nodes of an edge?

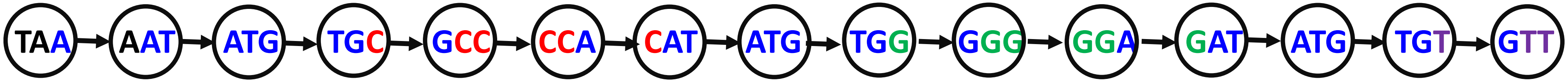
prefix of TAA



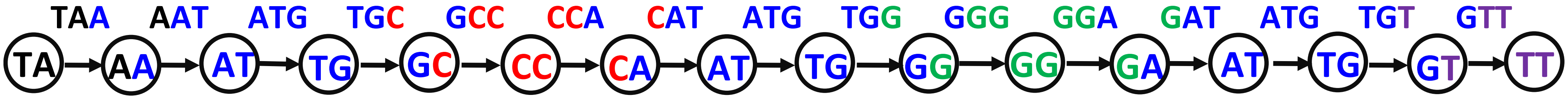
suffix **of TAA**

Labeling Nodes in the New Path

TAATGCCATGGGATGTT

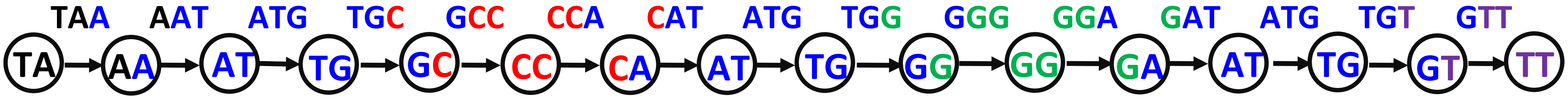


3-mers as nodes



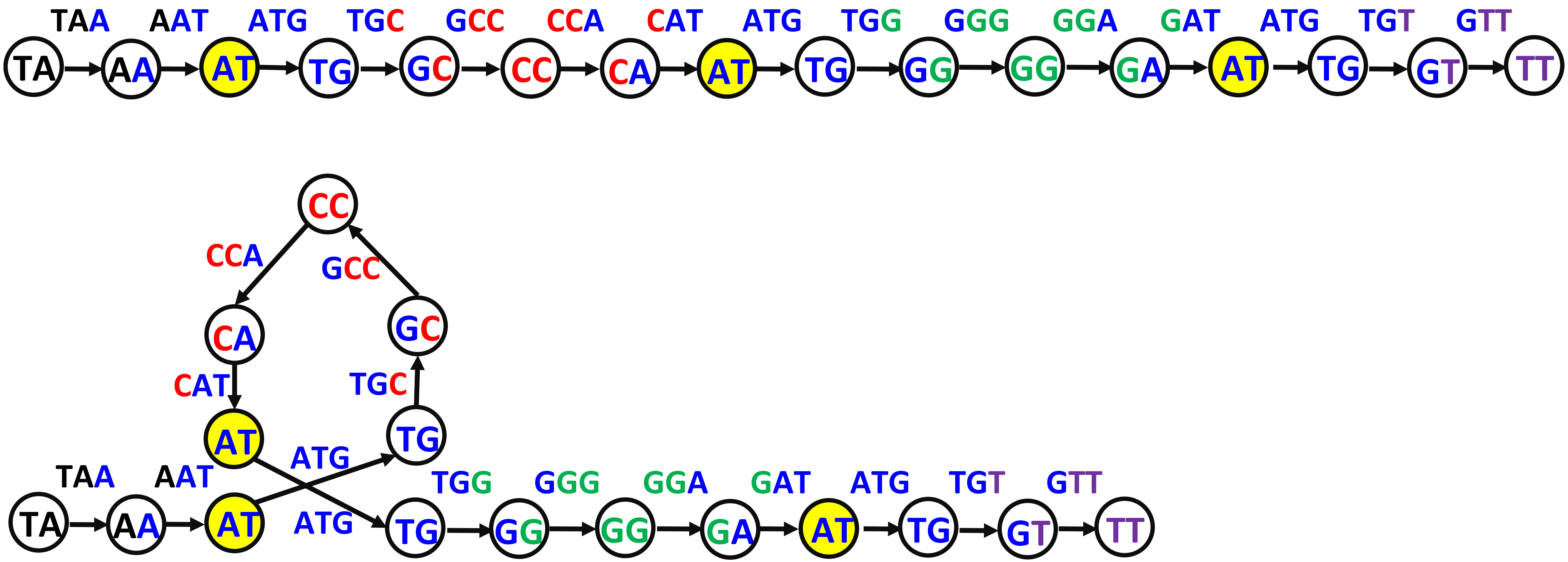
3-mers as edges and 2-mers as nodes

Labeling Nodes in the New Path

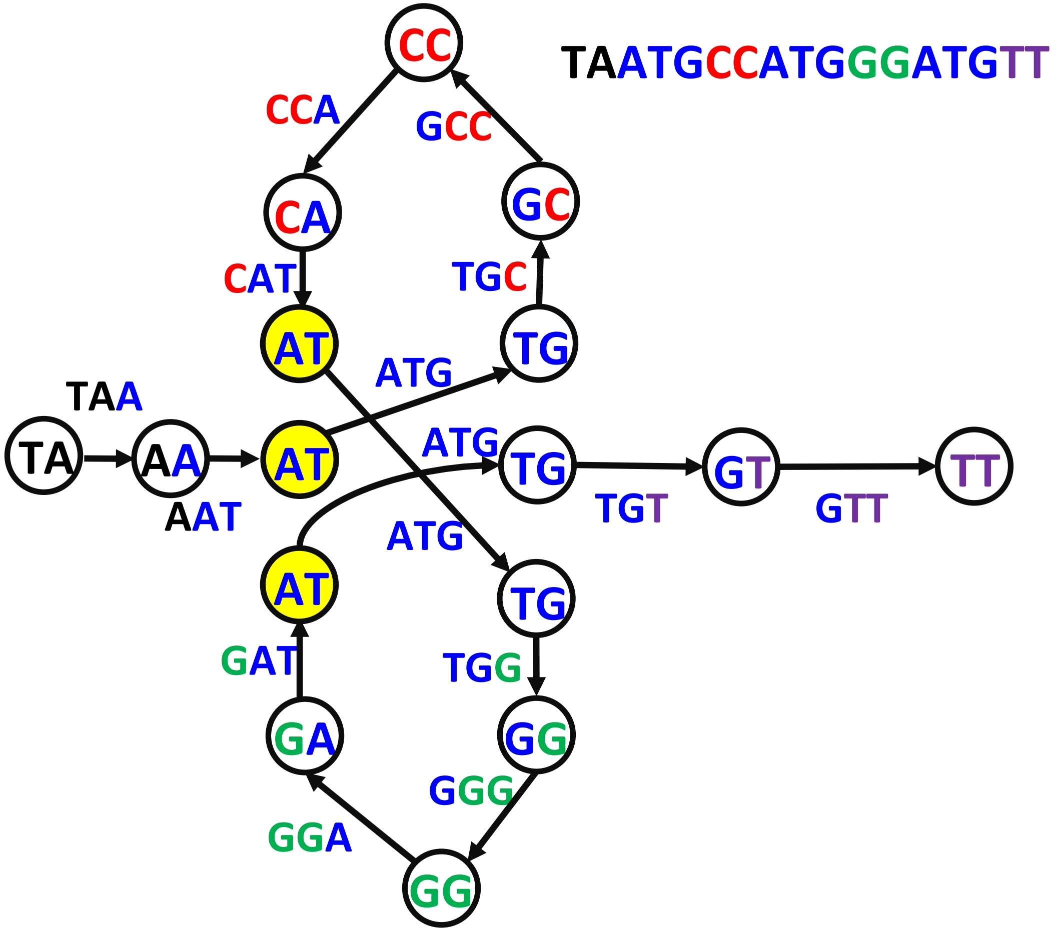


3-mers as edges and 2-mers as nodes

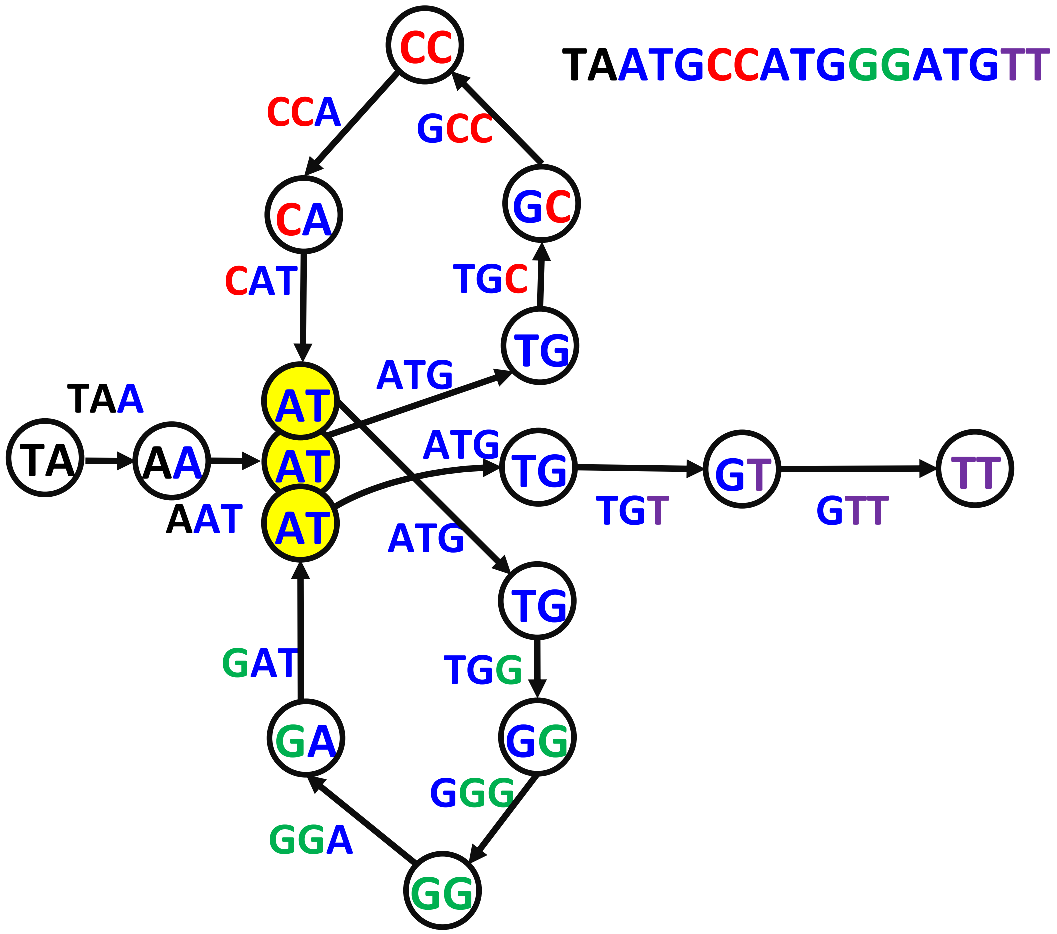
Gluing Identically Labeled Nodes


TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT
TA AA AT TG GC CC CA AT TG GG GG GA AT TG GT TT
CC
CCA
GCC
CA
GC
CAT
TGC
AT
TG
ATG
TAA
AAT
ATG
TGG GGG GGA GAT ATG TGT GTT
TA AA AT TG GG GG GA AT TG GT TT

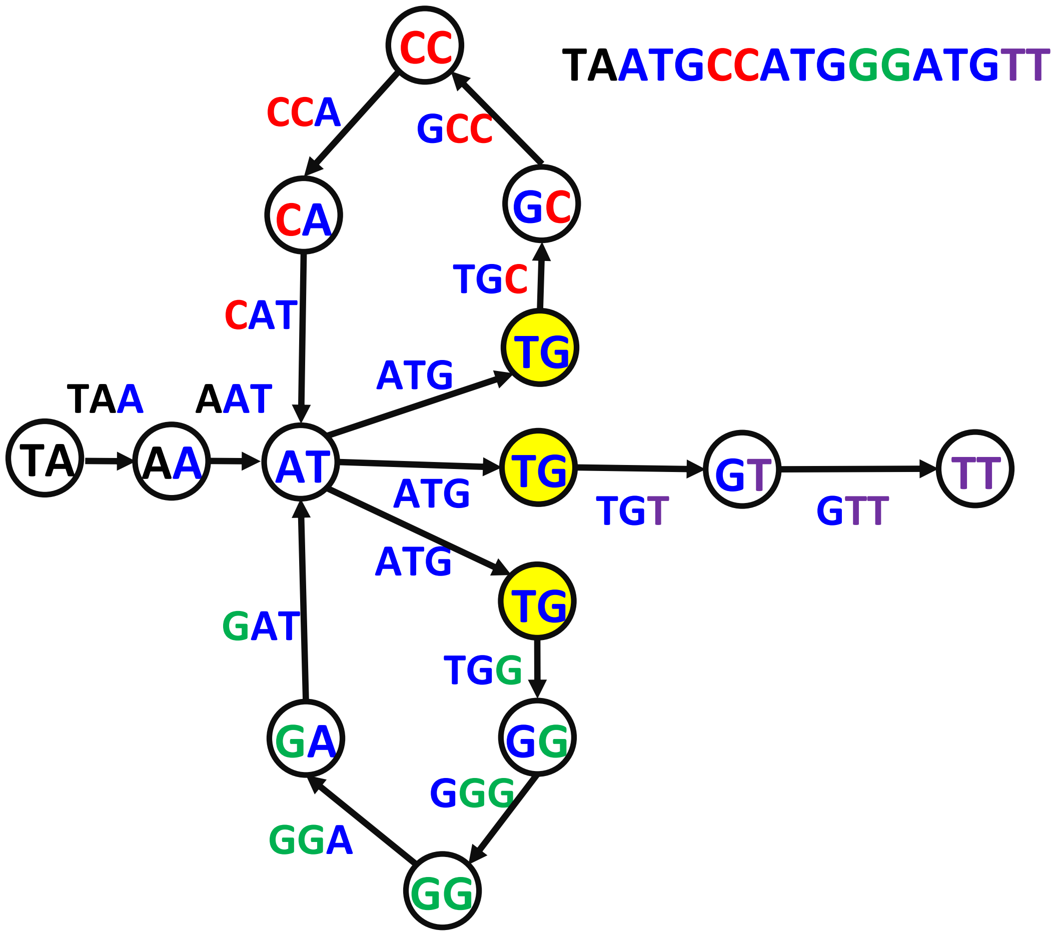
Gluing Identically Labeled Nodes



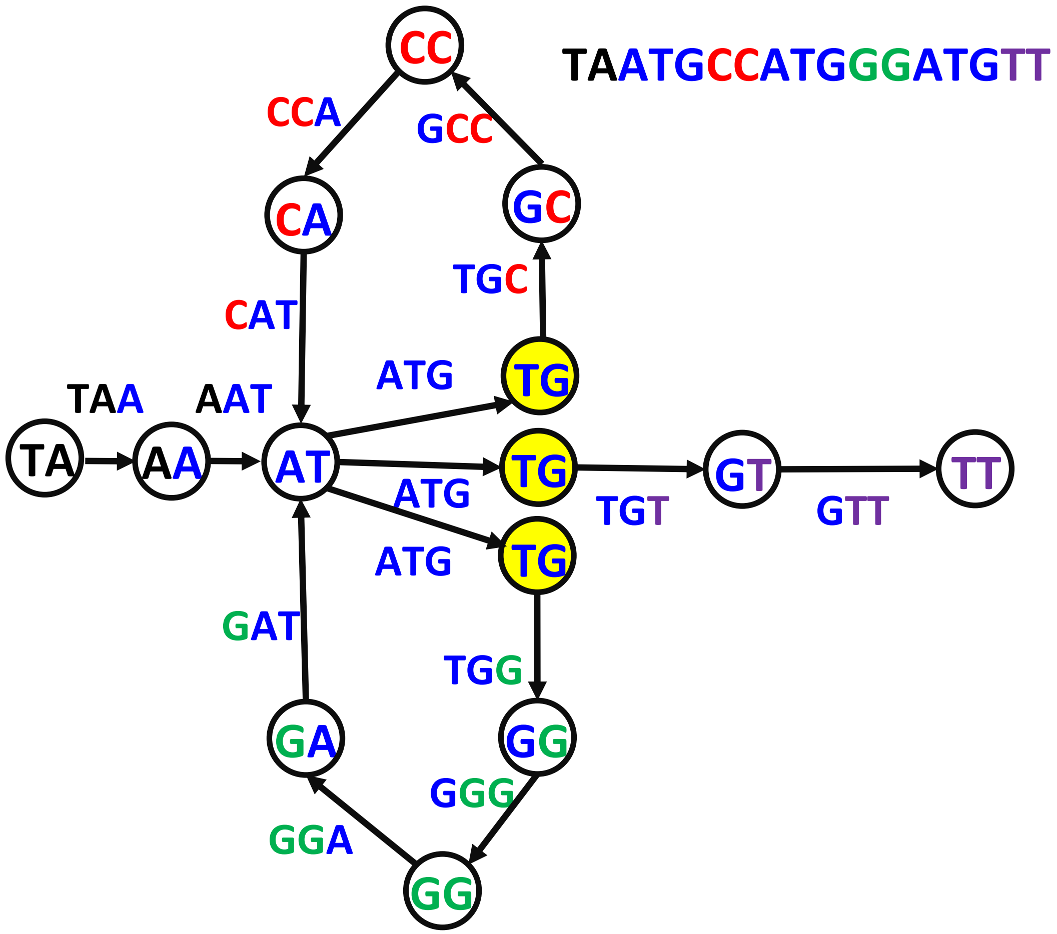
Gluing Identically Labeled Nodes



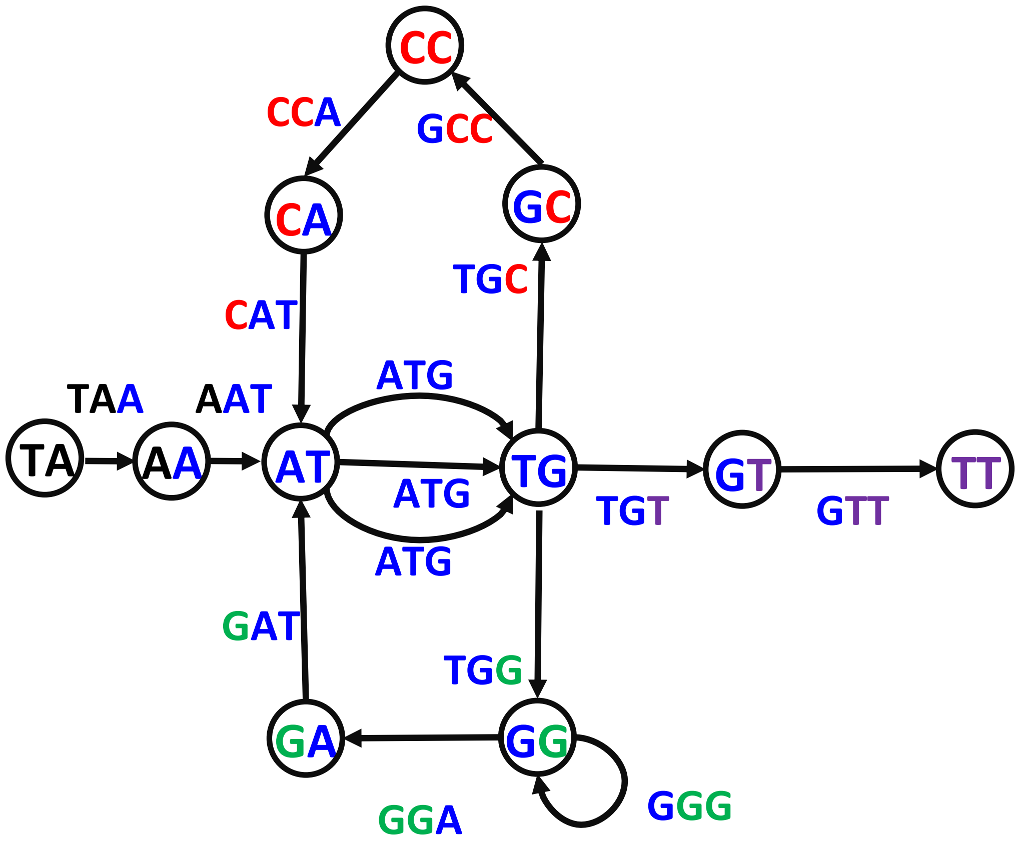
Gluing Identically Labeled Nodes



Gluing Identically Labeled Nodes



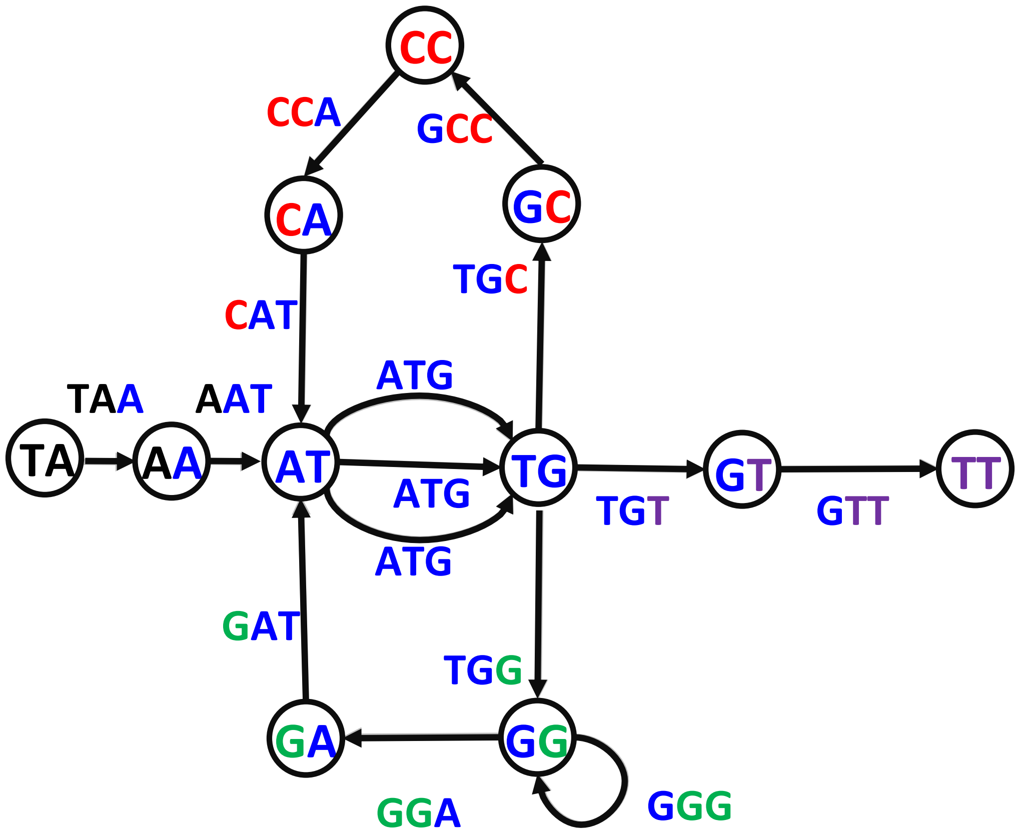
De Bruijn Graph of TAATGCCATGGGATGTT



Where is the Genome hiding in this graph?

It Was Always There!

TAATGCCATGGGATGTT

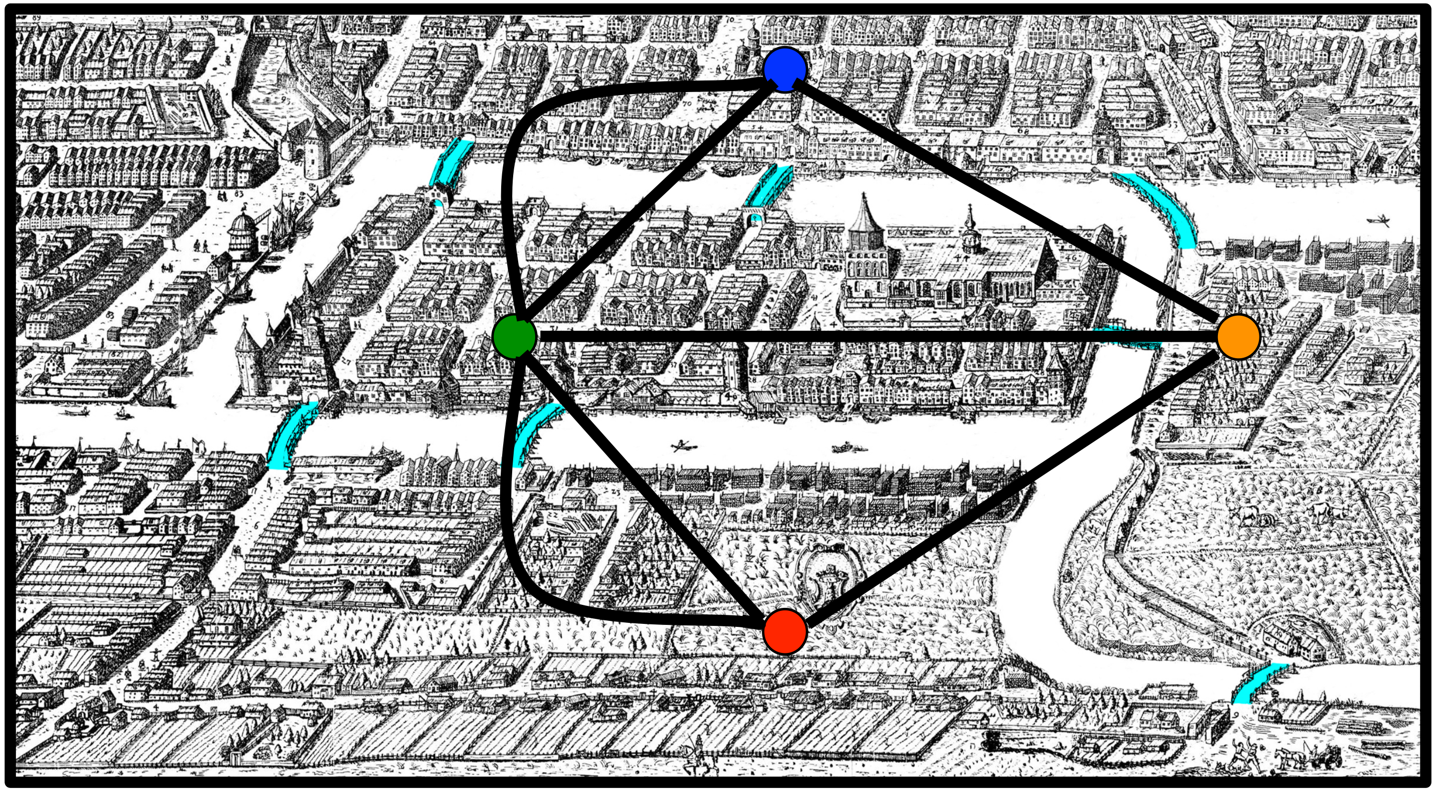


An Eulerian **path** in a graph is a path that visits each edge exactly once.

Eulerian Path Problem

Eulerian Path Problem. Find an Eulerian path in a graph.

- Input. A graph.
- Output. A path visiting every edge in the graph exactly once.



Eulerian Versus Hamiltonian Paths

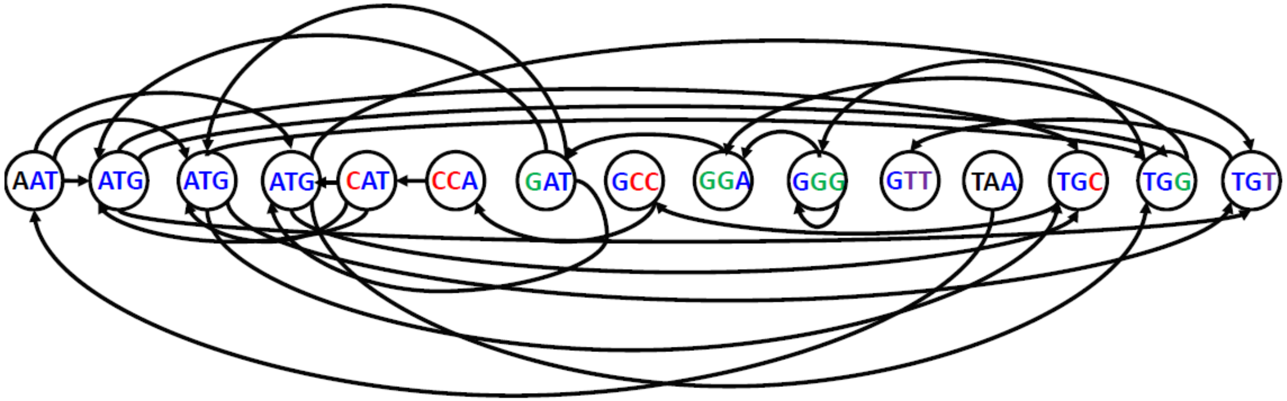
Eulerian Path Problem. Find an Eulerian path in a graph.

- Input. A graph.
- Output. A path visiting every edge in the graph exactly once.

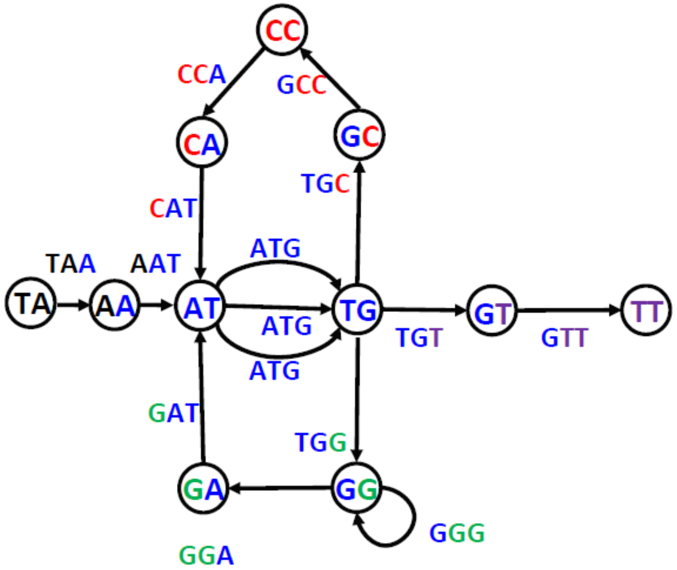
Hamiltonian Path Problem. Find a Hamiltonian path in a graph.

- Input. A graph.
- Output. A path visiting every node in the graph exactly once.

What Problem Would You Prefer to Solve?

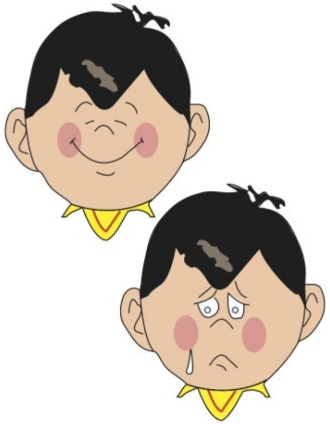


Hamiltonian Path Problem



Eulerian Path Problem

While Euler solved the Eulerian Path Problem (even for a city with a million bridges), nobody has developed a fast algorithm for the Hamiltonian Path Problem yet.



NP-Complete Problems

- The Hamiltonian Path Problem belongs to a collection containing thousands of computational problems for which no fast algorithms are known.

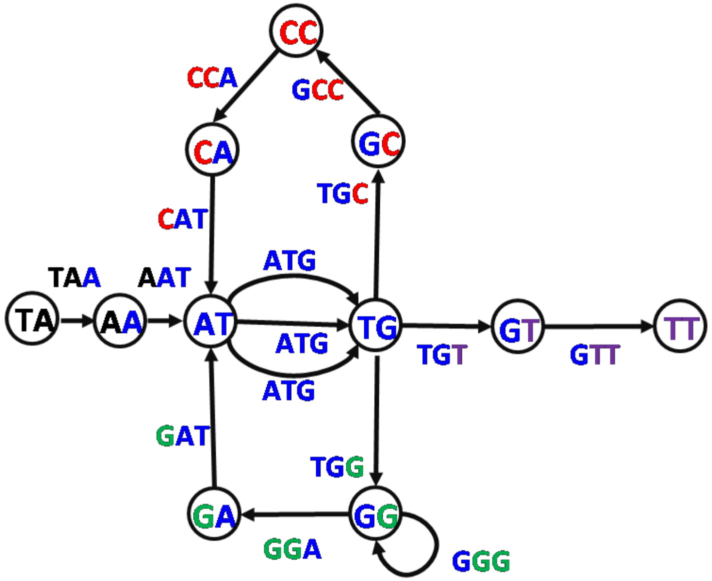
That would be an excellent argument, but the question of whether or not NP-Complete problems can be solved efficiently is one of seven **Millennium Problems** in mathematics.

NP-Complete problems are all equivalent: find an efficient solution to one, and you have an efficient solution to them all.

Eulerian Path Problem

Eulerian Path Problem. Find an Eulerian path in a graph.

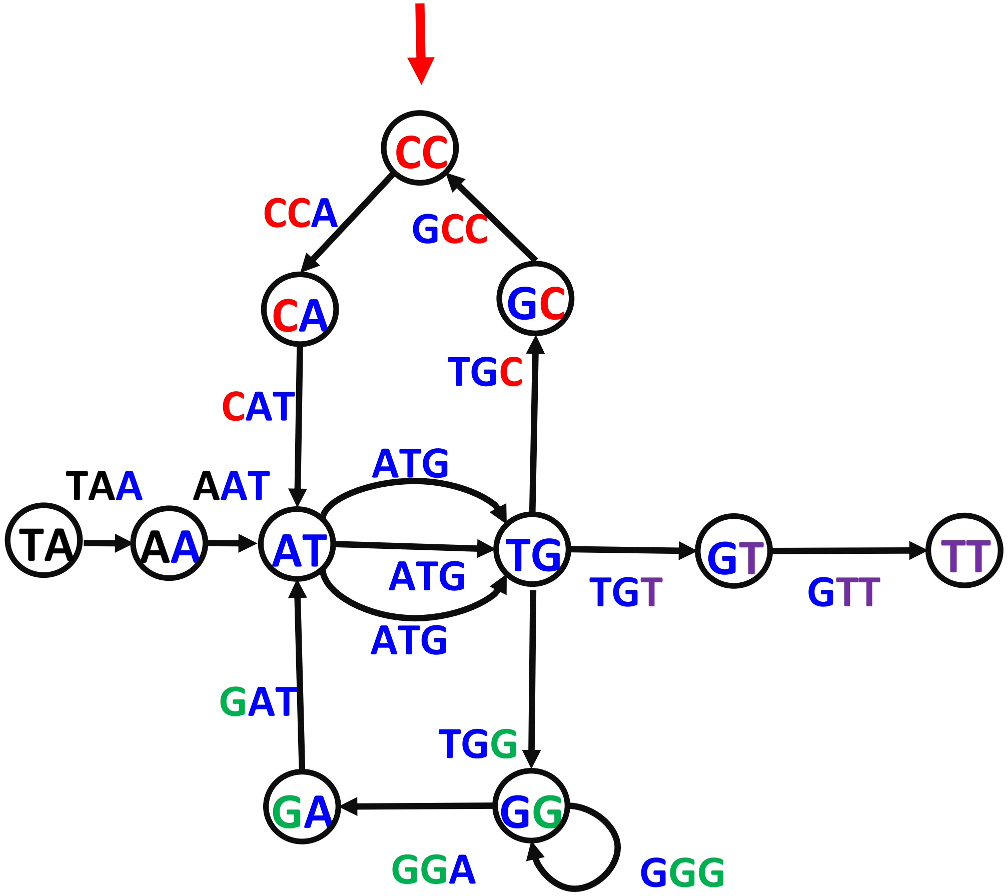
- Input. A graph.
- Output. A path visiting every edge in the graph exactly once.



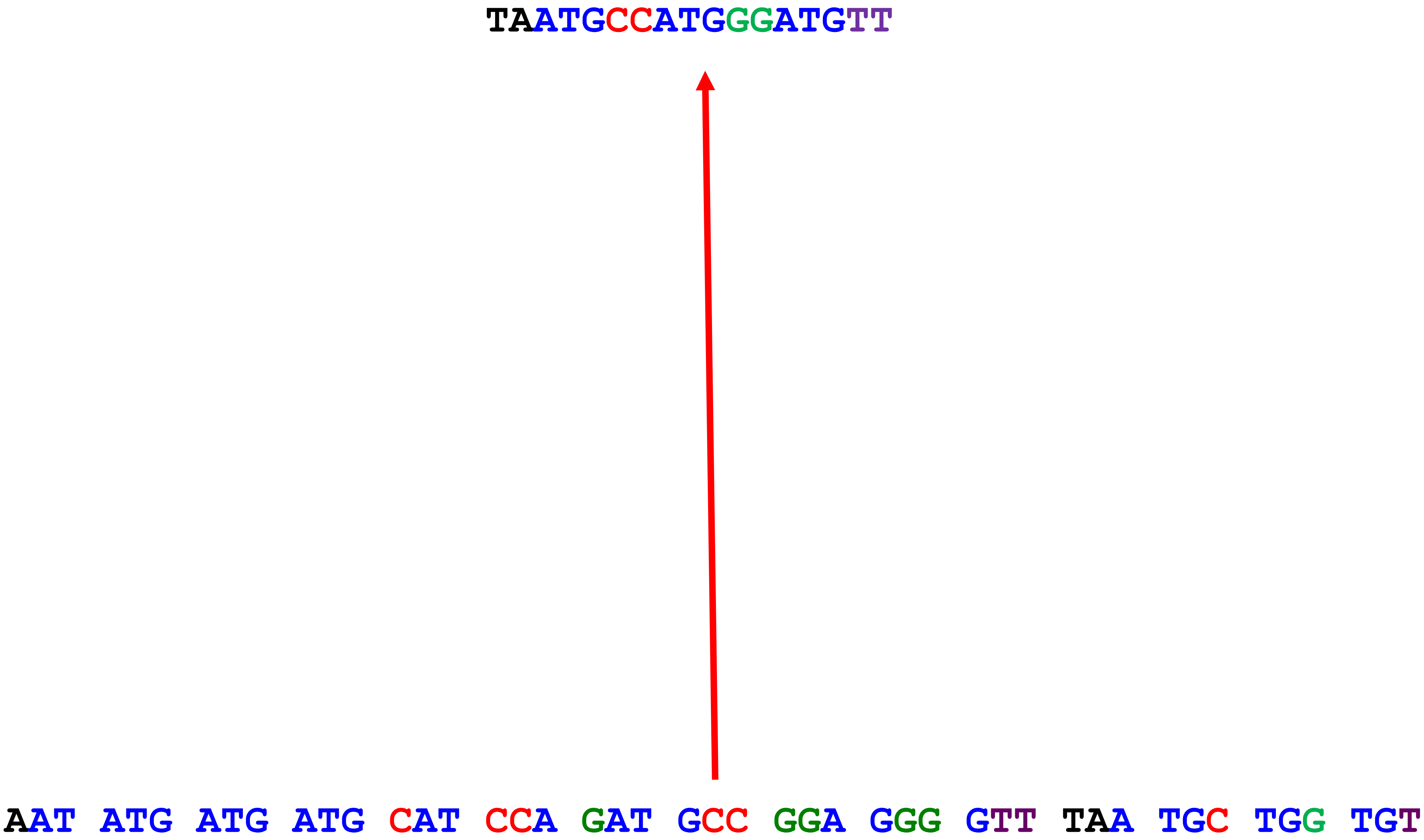
We constructed the de Bruijn graph from Genome, but in reality, Genome is unknown!

What We Have Done: From Genome to de Bruijn Graph

TAATGCCATGGGATGTT



What We Want: From Reads (k-mers) to Genome

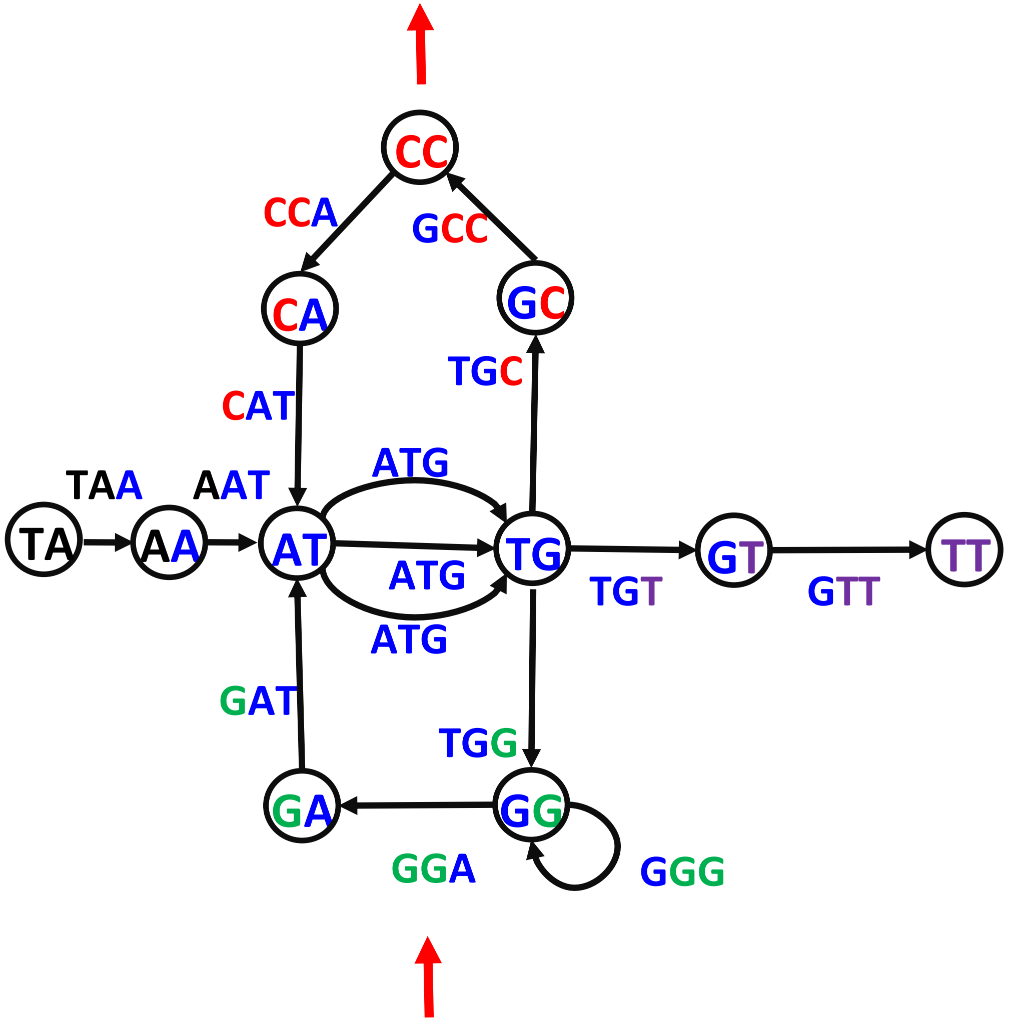


TAATGCCATGGGATGTT

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

What We will Show: From Reads to de Bruijn Graph to Genome

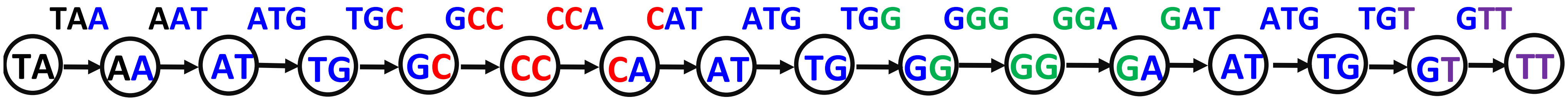
TAATGCCATGGGATGTT



AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

Constructing de Bruijn Graph when Genome Is Known

TAATGCCATGGGATGTT



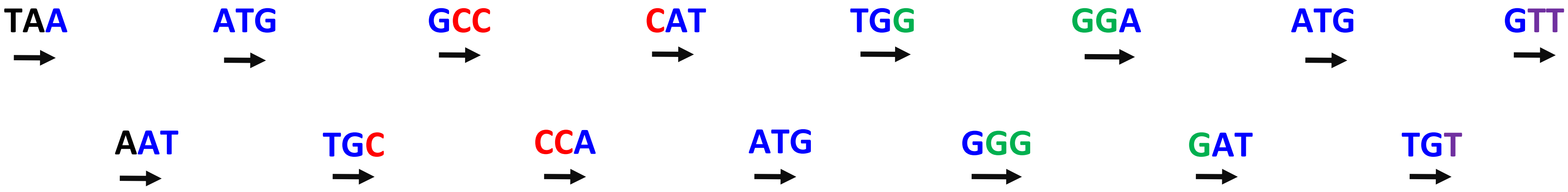
Constructing de Bruijn when Genome Is Unknown

TAA ATG GCC CAT TGG GGA ATG GTT

AAT TGC CCA ATG GGG GAT TGT

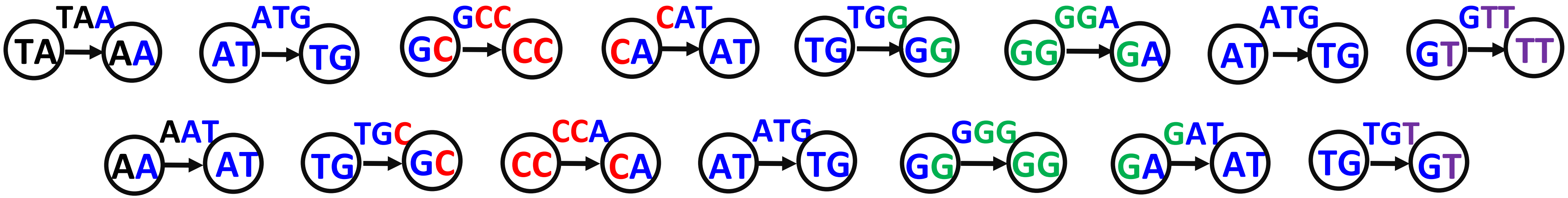
Composition$_3$(**TAATGCCATGGGATGTT**)

Representing Composition as a Graph Consisting of Isolated Edges



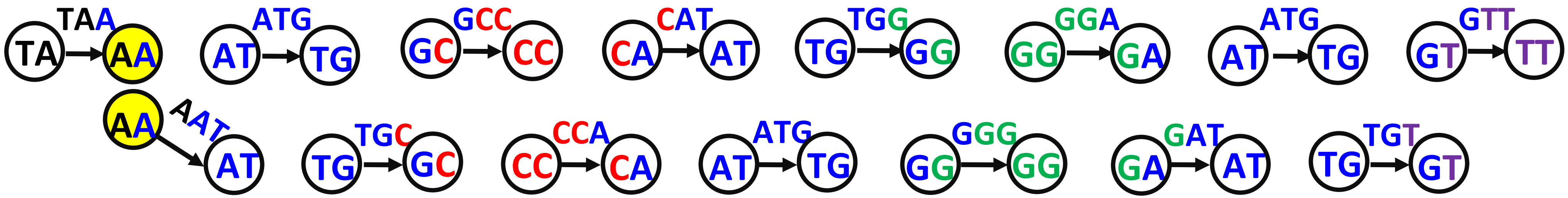
Composition$_3$(**TAATGCCATGGGATGTT**)

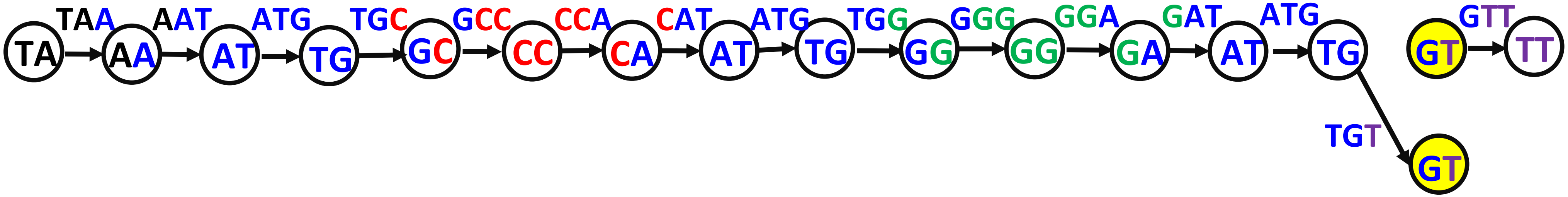
Constructing de Bruijn Graph from k-mer Composition



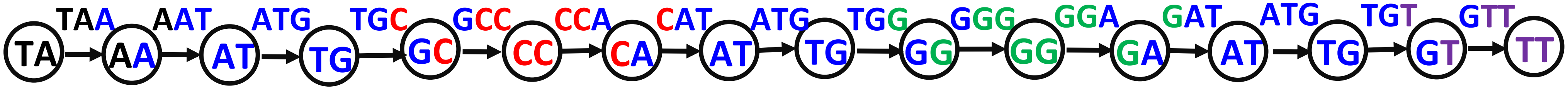
Composition$_3$(**TAATGCCATGGGATGTT**)

Gluing Identically Labeled Nodes

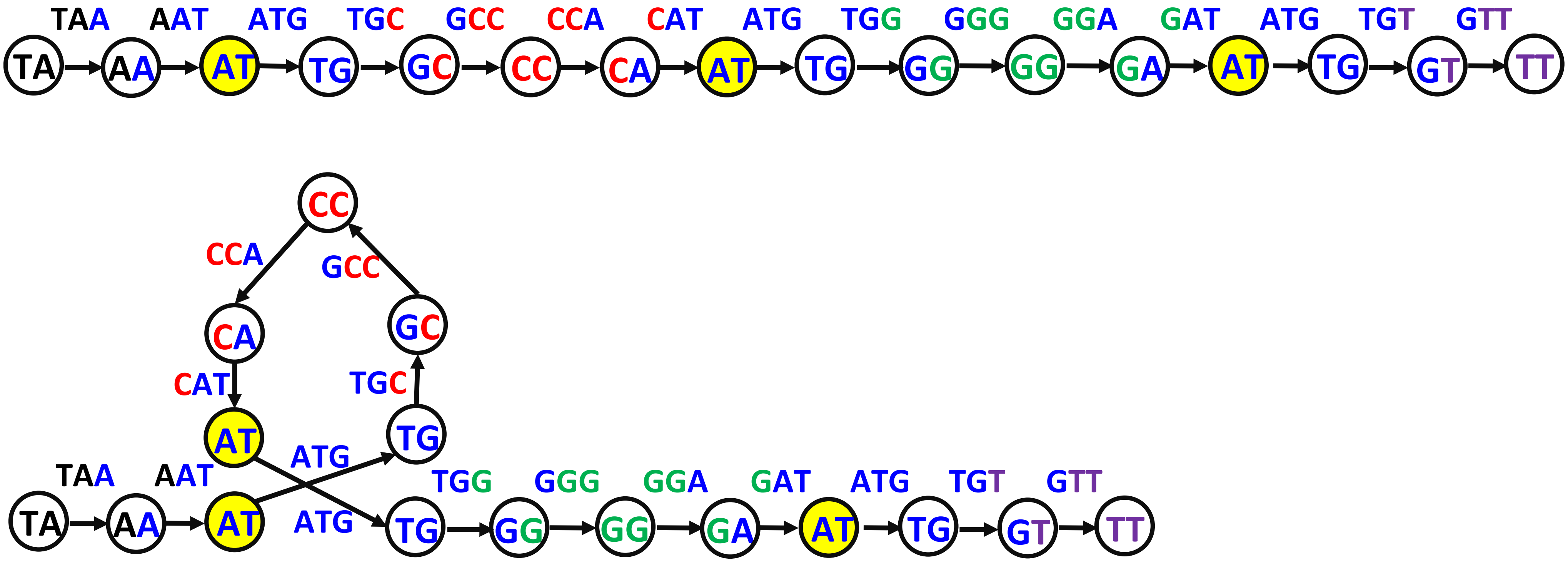



TA
TAA
AA
AAT
AT
ATG
TG
TGC
GC
GCC
CC
CCA
CA
CAT
AT
ATG
TG
TGG
GG
GGG
GG
GGA
GA
GAT
AT
ATG
TG
TGT
GT
GT
GTT
TT

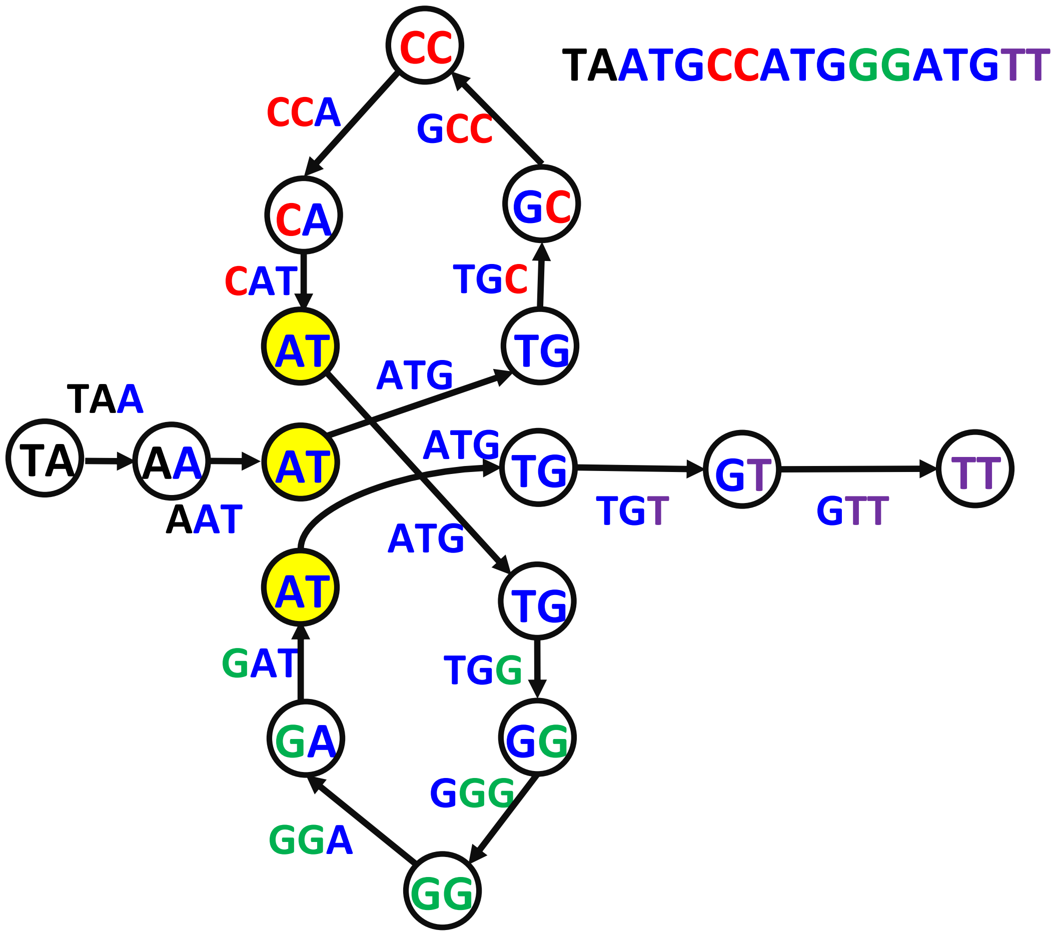
We Are Not Done with Gluing Yet



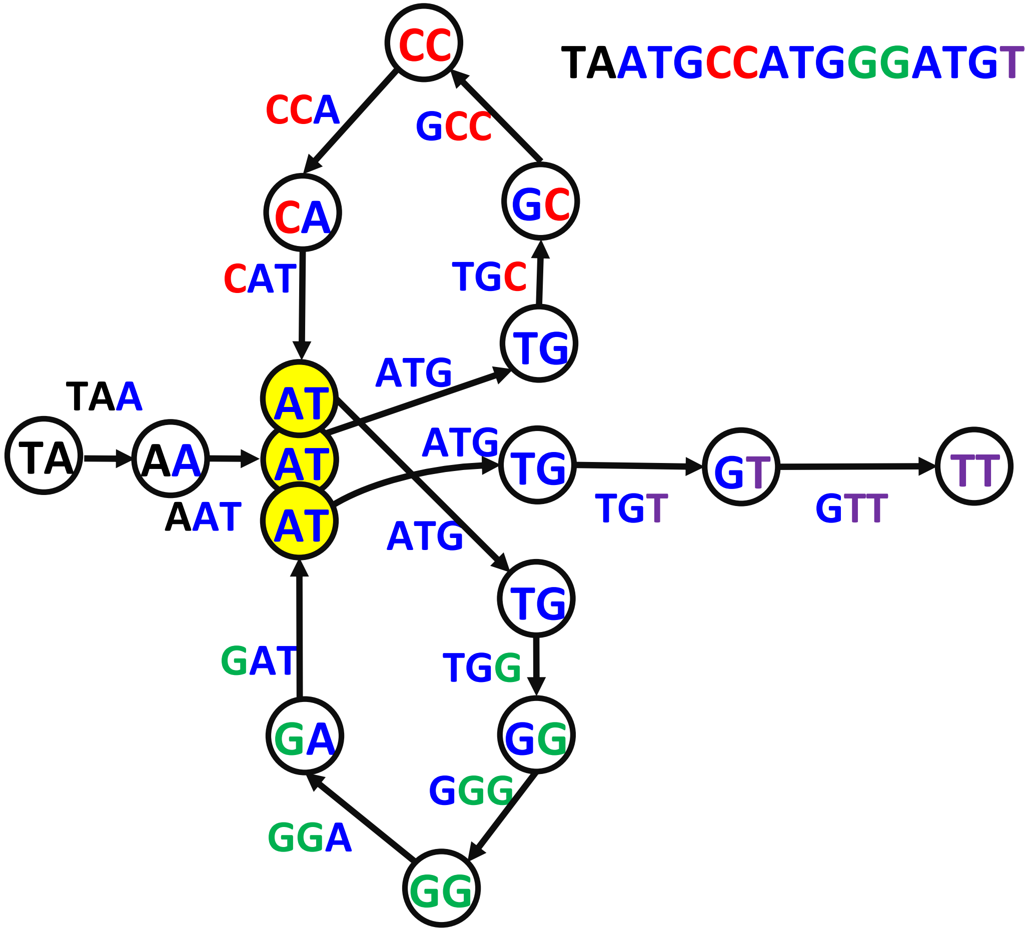
Gluing Identically Labeled Nodes



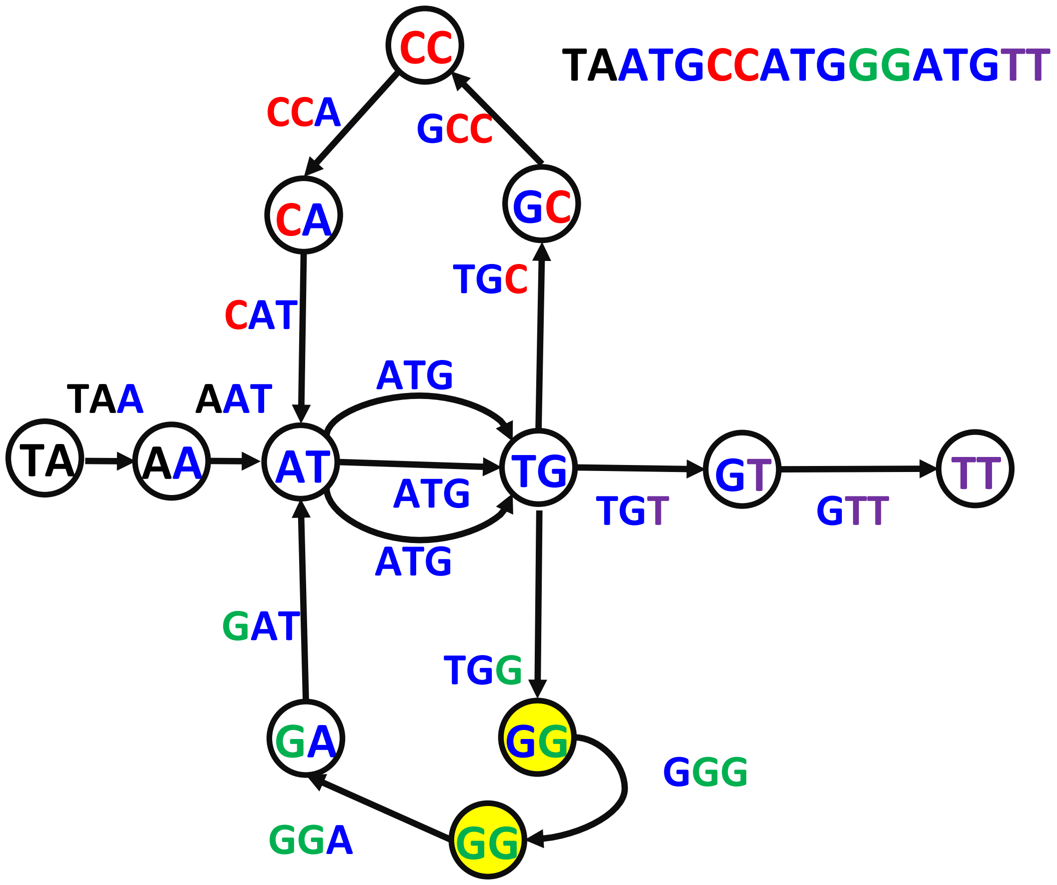
Gluing Identically Labeled Nodes



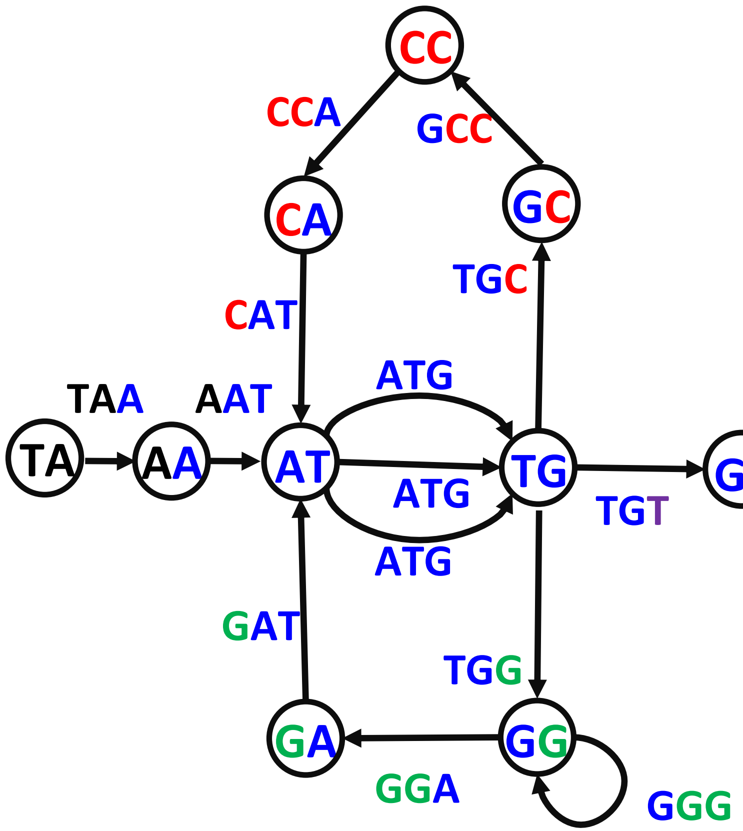
TAATGCCATGGGATGTT



Gluing Identically Labeled Nodes



The Same de Bruijn Graph: DeBruin(Genome)=DeBruin(Genome Composition)

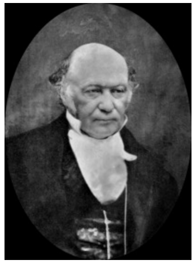


Constructing de Bruijn Graph

De Bruijn graph of a collection of *k*-mers:

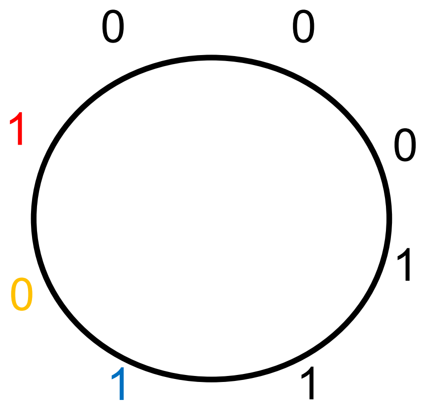
- Represent every *k*-mer as an edge between its prefix and suffix
- Glue **ALL** nodes with identical labels.

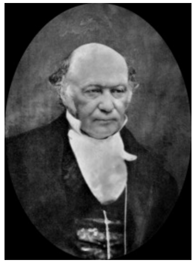
```
DeBruijn(k-mers)
  form a node for each (k-1)-mer from k-mers
  for each k-mer in k-mers
     connect its prefix node with its suffix node by an edge
```

From Hamilton  to Euler to de Bruijn

Universal String Problem (Nicolaas de Bruijn, 1946). Find a circular string containing each binary k-mer exactly once.

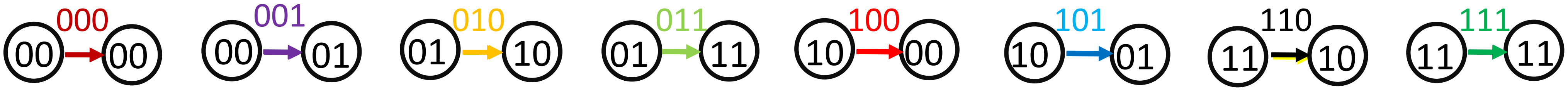
000 001 010 011 100 101 110 111

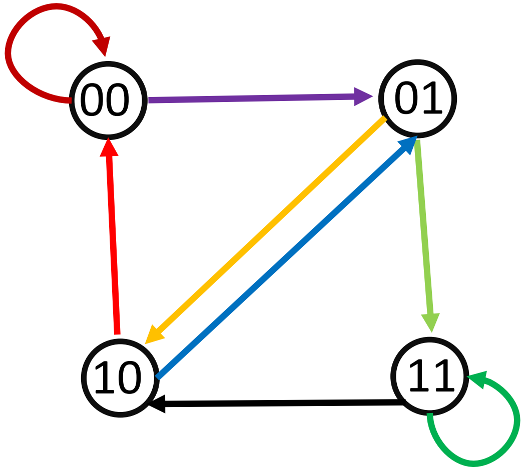


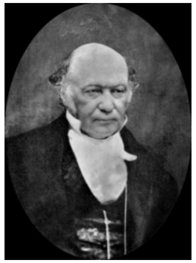
From Hamilton  to Euler to de Bruijn

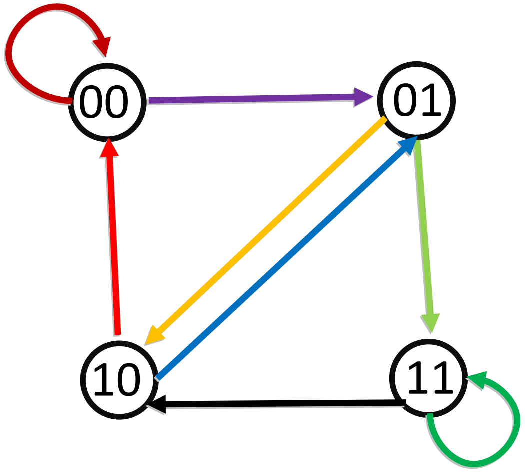
Universal String Problem (Nicolaas de Bruijn, 1946). Find a circular string containing each binary k-mer exactly once.

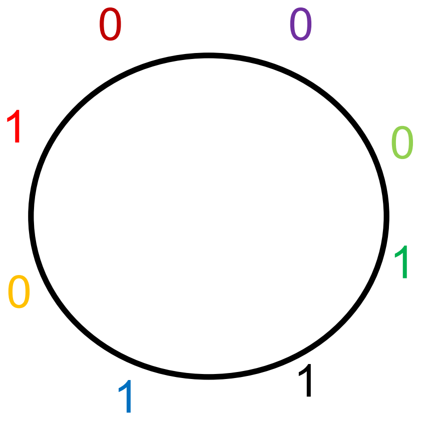
000 001 010 011 100 101 110 111



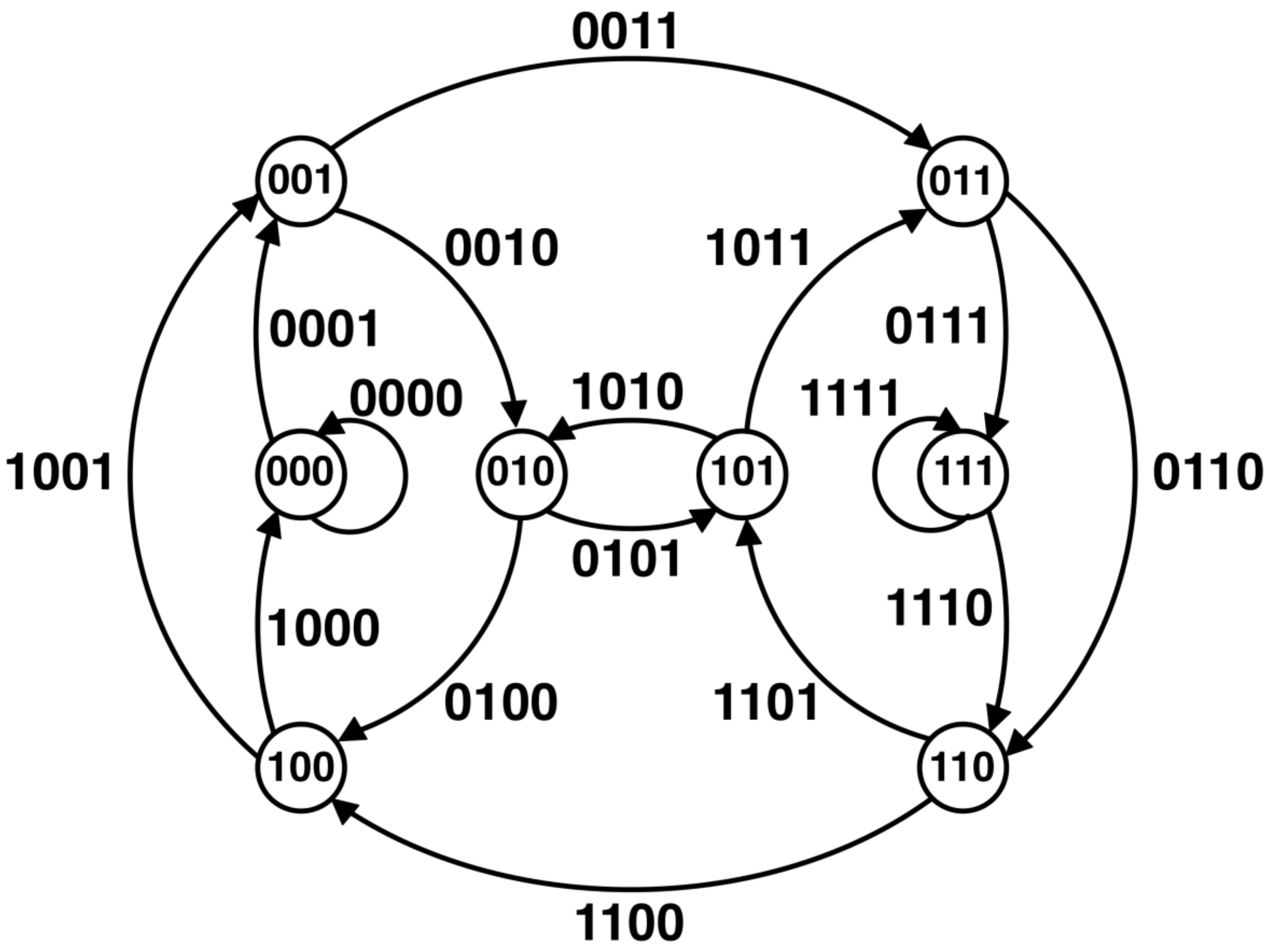


From Hamilton  to Euler to de Bruijn





De Bruijn Graph for 4-Universal String

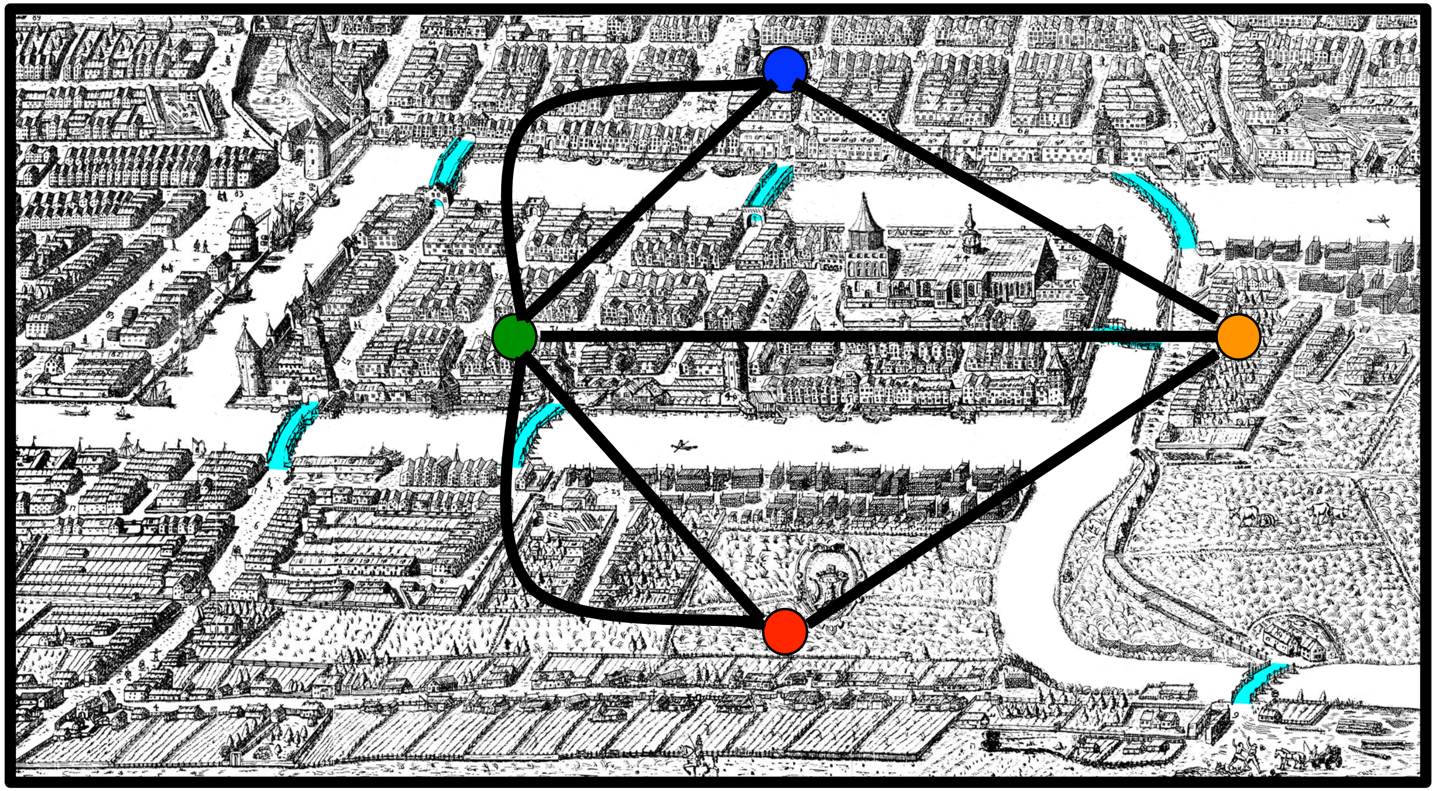


Does it have an Eulerian cycle? If yes, how can we find it?

Eulerian CYCLE Problem

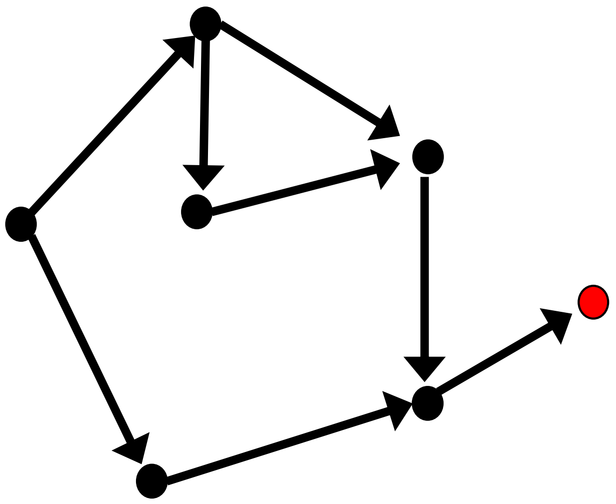
Eulerian CYCLE Problem. Find an Eulerian cycle in a graph.

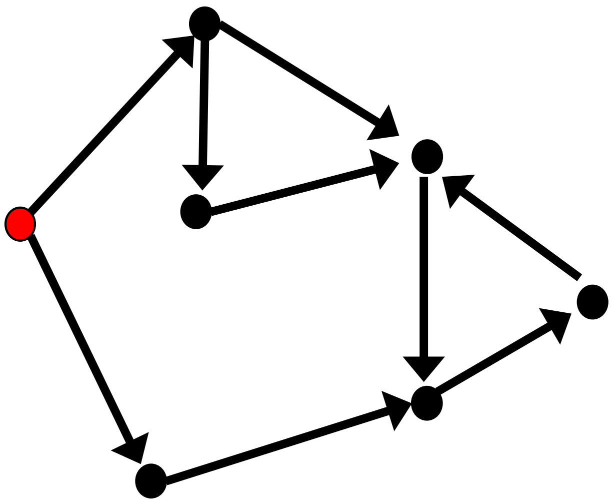
- Input. A graph.
- Output. A cycle visiting every edge in the graph exactly once.

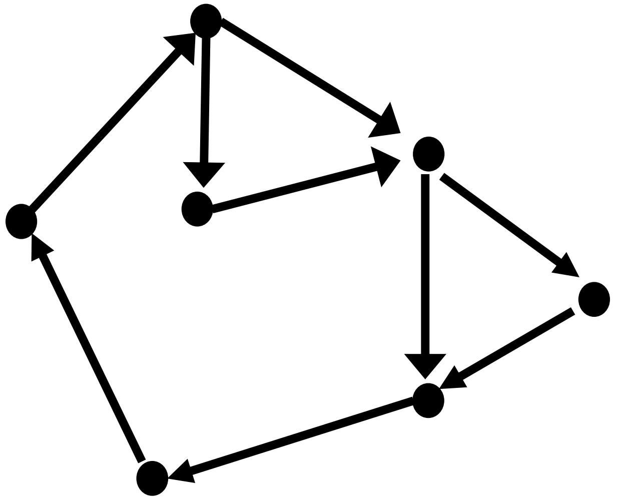


A Graph is **Eulerian** if It Contains an Eulerian Cycle.

Is this graph Eulerian?

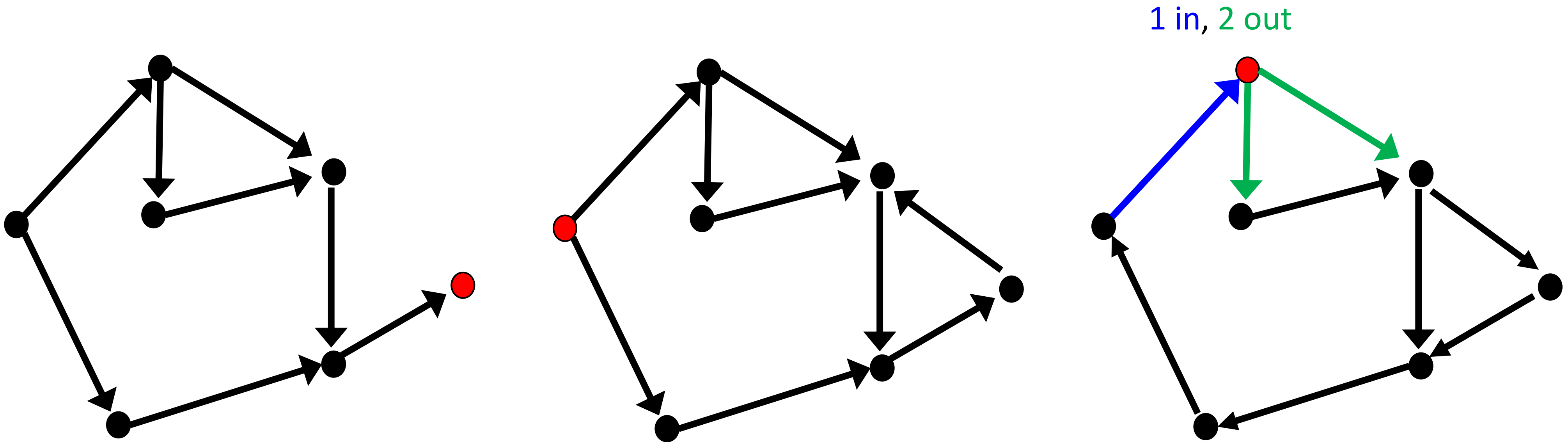






A Graph is **Eulerian** if It Contains an Eulerian Cycle.

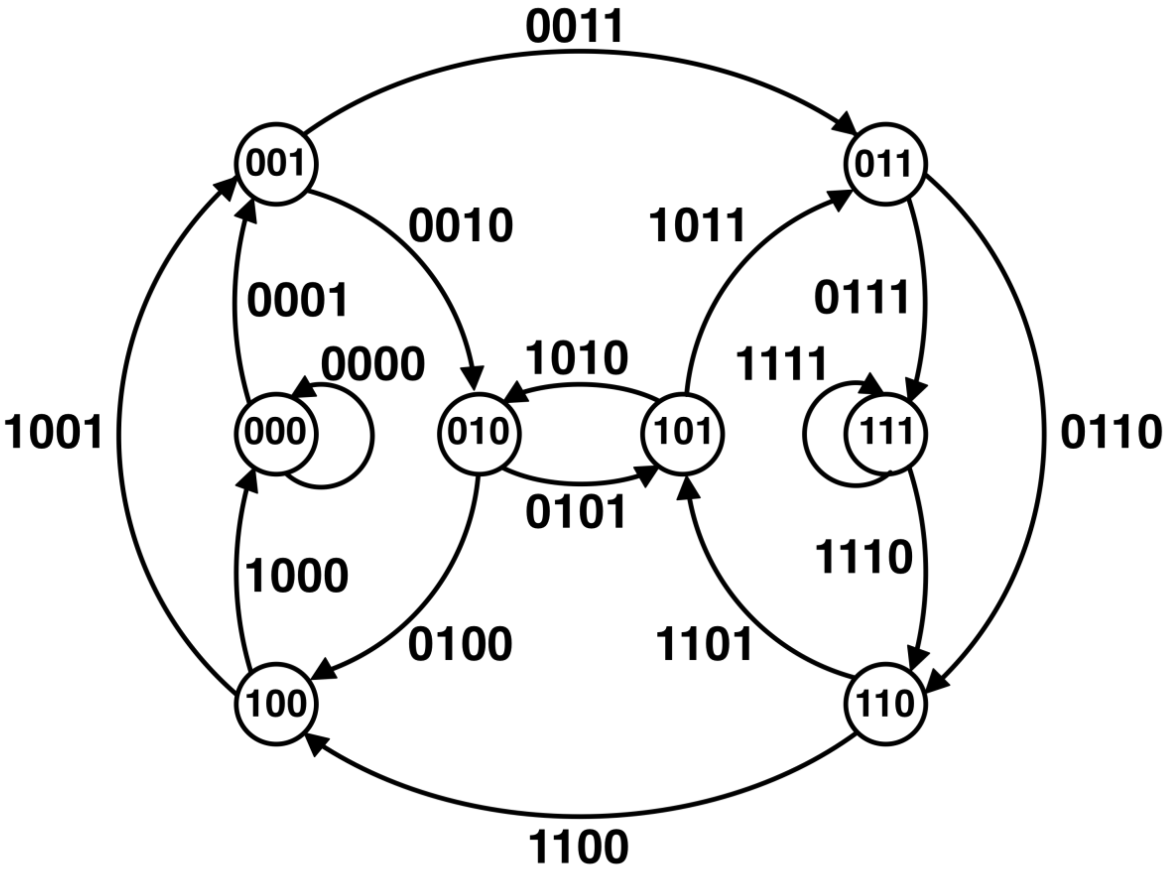
Is this graph Eulerian?



A graph is balanced if indegree = outdegree for each node

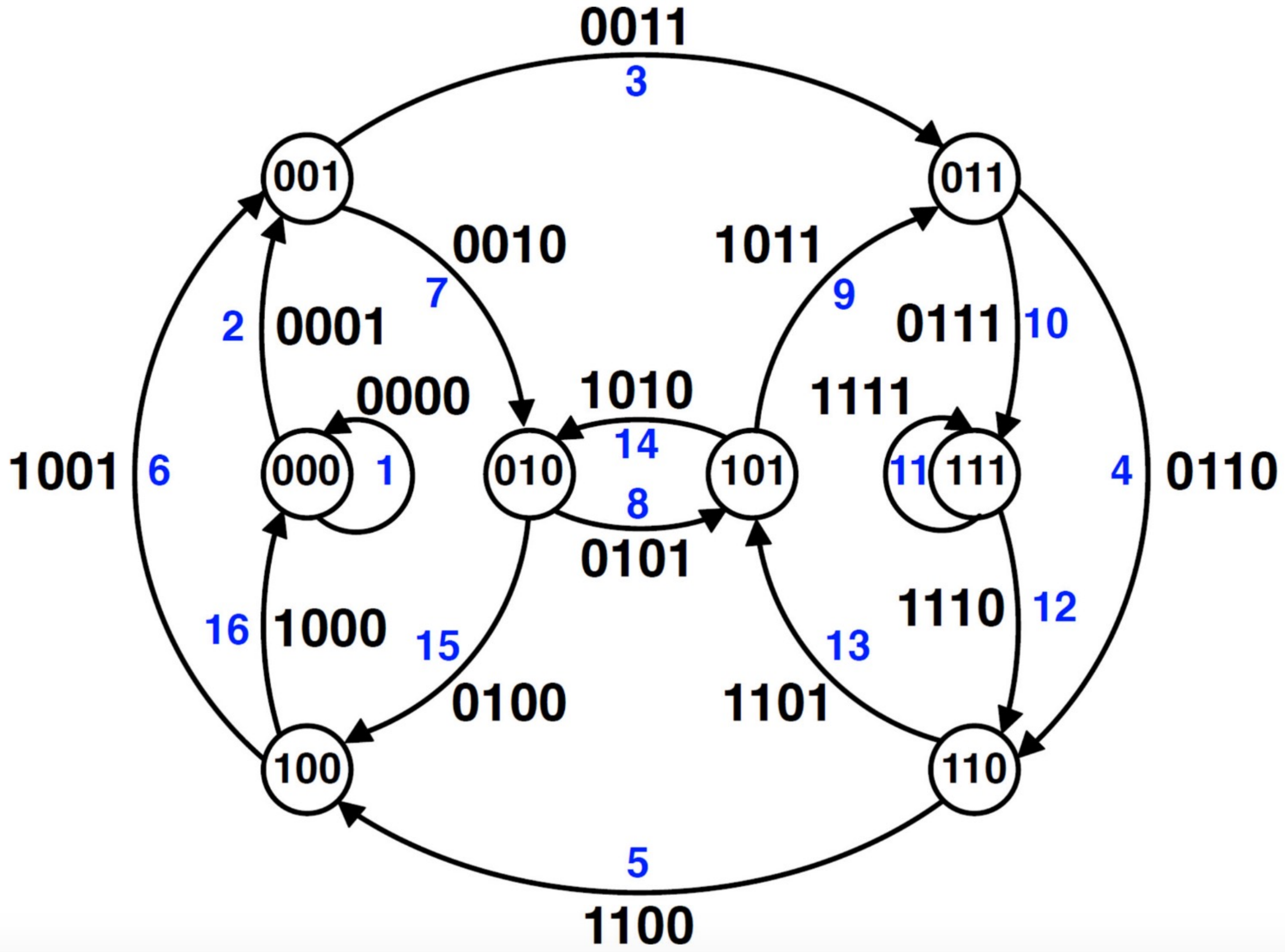
Euler's Theorem

- Every Eulerian graph is balanced
- **Every balanced* graph is Eulerian**



(*) and strongly connected, of course!

Euler's Theorem



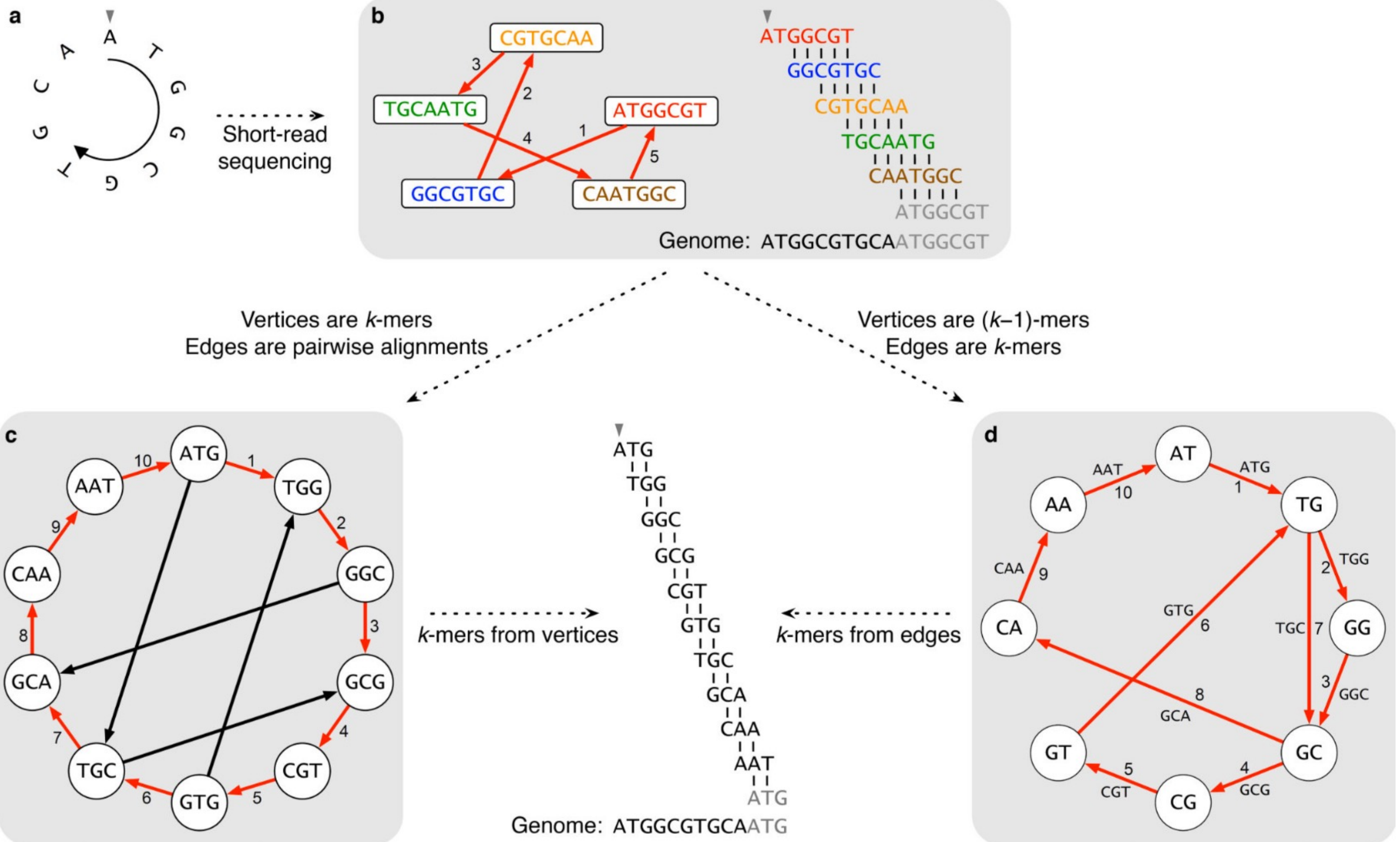
The de Bruijn graph for k = 4 and a 2-character alphabet composed of the digits 0 and 1.

This graph has an Eulerian cycle since each node has indegree and outdegree equal to 2.

Following the blue numbered edges in order 1, 2, ..., 16 gives an Eulerian cycle 0000, 0001, 0011, 0110, 1100, 1001, 0010, 0101, 1011, 0111, 1111, 1110, 1101, 1010, 0100, 1000, which spells the cyclic superstring 0000110010111101

.

Eulerian versus Hamiltonian cycles



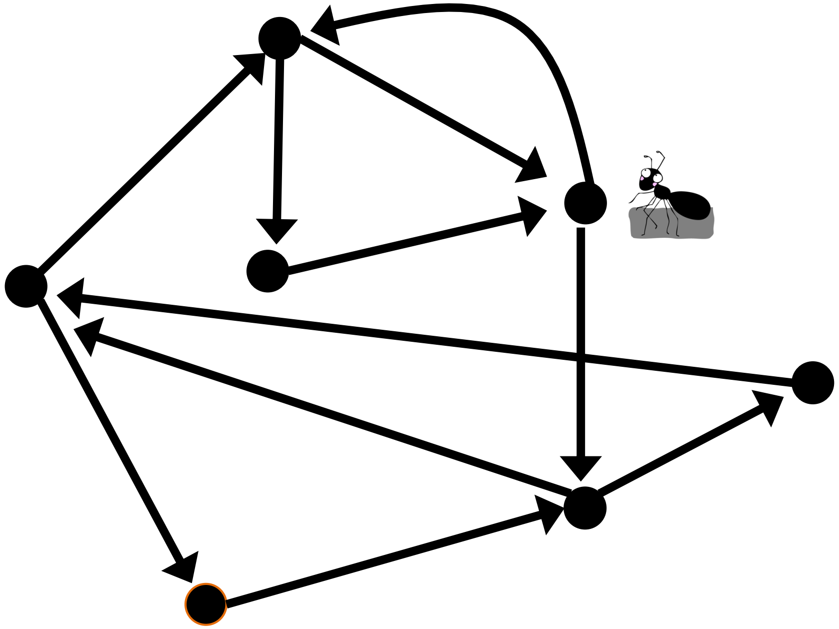
Hamiltonian cycle
Visit each vertex once
(harder to solve)

Eulerian cycle
Visit each edge once
(easier to solve)

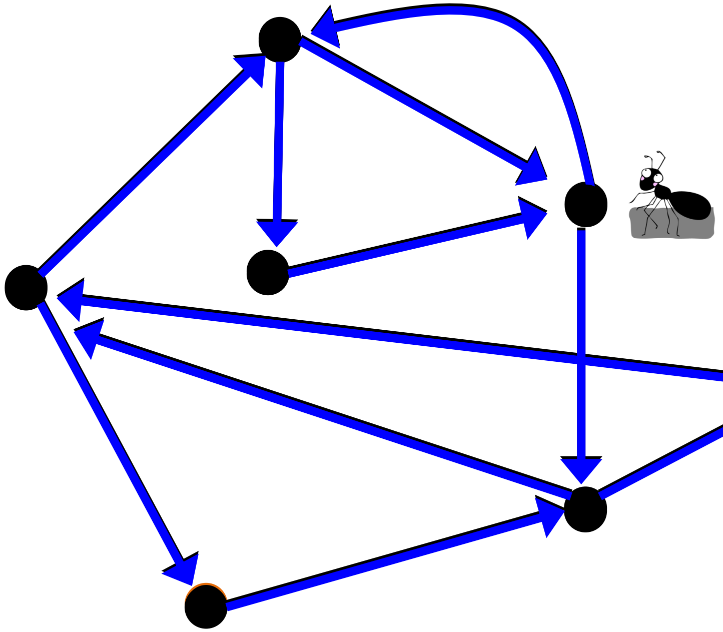
Recruiting an Ant to Prove Euler's Theorem

Let an ant randomly walk through the graph.

The ant cannot use the same edge twice!



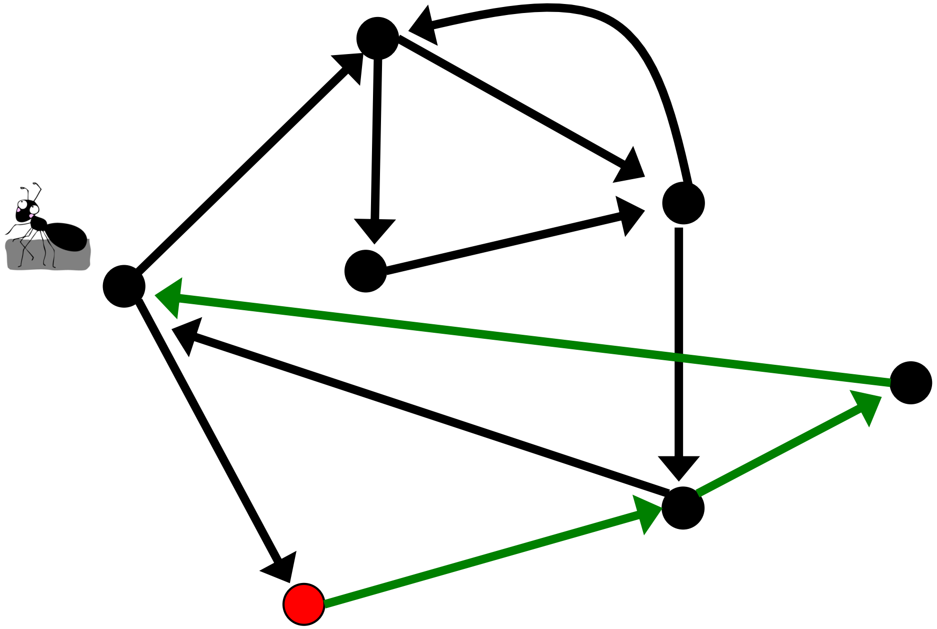
If Ant Was a Genius…



"Yay! Now can I go home please?"

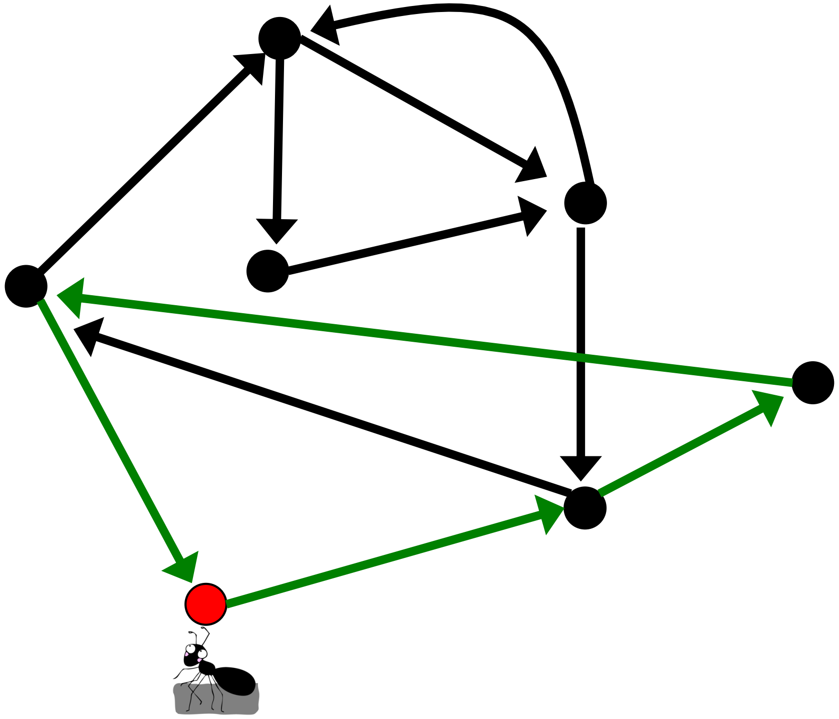
A Less Intelligent Ant Would Randomly Choose a Node and Start Walking…

Can it get stuck? **In what node?**



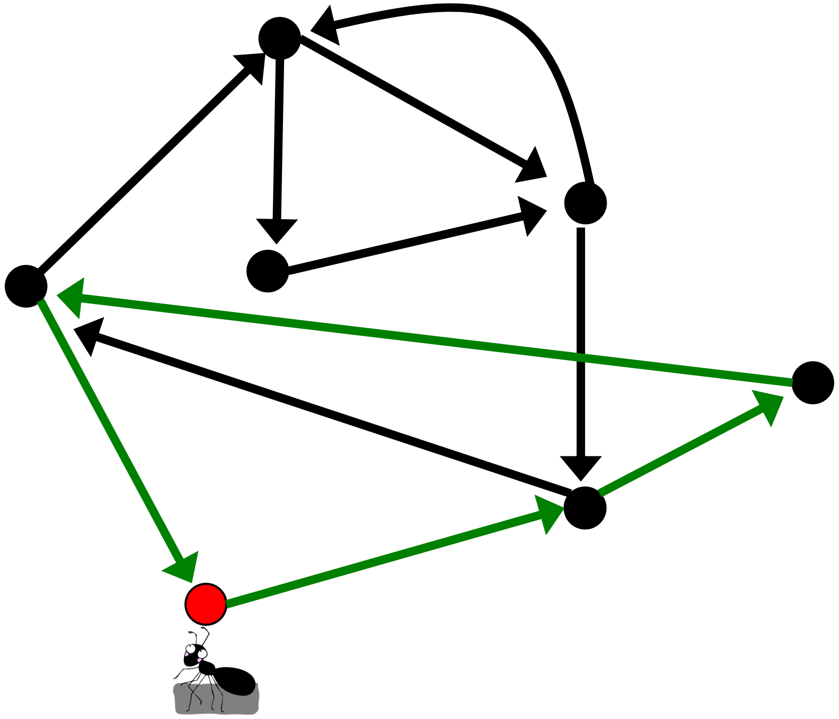
The Ant Has Completed a Cycle BUT has not Proven Euler's theorem yet…

The constructed cycle is not Eulerian. **Can we enlarge it?**



Let’s Start at a Different Node in the Green Cycle

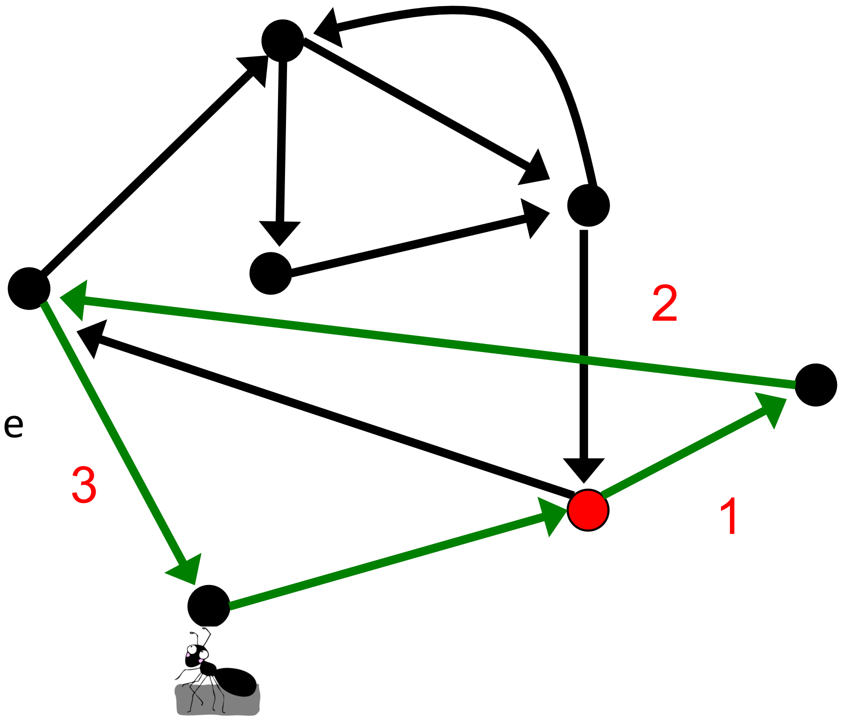
Let’s start at a node with still unexplored edges.



“Why should I start at a different node? Backtracking? I’m not evolved to walk backwards! And what difference does it make???”

An Ant Traversing Previously Constructed Cycle

Starting at a node that has an unused edge, traverse the already constructed (green cycle) and return back to the starting node.

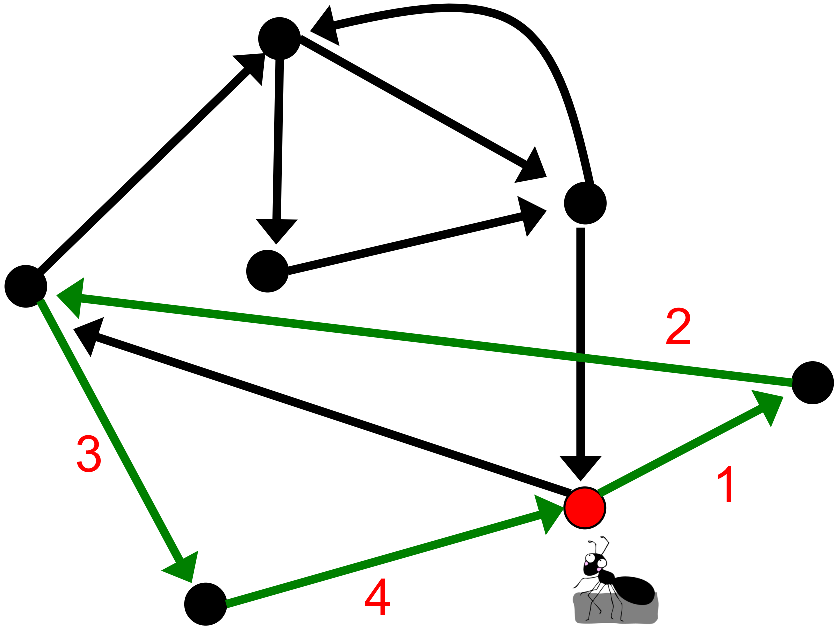


“Why do I have to walk along the same cycle again??? Can I see something new?”

I Returned Back BUT… I Can Continue Walking!

Starting at a node that has an unused edge, traverse the already constructed (green cycle) and return back to the starting node.

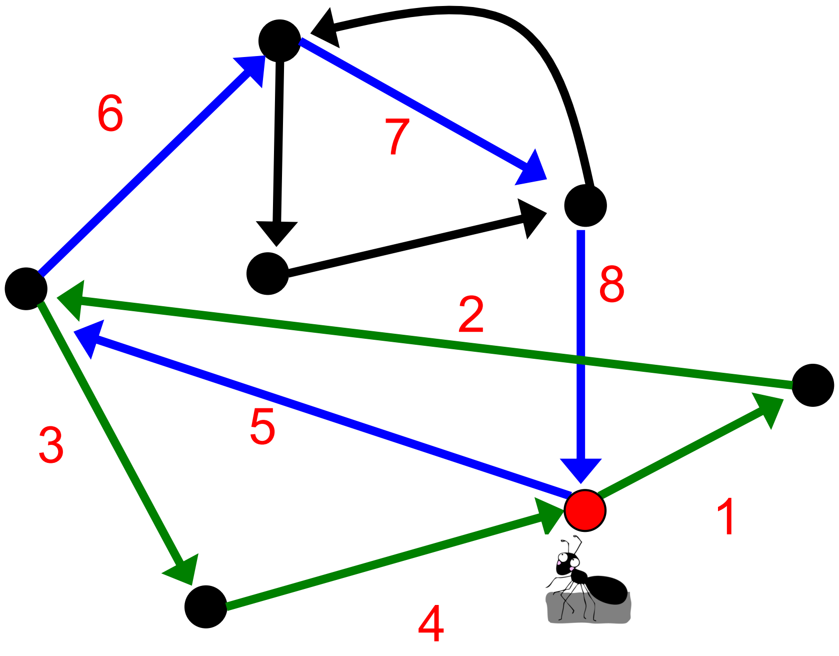
After completing the cycle, start random exploration of still untraversed edges in the graph.



Stuck Again!

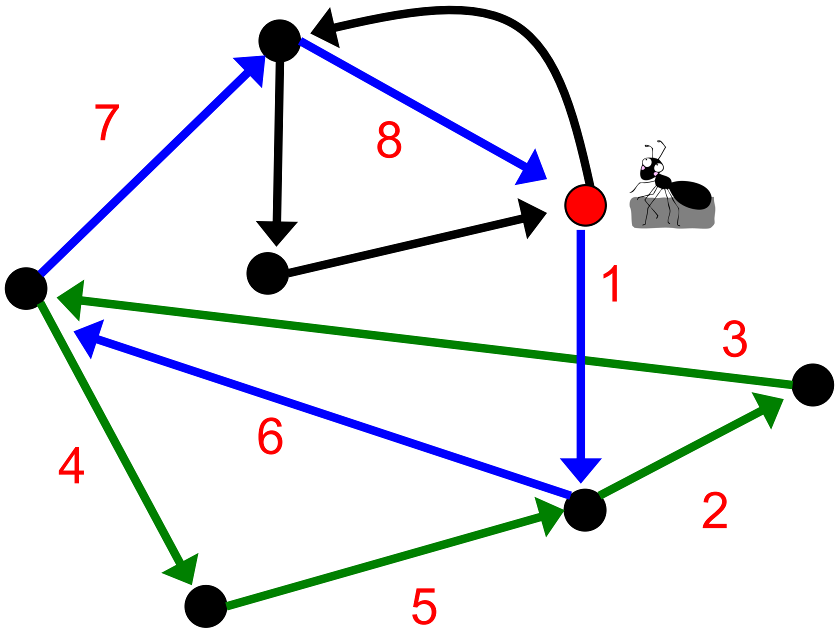
No Eulerian cycle yet… can we enlarge the green-blue cycle?

The ant should walk along the constructed cycle starting at yet another node. Which one?



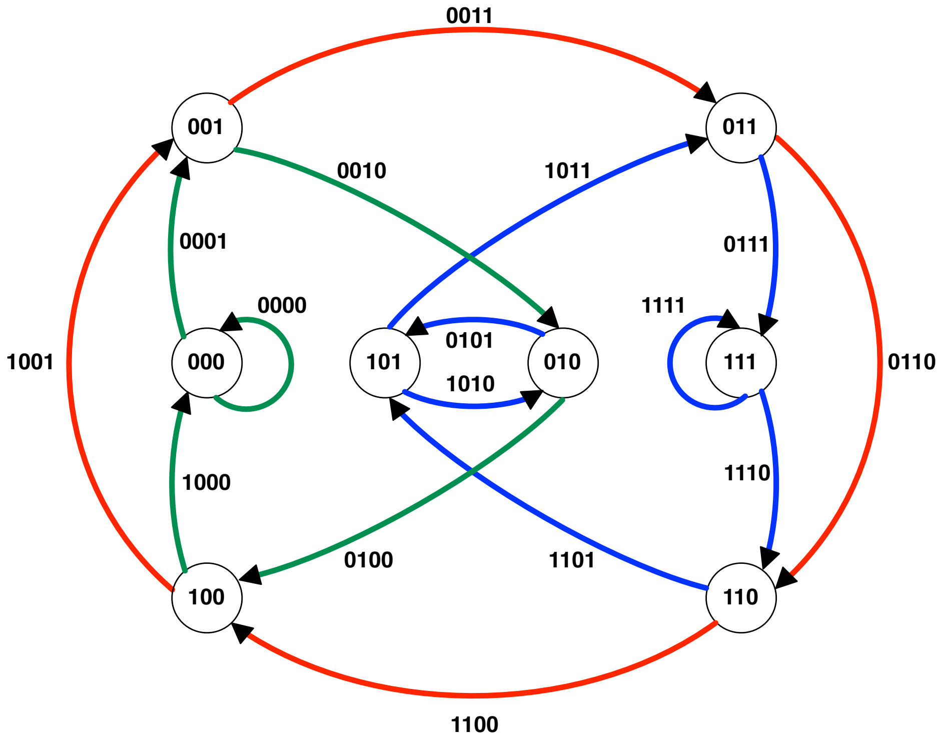
I Returned Back BUT… I Can Continue Walking!

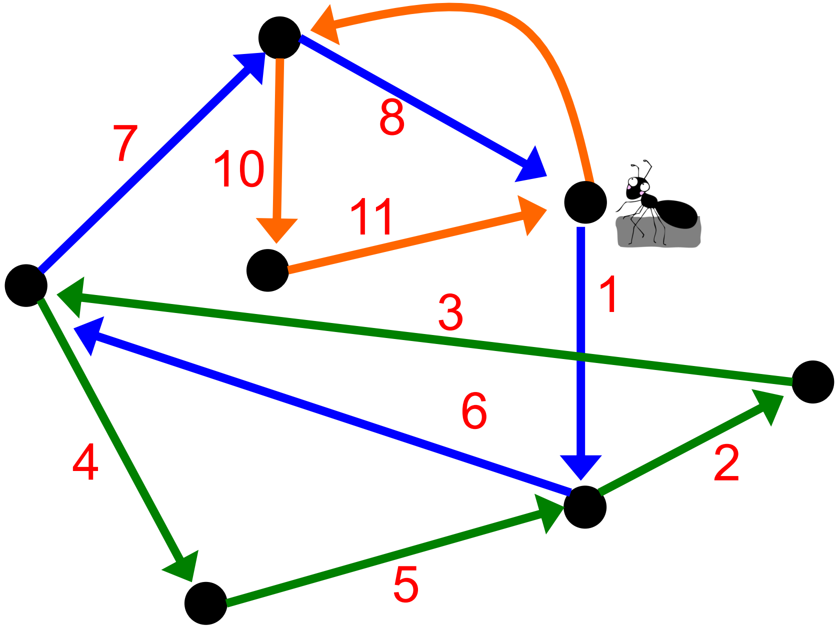
“Hmm, maybe these instructions were not that stupid…”



I Proved Euler's Theorem!

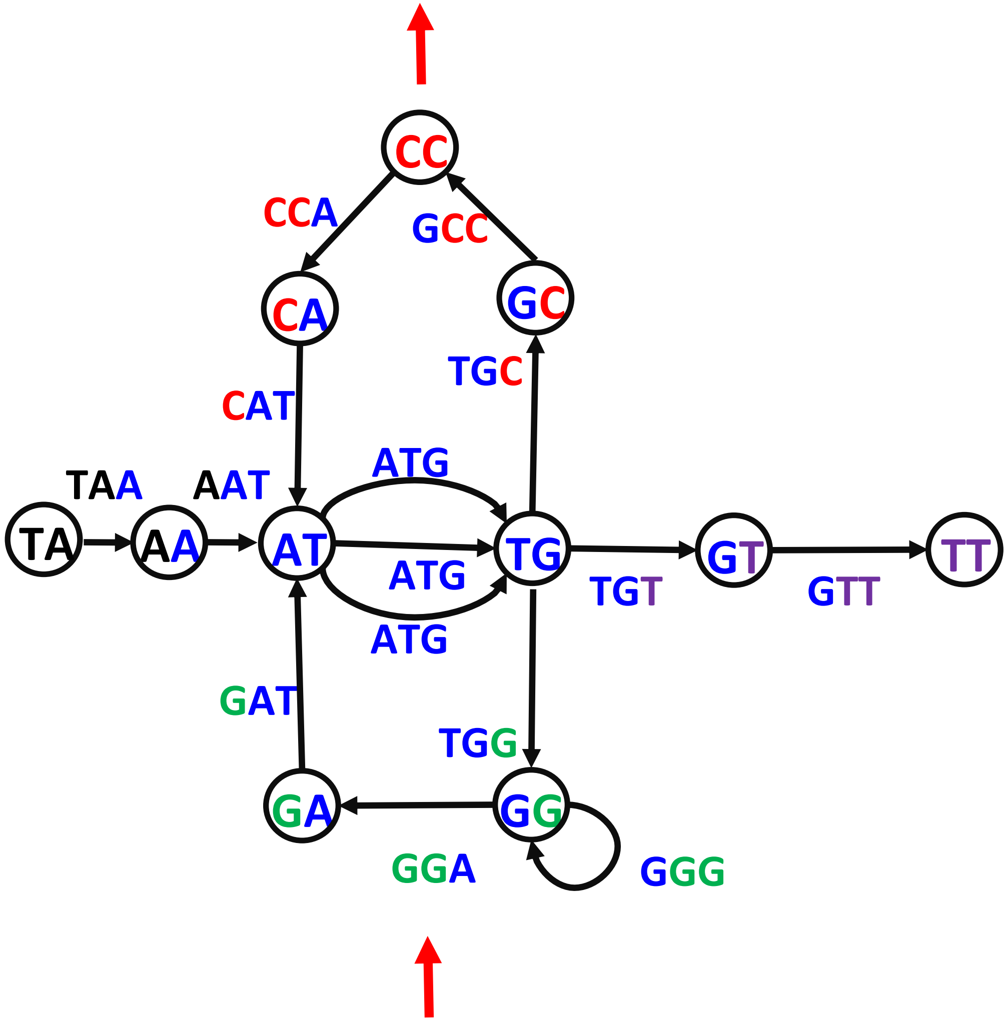
```
**EulerianCycle**(Balanced*Graph*)
  form a *Cycle* by randomly walking in *BalancedGraph* (avoiding already visited edges)
    **while** *Cycle* is not Eulerian
      select a node newStart  in Cycle with still unexplored outgoing edges
      form a *Cycle'* by traversing *Cycle* from newStart and randomly walking afterwards
      *Cycle* ← *Cycle'*
  **return** *Cycle*
```





From Reads to de Bruijn Graph to Genome

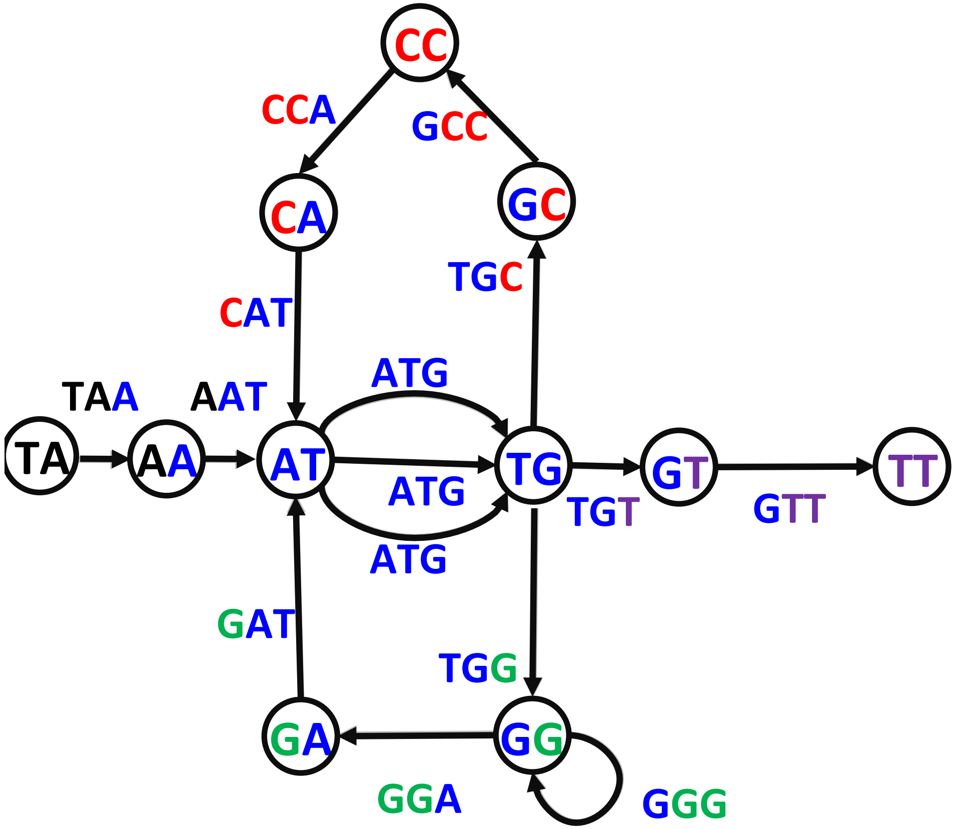
TAATGCCATGGGATGTT



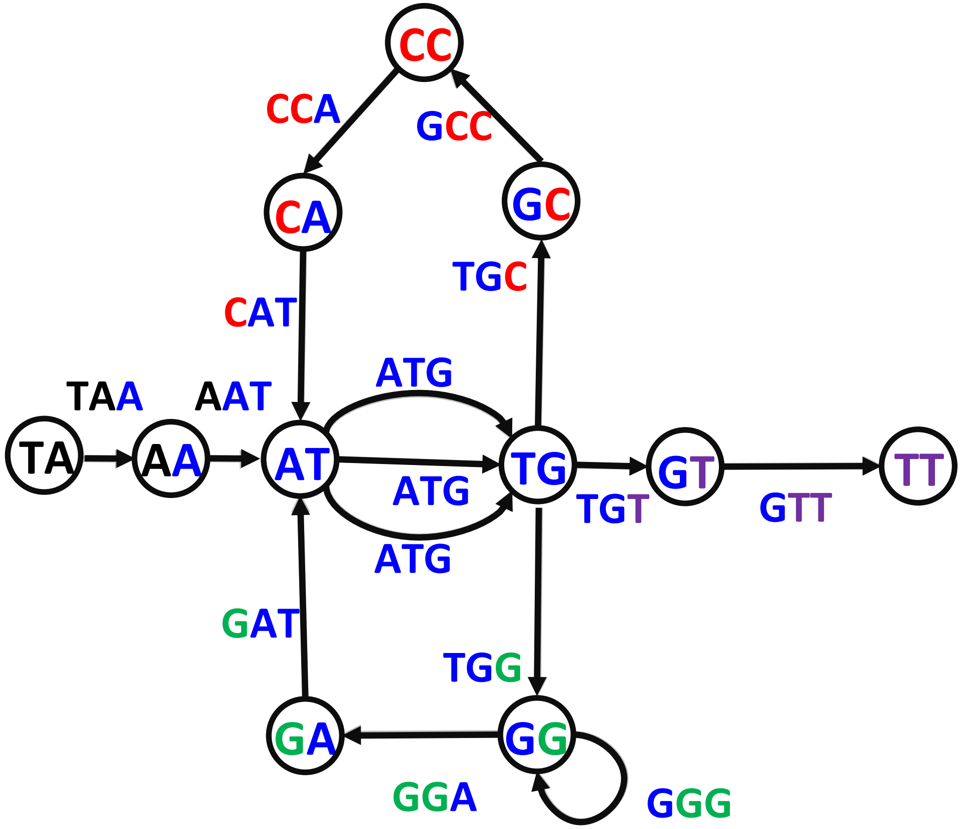
AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

Multiple Eulerian Paths

TAATGCCATGGGATGTT

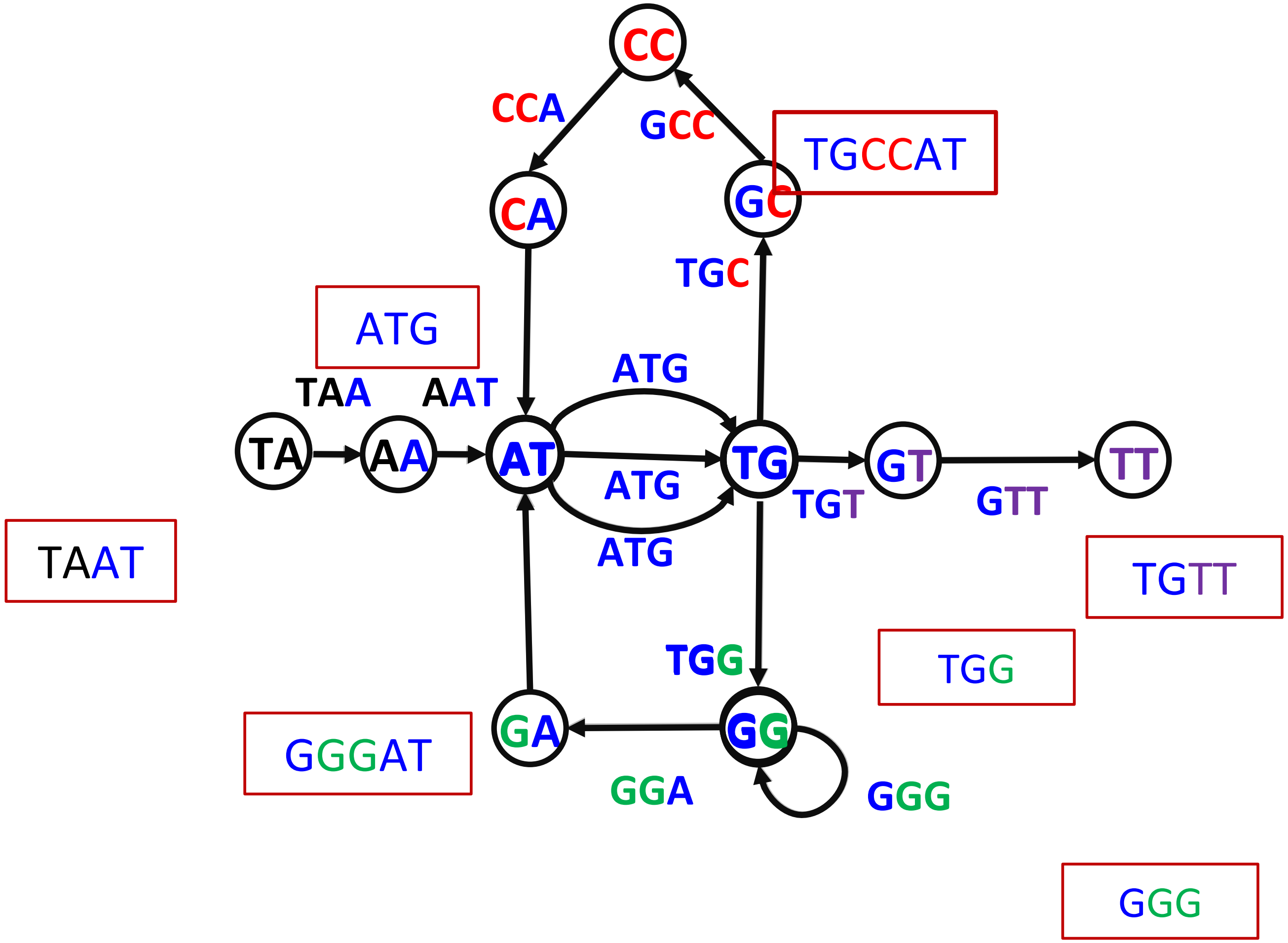


TAATGGGATGCCATGTT

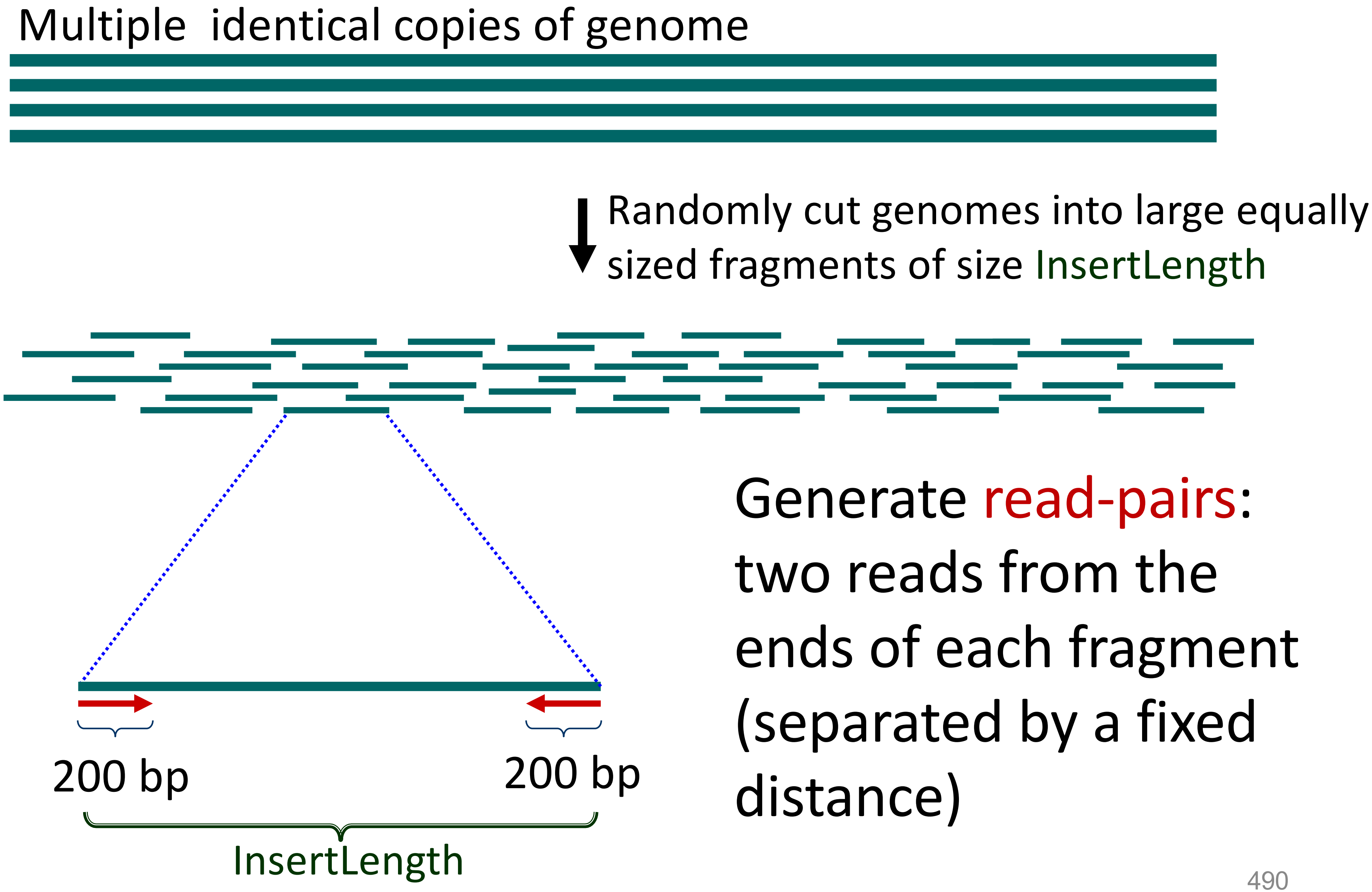


Breaking Genome into Contigs

TAATGCCATGGGATGTT



DNA Sequencing with Read-pairs



Multiple identical copies of genome

Randomly cut genomes into large equally sized fragments of size InsertLength

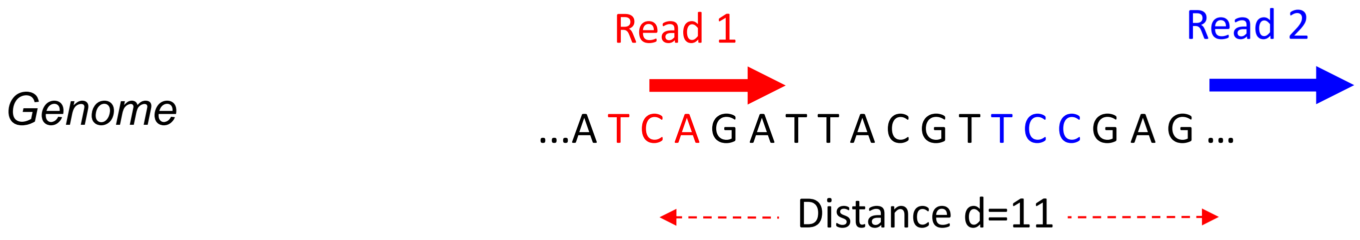
200 bp

200 bp

InsertLength

Generate read-pairs: two reads from the ends of each fragment (separated by a fixed distance)

From *k*-mers to Paired *k*-mers



Read 1

Read 2

Genome

...A T C A G A T T A C G T T C C G A G ...

Distance d=11

A paired k-mer is a pair of k-mers at a fixed distance d apart in Genome.
E.g. TCA and TCC are at distance d=11 apart.

Disclaimers:

1. In reality, Read1 and Read2 are typically sampled from different strands:

(→ ……. ← rather than → ……. →)

2. In reality, the distance d between reads is measured with errors.

What is PairedComposition(TAATGCCATGGGATGTT)?

TAA GCC
AAT CCA
ATG CAT
TGC ATG
GCC TGG
CCA GGG
CAT GGA
ATG GAT
TGG ATG
GGG TGT
GGA GTT

Representing a paired 3-mer TAA GCC as a 2-line expression:

TAA
GCC

TAA	AAT	ATG	TGC	GCC	CCA	CAT	ATG	TGG	GGG	GGA
GCC	CCA	CAT	ATG	TGG	GGG	GGA	GAT	ATG	TGT	GTT

PairedComposition(TAATGCCATGGATGTT)

```
TAA GCC
 AAT CCA
  ATG CAT
   TGC ATG
    GCC TGG
     CCA GGG
      CAT GGA
       ATG GAT
        TGG ATG
         GGG TGT
          GGA GTT
```

```
TAA  AAT  ATG  TGC  GCC  CCA  CAT  ATG  TGG  GGG  GGA
GCC  CCA  CAT  ATG  TGG  GGG  GGA  GAT  ATG  TGT  GTT

AAT  ATG  ATG  CAT  CCA  GCC  GGA  GGG  TAA  TGC  TGG
CCA  CAT  GAT  GGA  GGG  TGG  GTT  TGT  GCC  ATG  ATG
```

Representing PairedComposition in lexicographic order

String Reconstruction from Read-Pairs Problem

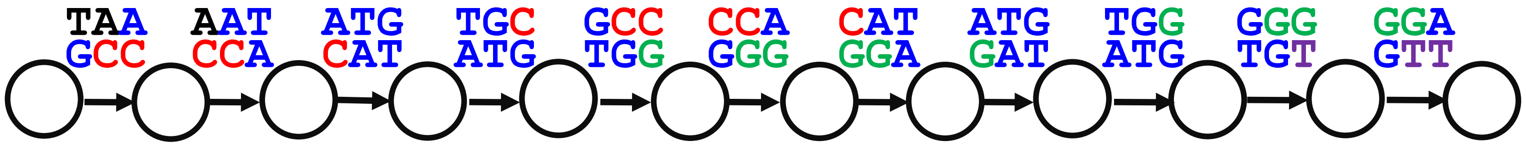
String Reconstruction from Read-Pairs Problem. Reconstruct a string from its paired *k*-mers.

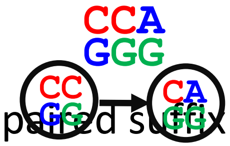
- **Input.** A collection of paired *k*-mers.
- **Output.** A string *Text* such that *PairedComposition*(*Text*) is equal to the collection of paired k-mers.

How Would de Bruijn Assemble Paired *k*-mers?

Representing Genome TAATGCCATGGGATGTT as a Path

TAA GCC
AAT CCA
ATG CAT
TGC ATG
GCC TGG
CCA GGG
CAT GGA
ATG GAT
TGG ATG
GGG TGT
GGA GTT

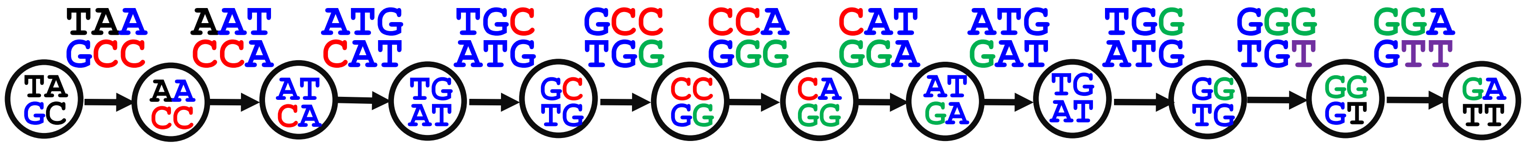


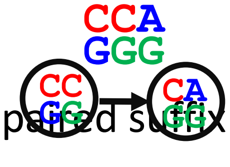


paired prefix of ⟶ CCA
GGG ← paired suffix of

CCA
GGG

Labeling Nodes by Paired Prefixes and Suffixes

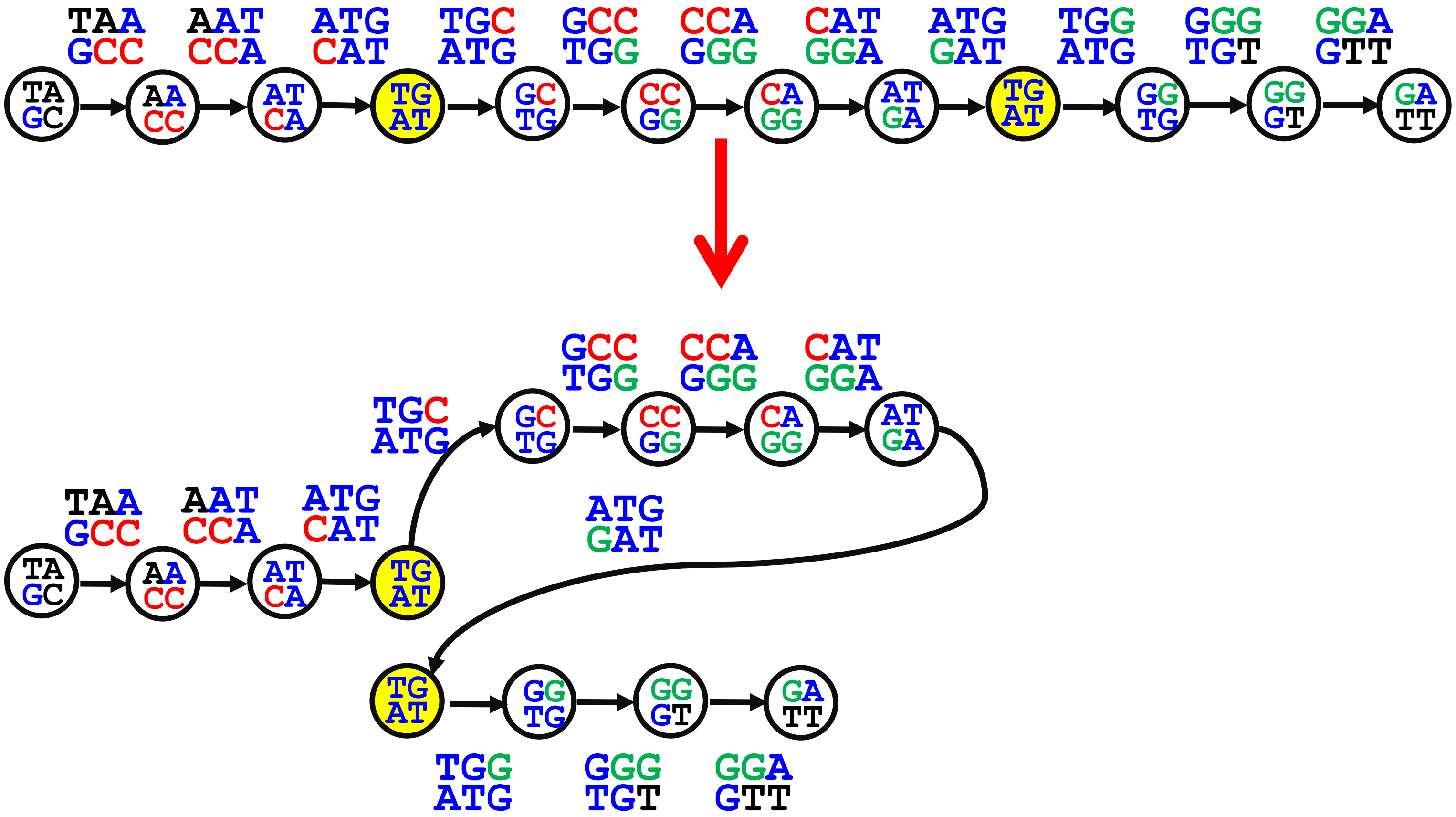




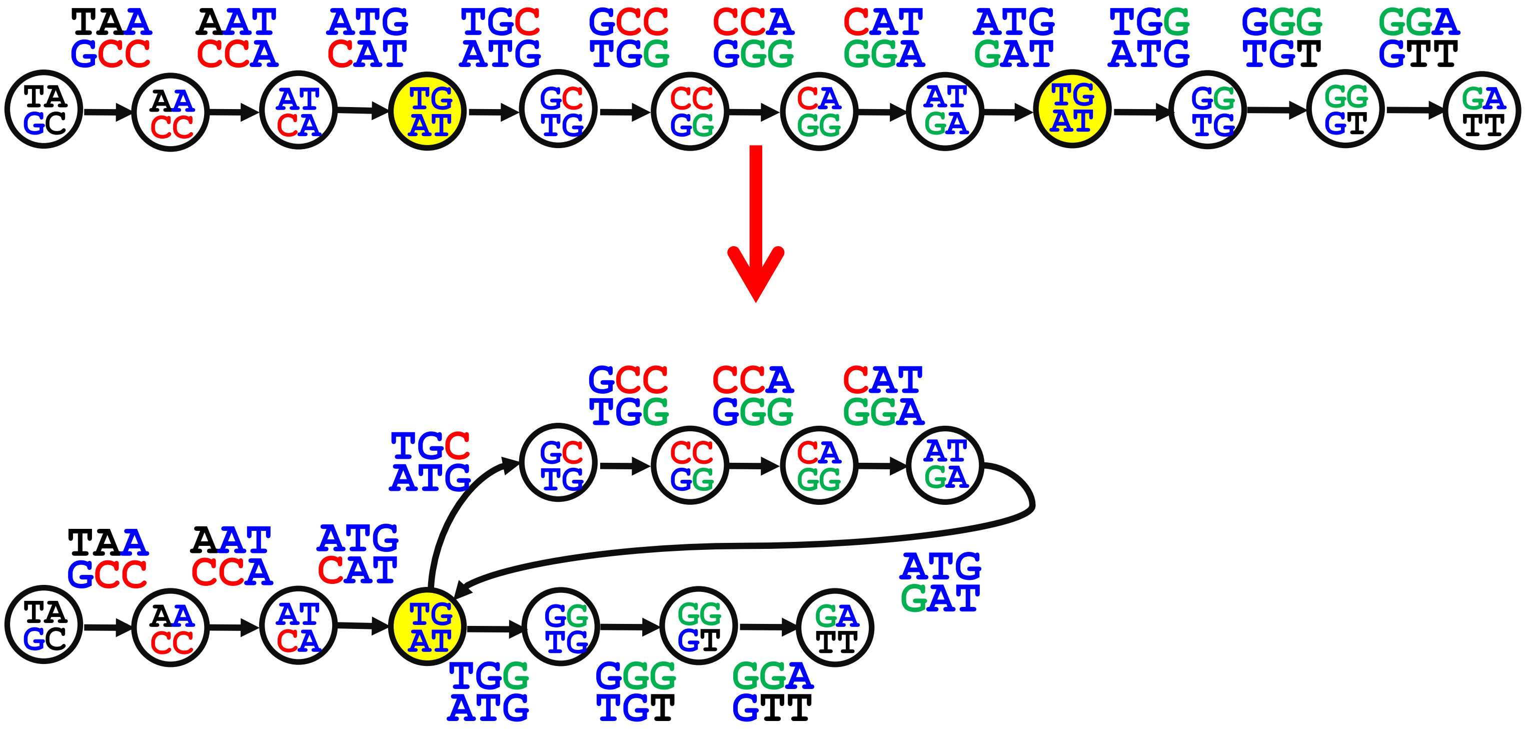
paired prefix of ⟶ CCA GGG ← paired suffix of

CCA GGG

Glue nodes with identical labels

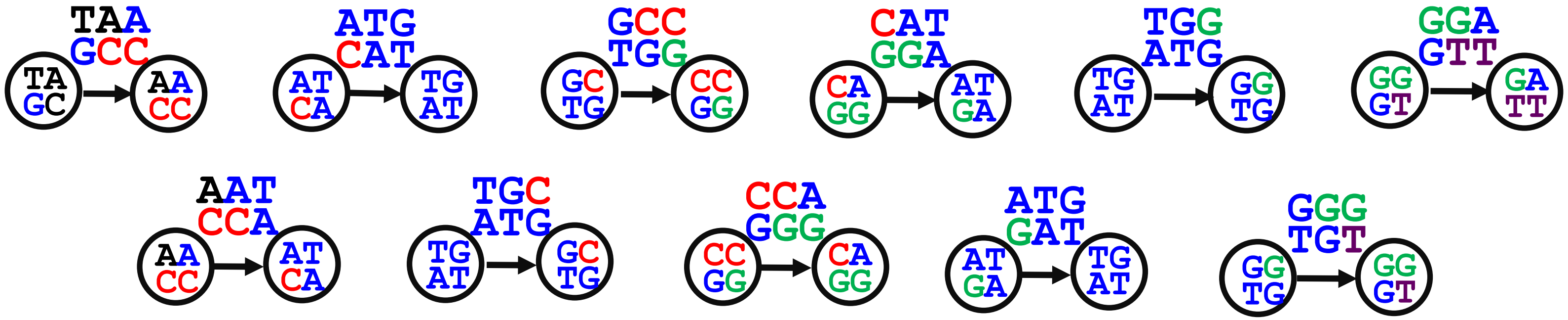


Glue nodes with identical labels

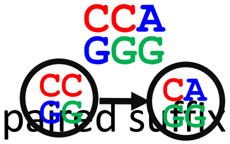


Paired de Bruijn Graph from the Genome

Constructing Paired de Bruijn Graph

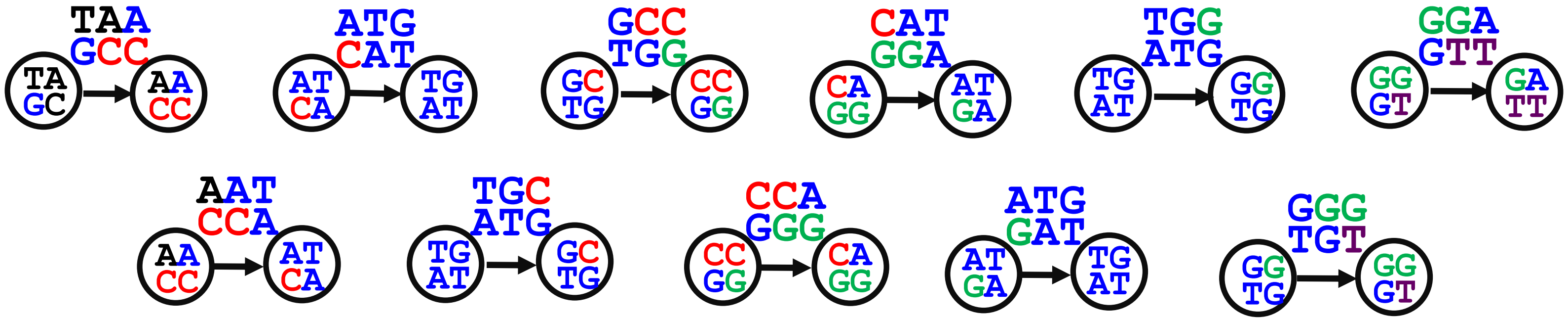


paired prefix of ⟶ CCA
GGG
← paired suffix of



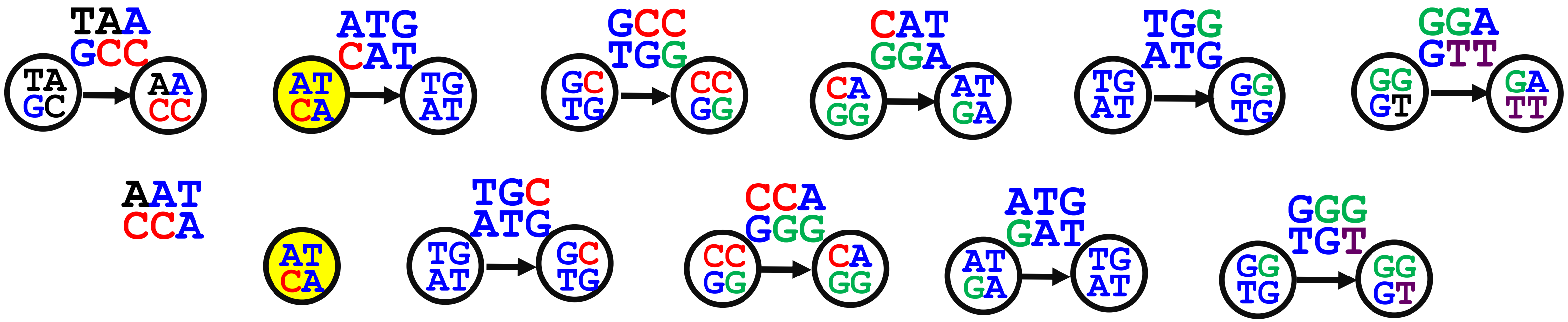
CCA
GGG

Constructing Paired de Bruijn Graph

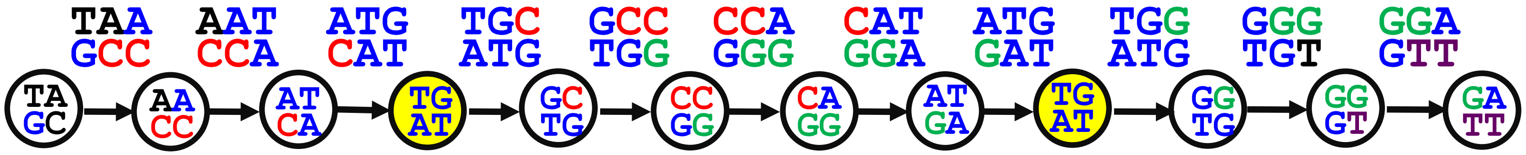


- **Paired de Bruijn graph for a collection of paired *k*-mers:**
  - Represent every paired *k*-mer as an edge between its paired prefix and paired suffix.
  - Glue **ALL** nodes with identical labels.

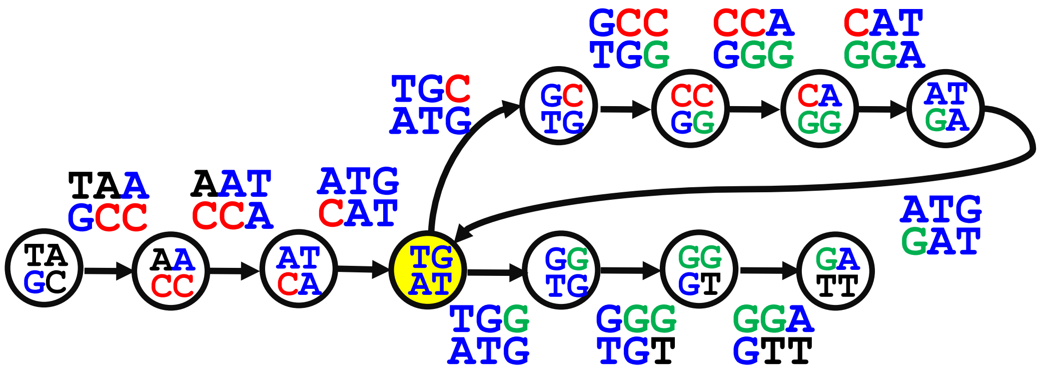
Constructing Paired de Bruijn Graph



We Are Not Done with Gluing Yet



Constructing Paired de Bruijn Graph



Paired de Bruijn Graph from read-pairs

- **Paired de Bruijn graph for a collection of paired *k*-mers:**
  – Represent every paired *k*-mer as an edge between its paired prefix and paired suffix.
  – Glue **ALL** nodes with identical labels.

Which Graph Represents a Better Assembly?

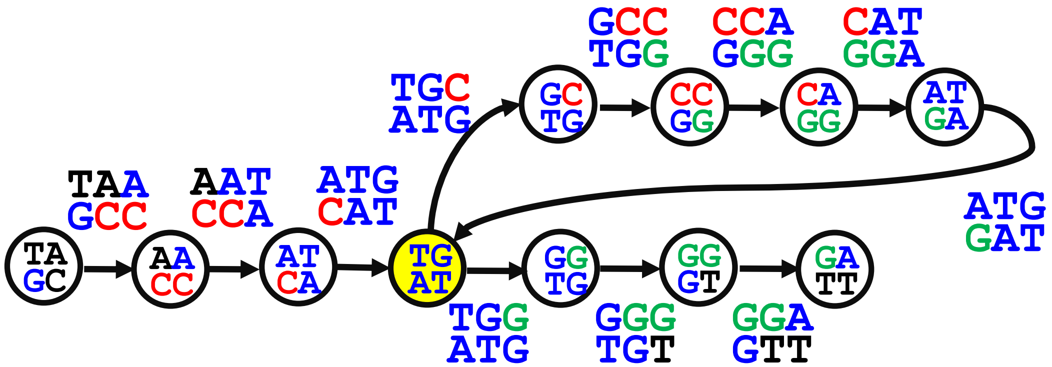
Unique genome reconstruction

TAATGCCATGGGATGTT

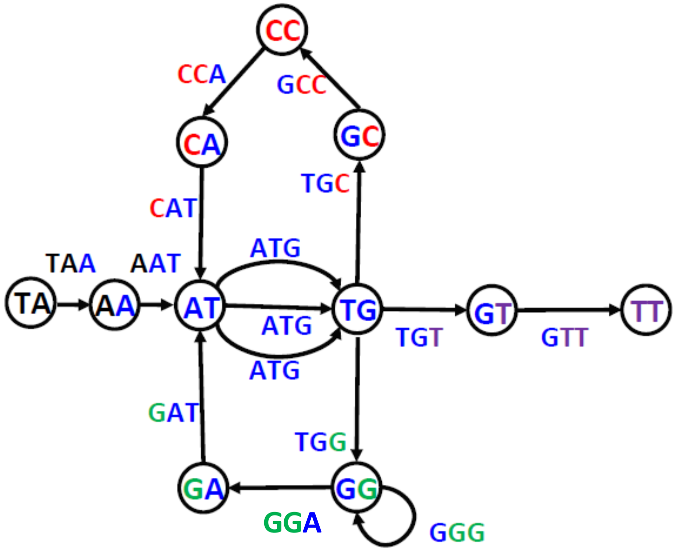
Multiple genome reconstructions

TAATGCCATGGGATGTT

TAATGGGATGCCATGTT



Paired de Bruijn Graph



De Bruijn Graph

Some Ridiculously Unrealistic Assumptions

- Perfect coverage of genome by reads (every *k*-mer from the genome is represented by a read)

- Reads are error-free.

- Multiplicities of *k*-mers are known

- Distances between reads within read-pairs are exact.

Some Ridiculously Unrealistic Assumptions

- **Imperfect** coverage of genome by reads (every *k*-mer from the genome is represented by a read)

- Reads are **error-prone**.

- Multiplicities of *k*-mers are **unknown.**

- Distances between reads within read-pairs are **inexact.**

- **Etc., etc., etc.**

1st Unrealistic Assumption: Perfect Coverage

```
atgccgtatggacaacgact
atgccgtatg
  gccgtatgga
     gtatggacaa
          gacaacgact
```

250-nucleotide reads generated by Illumina technology capture only a small fraction of 250-mers from the genome, thus violating the key assumption of the de Bruijn graphs.

Breaking Reads into Shorter *k*-mers

```
atgccgtatggacaacgact     atgccgtatggacaacgact
atgccgtatg               atgcc
  gccgtatgga              tgccg
     gtatggacaa            gccgt
          gacaacgact        ccgta
                             cgtat
                              gtatg
                               tatgg
                                atgga
                                 tggac
                                  ggaca
                                   gacaa
                                    acaac
                                     caacg
                                      aacga
                                       acgac
                                        cgact
```

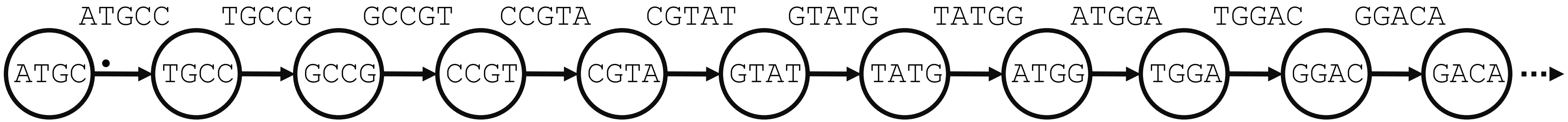
2nd Unrealistic Assumption: Error-free Reads

```
atgccgtatggacaacgact
atgccgtatg
  gccgtatgga
     gtatggacaa
          gacaacgact
    cgtaCggaca
```

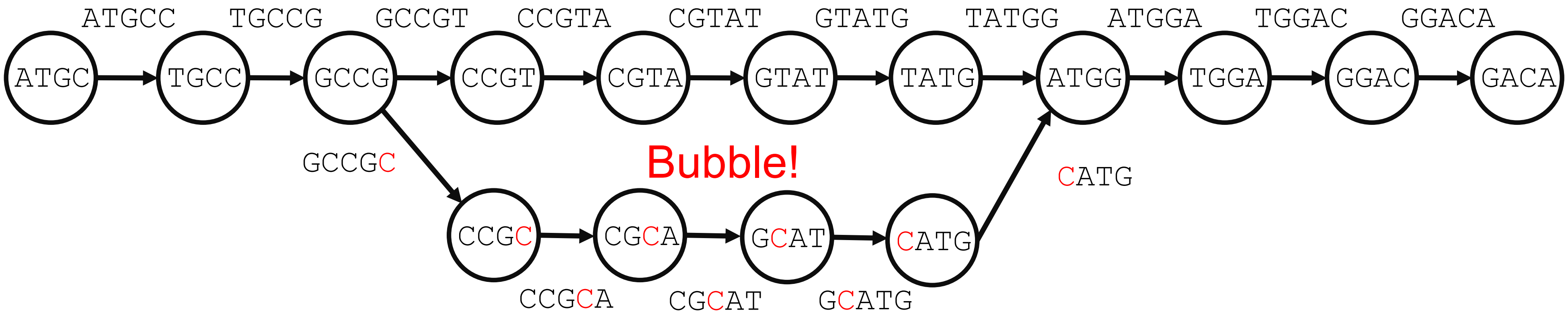
Erroneous read
(change of t into C)

```
atgccgtatggacaacgact
atgcc
 tgccg
  gccgt
   ccgta
    cgtat
     gtatg
      tatgg
       atgga
        tggac
         ggaca
          gacaa
           acaac
            caacg
             aacga
              acgac
               cgact
    cgtaC
     gtaCg
      taCgg
       aCgga
        Cggac
```

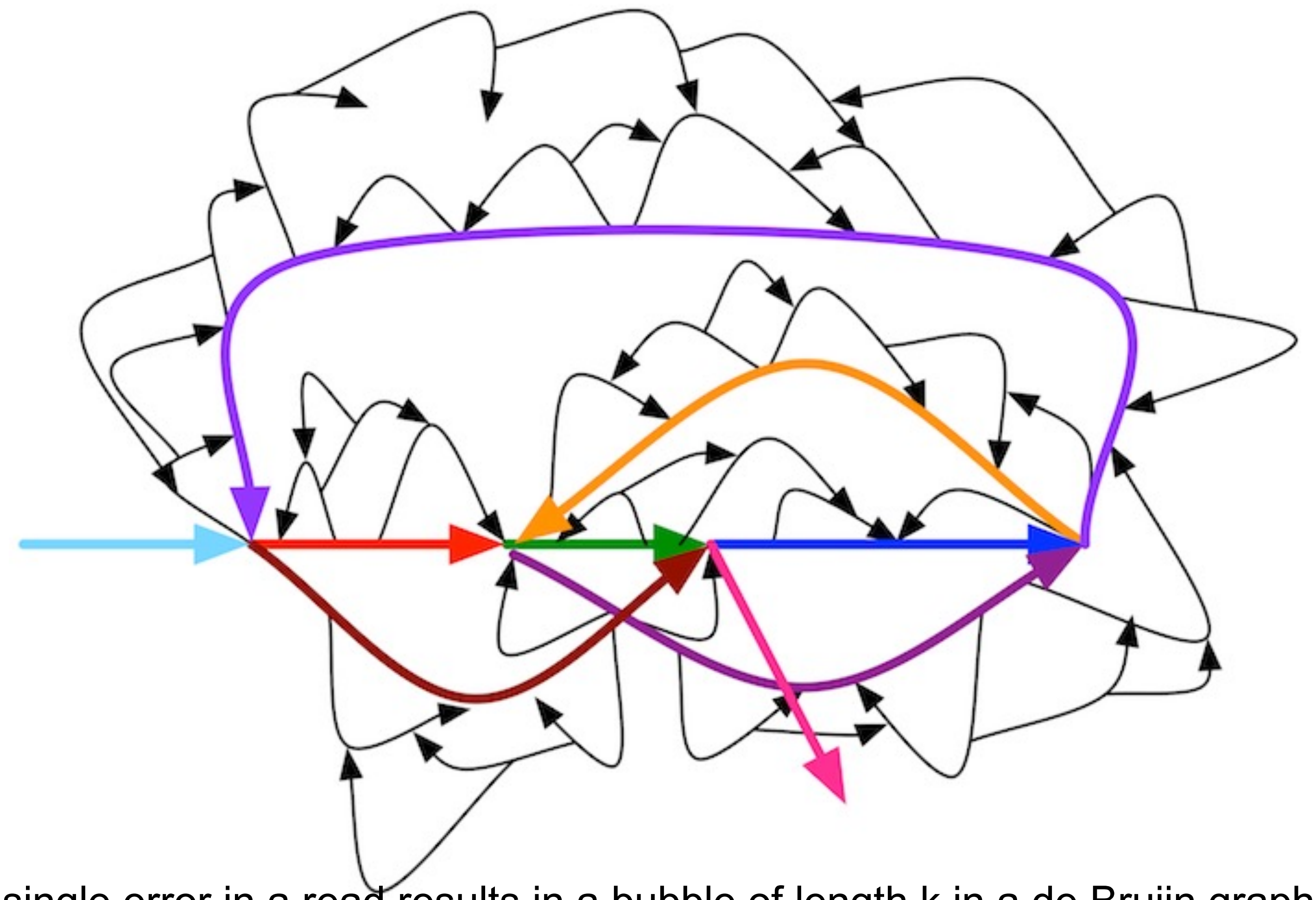
De Bruijn Graph of ATGGCGTGCAATG…
Constructed from Error-Free Reads



Errors in Reads Lead to **Bubbles** in the De Bruijn Graph

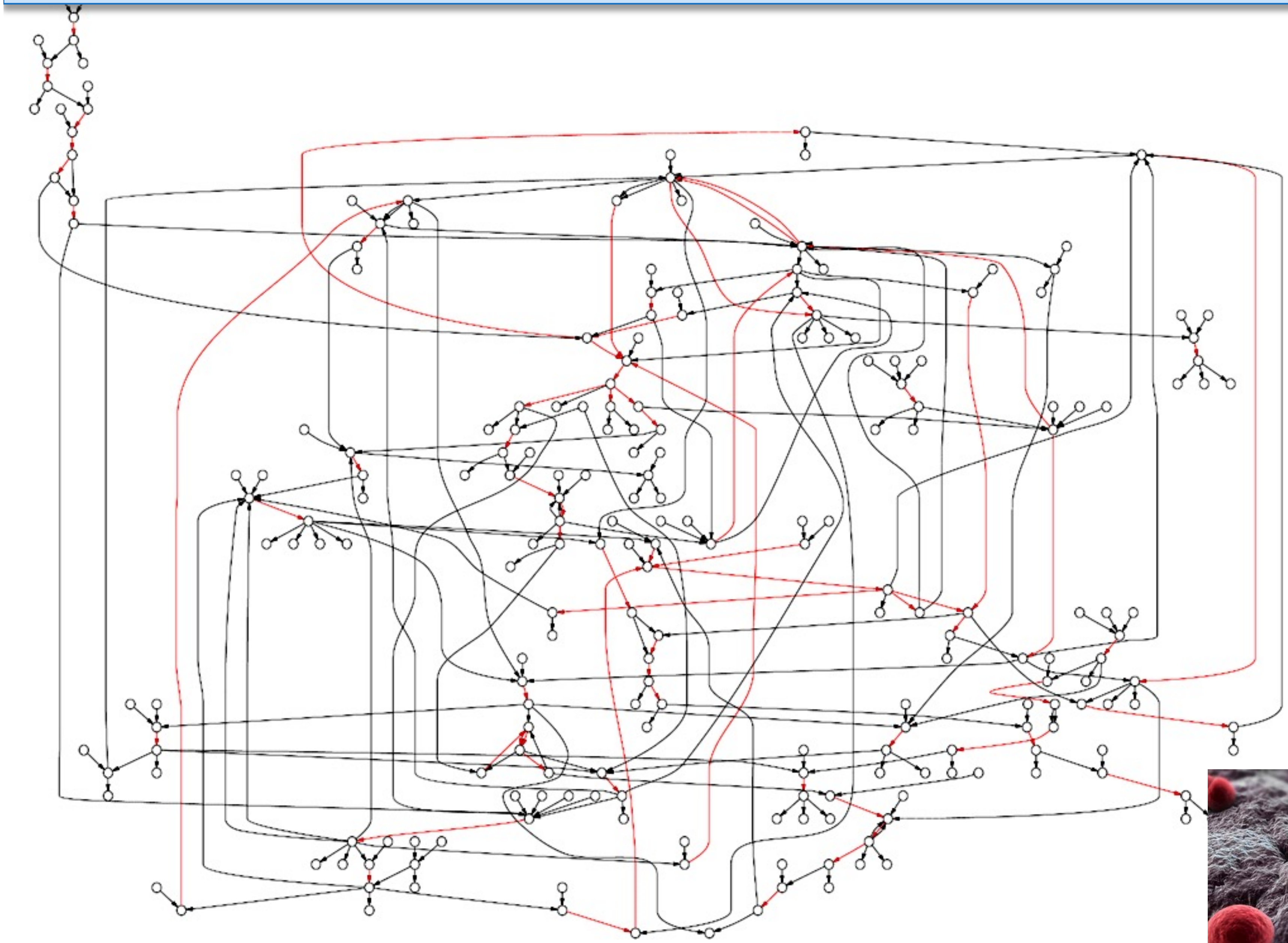


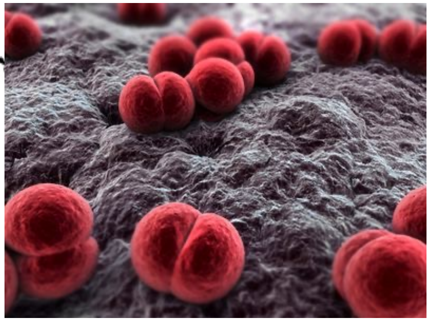
Bubble Explosion



A single error in a read results in a bubble of length k in a de Bruijn graph constructed from k-mers. Multiple errors in various reads may form longer bubbles, but since the error rate in reads is rather small (less than 1% per nucleotide in Illumina reads), most bubbles are small.

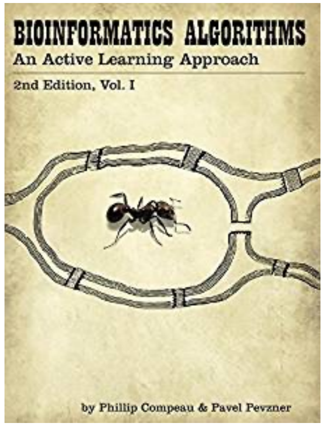
Example Results: De Bruin Graph of *N. meningitidis* Genome AFTER Removing Bubbles

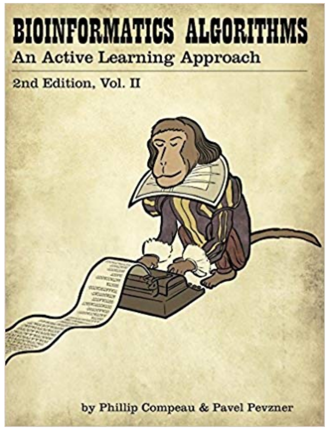




Red edges represent repeats

Reference for this section


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. I
by Phillip Compeau & Pavel Pevzner

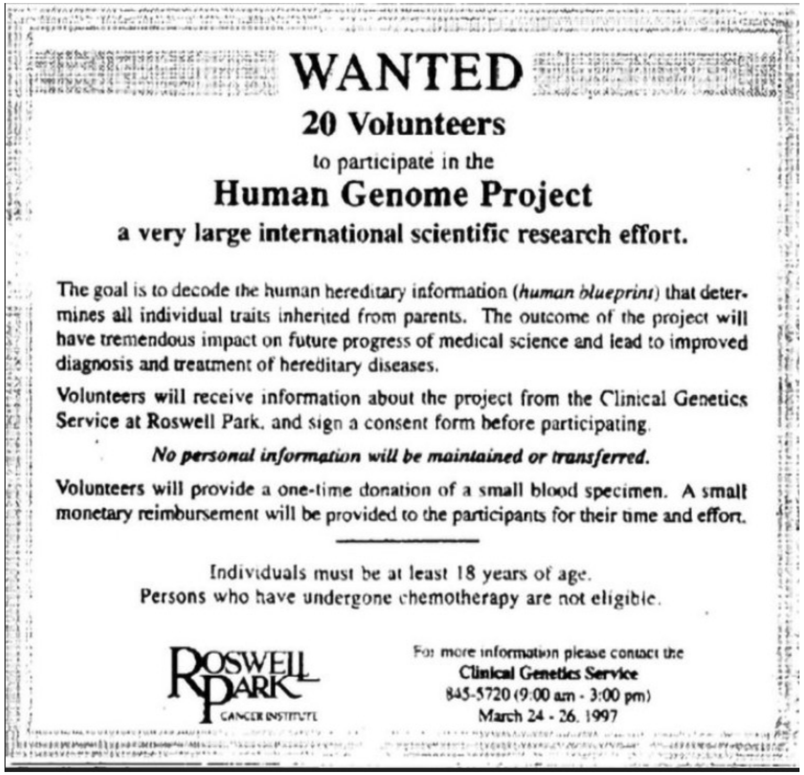

BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. II
by Phillip Compeau & Pavel Pevzner

- Chapter 3 Vol 1 (pag 115-170)

Section 6

Genome assemble

- Suffix tree
- Algorithm: Burrow-Wheeler Transform



WANTED

20 Volunteers

to participate in the

Human Genome Project

a very large international scientific research effort.

The goal is to decode the human hereditary information (*human blueprint*) that determines all individual traits inherited from parents. The outcome of the project will have tremendous impact on future progress of medical science and lead to improved diagnosis and treatment of hereditary diseases.

Volunteers will receive information about the project from the Clinical Genetics Service at Roswell Park, and sign a consent form before participating.

No personal information will be maintained or transferred.

Volunteers will provide a one-time donation of a small blood specimen. A small monetary reimbursement will be provided to the participants for their time and effort.

Individuals must be at least 18 years of age.
Persons who have undergone chemotherapy are not eligible.

ROSWELL PARK
CANCER INSTITUTE

For more information please contact the
Clinical Genetics Service
845-5720 (9:00 am - 3:00 pm)
March 24 - 26, 1997

Biologists need algorithms for genome assemble

- **Reference genome**: database genome used for comparison (GRCh38).
- https://www.ncbi.nlm.nih.gov/genome/guide/human/

- **Question:** How can we assemble individual genomes efficiently using the reference genome?

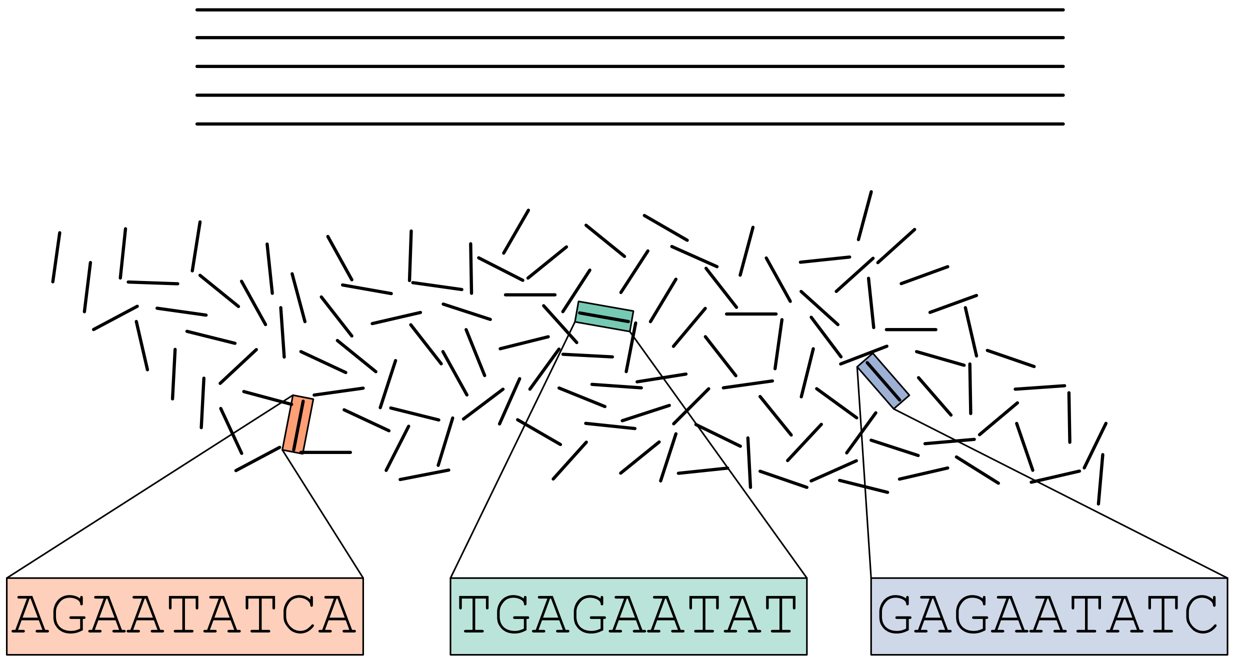
CTGATGATGGACTACGCTACTACTGCTAGCTGTAT	Individual
CTGAGGATGGACTACGCTACTACTGATAGCTGTTT	Reference

Why Not Use Assembly?

Multiple copies of a genome

Shatter the genome into reads

Sequence the reads



AGAATATCA

TGAGAATAT

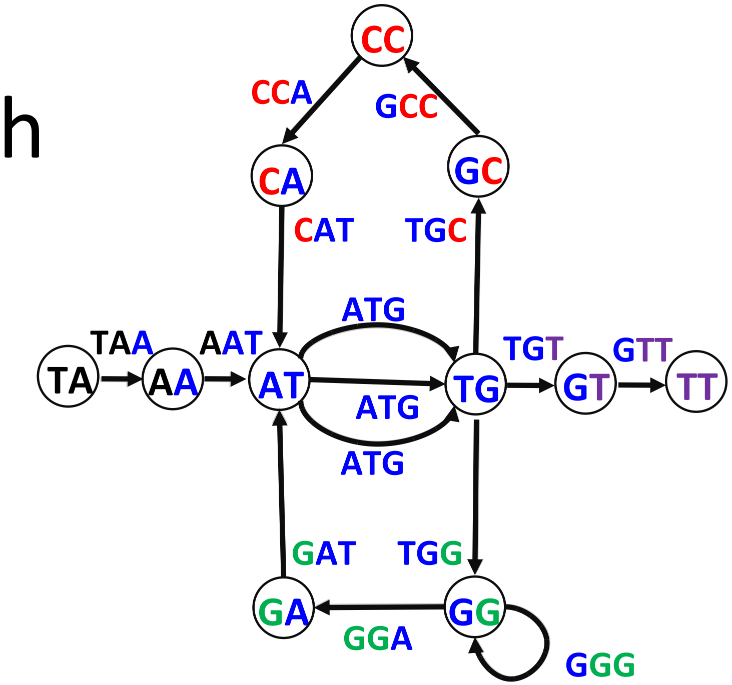
GAGAATATC

Assemble the genome with overlapping reads

```
    AGAATATCA
   GAGAATATC
  TGAGAATAT
...TGAGAATATCA...
```

Why Not Use Assembly?

- Constructing a de Bruijn graph takes a lot of memory.



- Hope: a machine in a clinic that would collect and map reads in 10 minutes.

- Idea: use existing structure of reference genome to help us sequence a patient’s genome.

Read Mapping

- **Read mapping**: determine where each read has high similarity to the reference genome.

```
CTGAGGATGGACTACGCTACTACTGATAGCTGTTT   Reference
  GAGGA    CCACG       TGA-A          Reads
```

Why Not Use Alignment?

- **Fitting alignment:** align each read *Pattern* to the best substring of *Genome*.

- Has runtime O(|*Pattern*| * |*Genome*|) for each *Pattern*.

- Has runtime O(|*Patterns*| * |*Genome*|) for a collection of *Patterns*.

Exact Pattern Matching

- Focus on a simple question: where do the reads match the reference genome *exactly*?

- **Single Pattern Matching Problem:**
  - **Input:** A string *Pattern* and a string *Genome.*
  - **Output:** All positions in *Genome* where *Pattern* appears as a substring.

Exact Pattern Matching

- Focus on a simple question: where do the reads match the reference genome *exactly*?

- **Multiple Pattern Matching Problem:**
  - **Input:** A collection of strings *Patterns* and a string *Genome.*
  - **Output:** All positions in *Genome* where a string from *Patterns* appears as a substring.

A Brute Force Approach

- We can simply iterate a brute force approach method, sliding each *Pattern* down *Genome.*

```
panamabananas    Genome
       nana      Pattern
```

- **Note**: we use words instead of DNA strings for convenience.

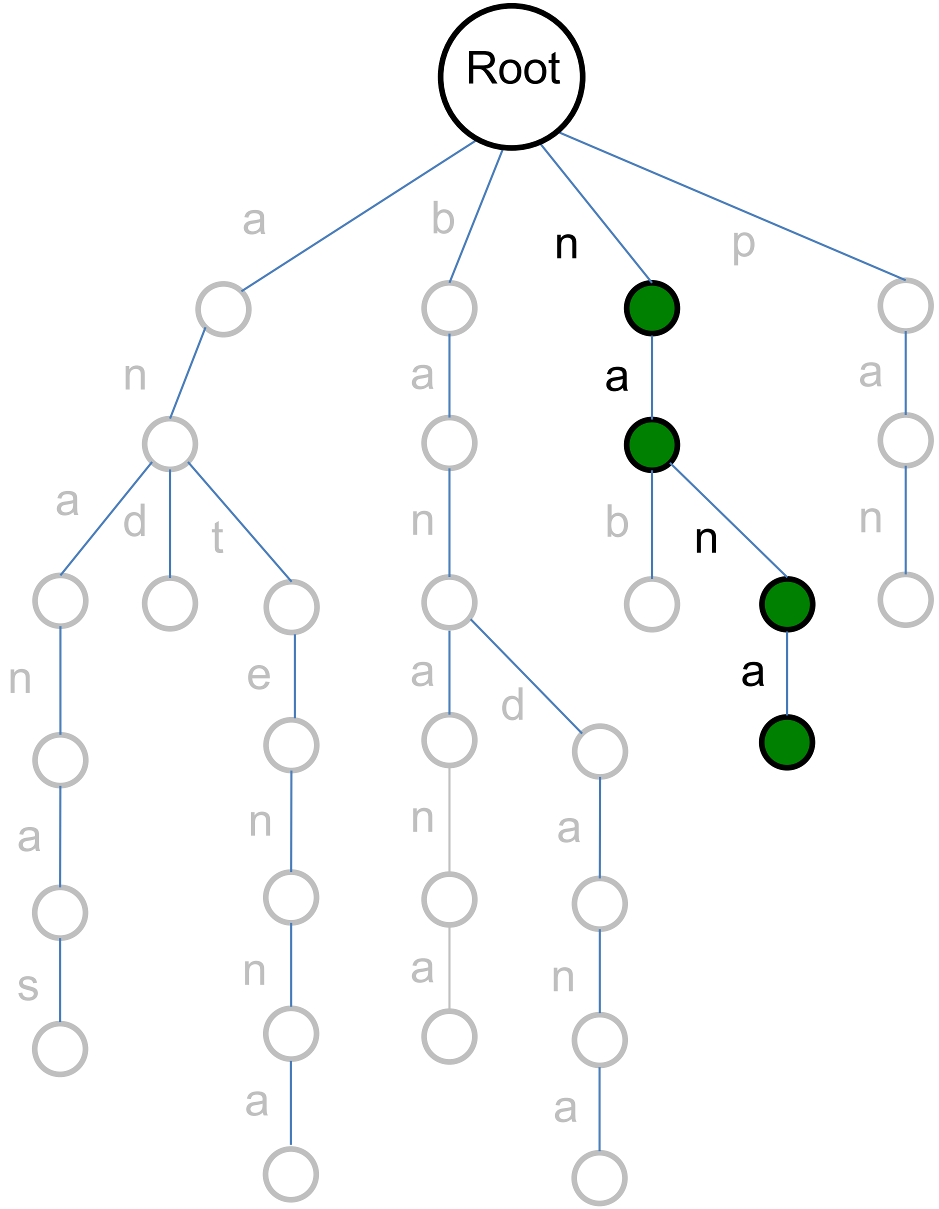
Brute Force Is Too Slow

- The runtime of the brute force approach is too high!
  - Single *Pattern*: O(|*Genome*| * |*Pattern*|)
  - Multiple *Patterns*: O(|*Genome*| * |*Patterns*|)
  - |*Patterns*| = combined length of *Patterns*

Processing Patterns into a Trie

- Idea: combine reads into a graph. Each substring of the genome can match at most one read. So each read will correspond to a unique path through this graph.

- The resulting graph is called a **trie**.



Patterns

banana

pan

and

nab

antenna

bandana

ananas

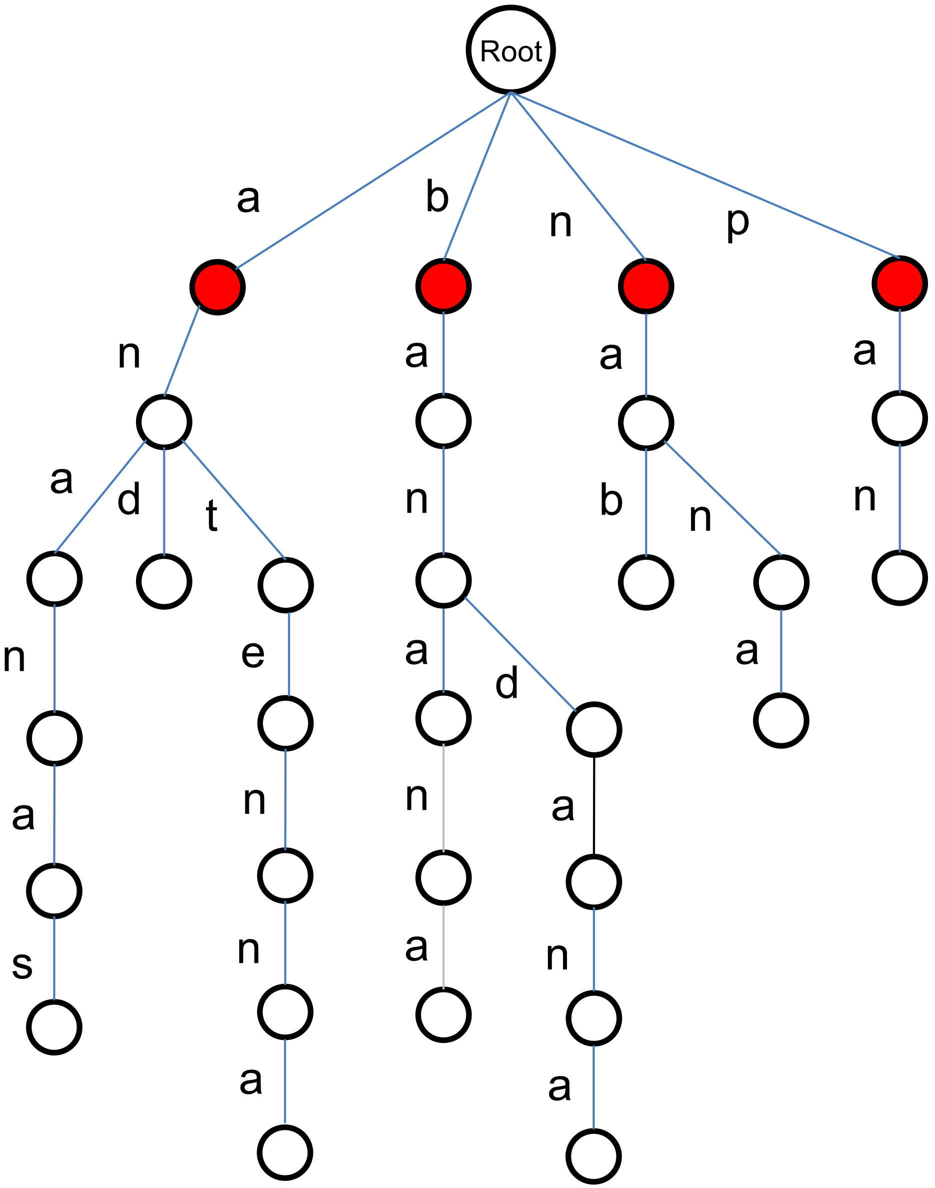
nana

Using the Trie for Pattern Matching

- **TrieMatching**: Slide the trie down the genome.

- At each position, walk down the trie and see if we can reach a leaf by matching symbols.

panamabanana**s**



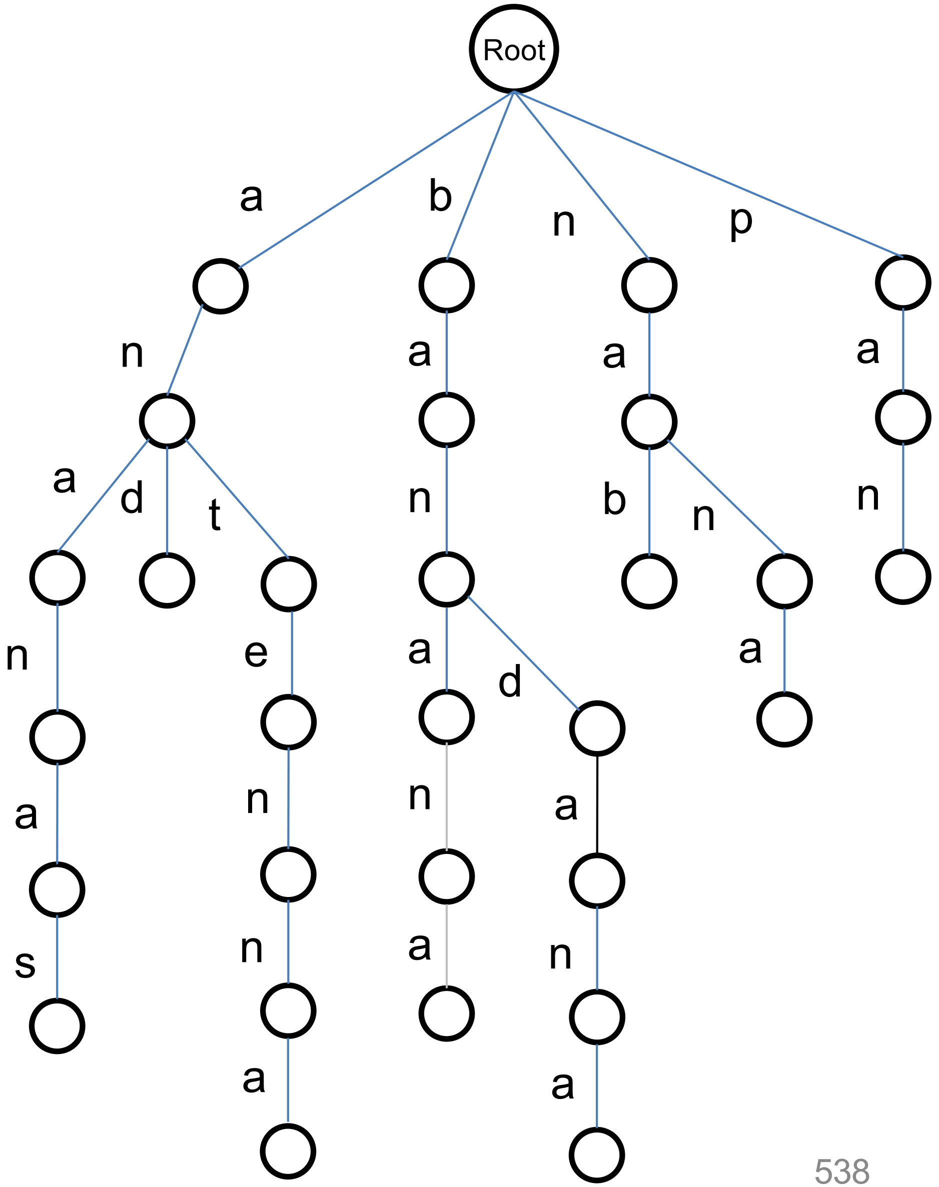
- Runtime of Brute Force:
  - Total: O(|*Genome*|*|*Patterns*|)

- Runtime of Trie Matching:
  - Trie Construction: O(|*Patterns*|)
  - Pattern Matching: O(|*Genome*| * |*LongestPattern*|)

Memory Analysis of Trie Matching

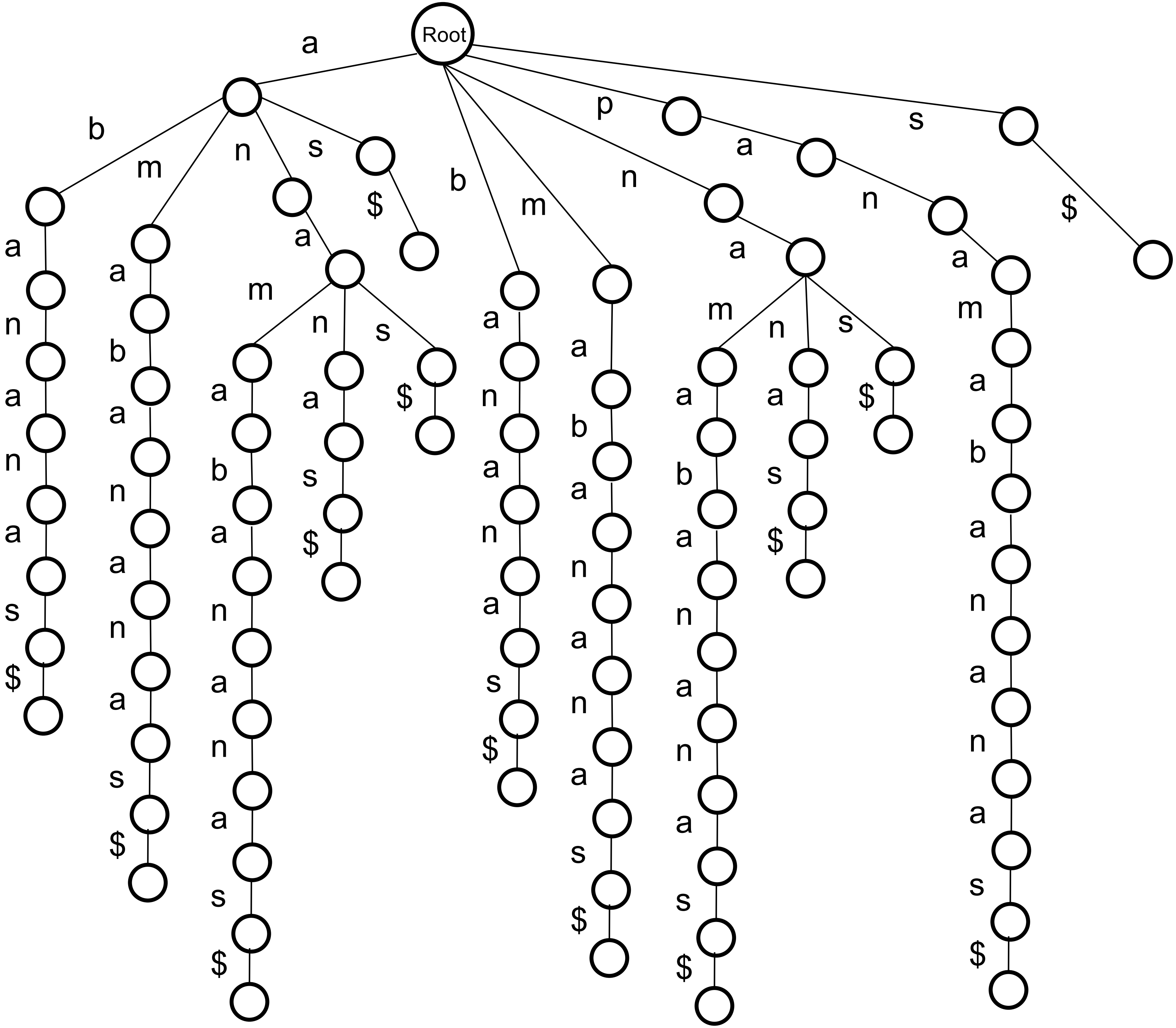
- Our trie: 30 edges, |*Patterns*| = 39

- Worst case: # edges = **O(|*Patterns*|)**



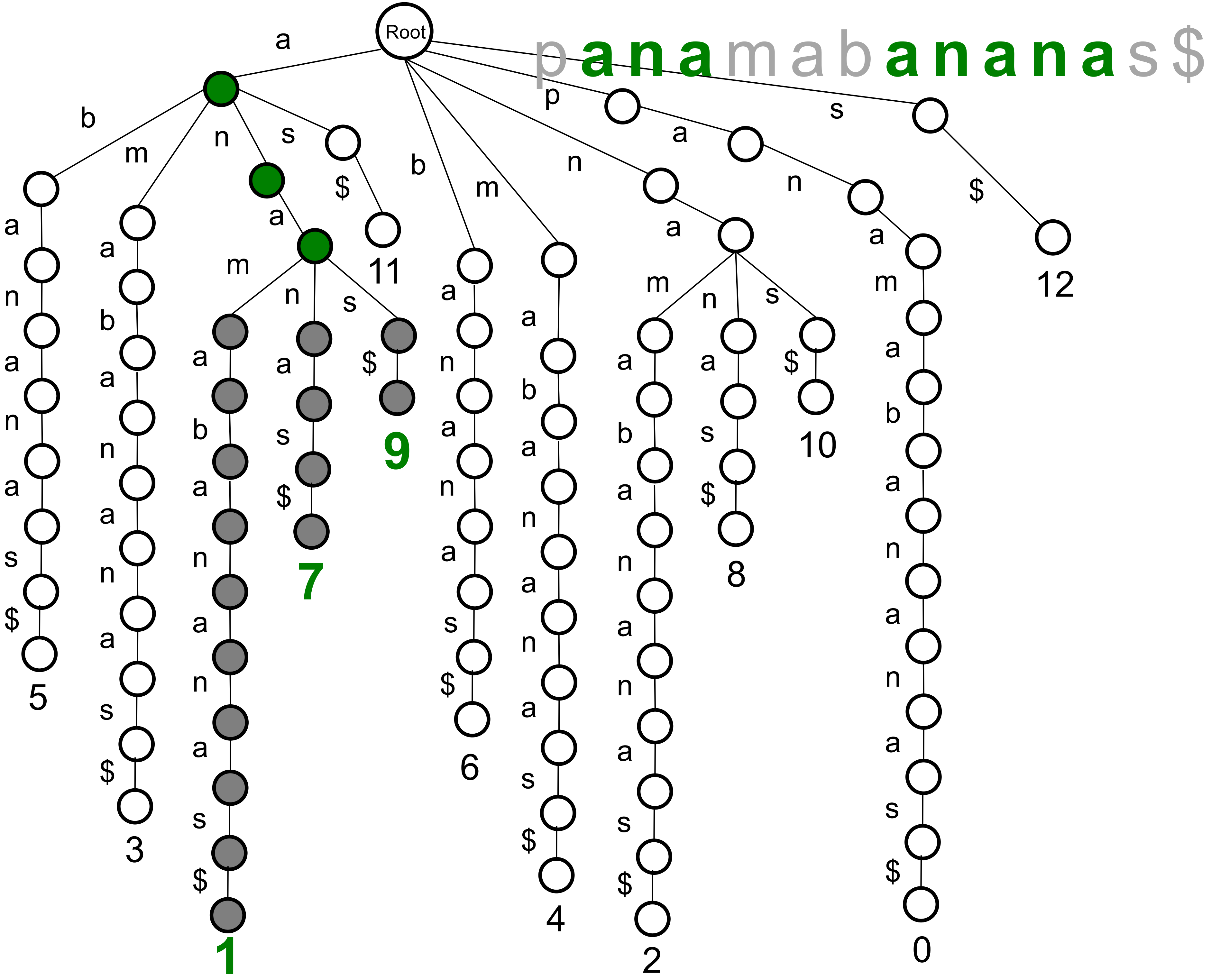
Preprocessing the Genome

- What if instead we create a data structure from the genome itself?
- Split *Genome* into all its suffixes. (Show matching “banana” by finding the suffix “bananas”.)
- How can we combine these suffixes into a data structure?
- Let’s use a trie!


Root
a
b
a
n
a
n
a
s
$
m
a
b
a
n
a
n
a
s
$
n
a
m
a
b
a
n
a
n
a
s
$
n
a
s
$
s
$
s
$
b
a
n
a
n
a
s
$
m
a
b
a
n
a
n
a
s
$
n
a
m
a
b
a
n
a
n
a
s
$
n
a
s
$
s
$
p
a
n
a
m
a
b
a
n
a
n
a
s
$
s
$

The Suffix Trie and Pattern Matching

- For each *Pattern*, see if *Pattern* can be spelled out from the root downward in the suffix trie.


panamabananas$
Root
5
3
1
7
9
11
6
4
2
8
10
0
12

Memory Trouble Once Again

- Worst case: the suffix trie holds O(|*Suffixes*|) nodes.

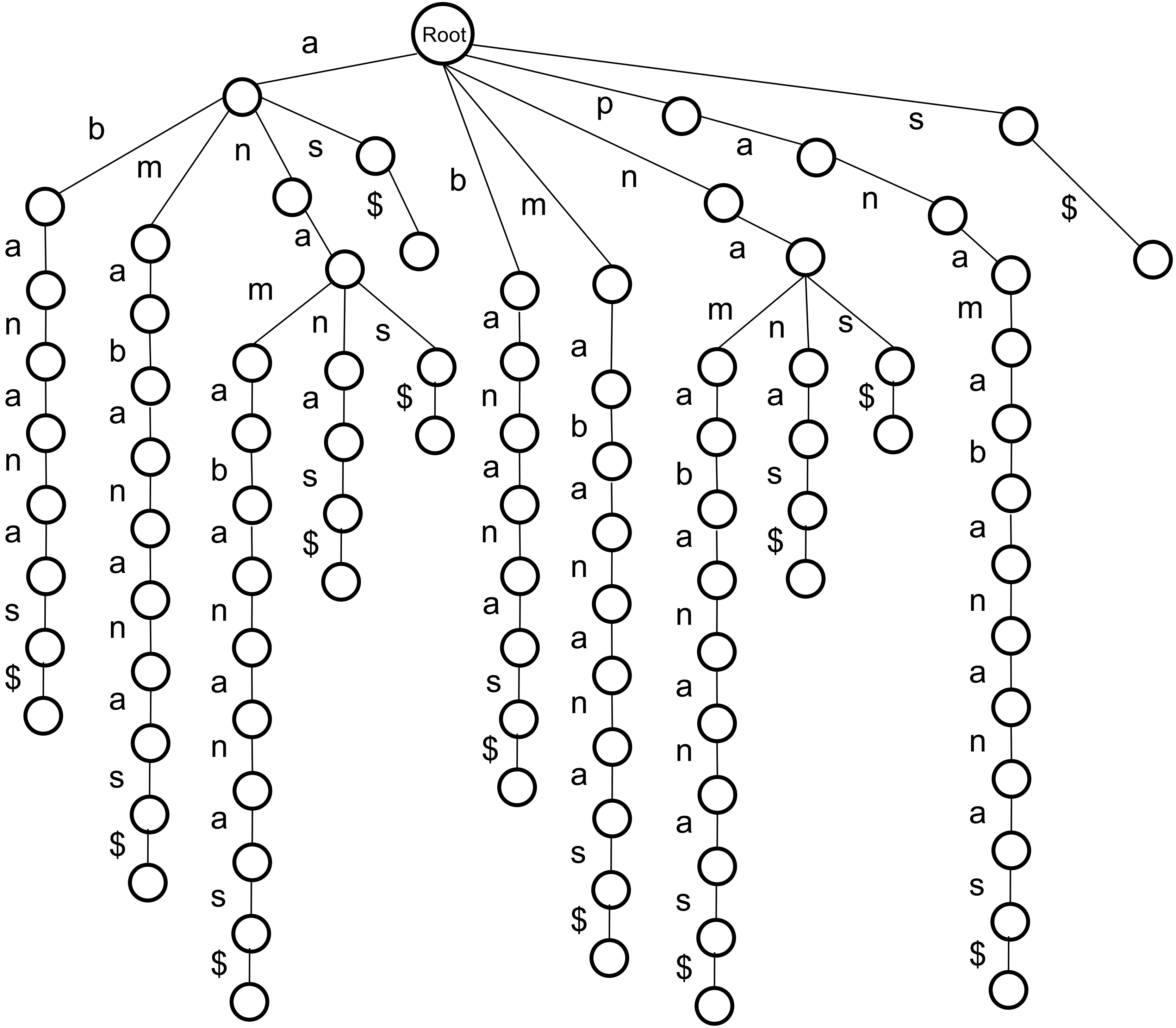
- For a *Genome* of length *n*, |*Suffixes*| = $n(n-1)/2 = O(n^2)$

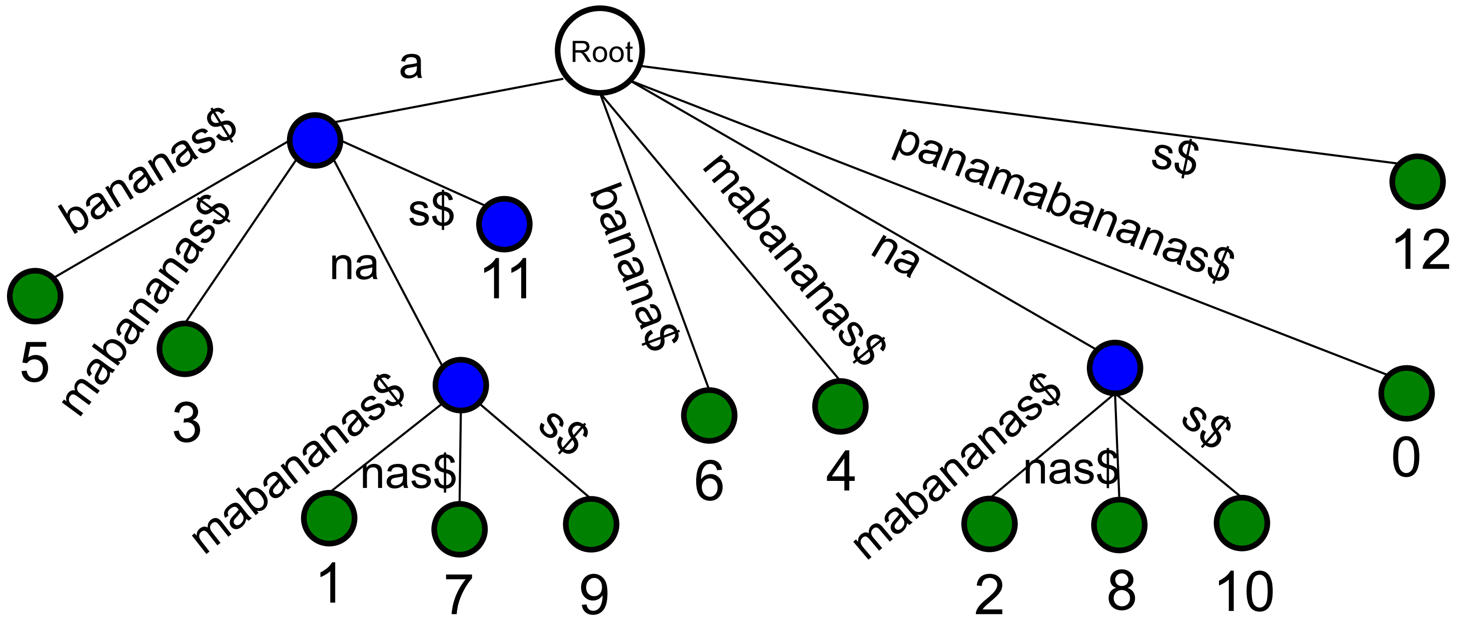
Suffixes

```
panamabananas$
 anamabananas$
  namabananas$
   amabananas$
    mabananas$
     abananas$
      bananas$
       ananas$
        nanas$
         anas$
          nas$
           as$
            s$
             $
```

Compressing the Trie

- This doesn’t mean that our idea was bad!
- To reduce memory, we can compress each “nonbranching path” of the tree into an edge.


Root
a
b
a n a n a s $
m
a b a n a n a s $
n
a
m
a b a n a n a s $
n
a s $
s
$
s
$
b
a n a n a s $
m
a b a n a n a s $
n
a
m
a b a n a n a s $
n
a s $
s
$
p
a
n
a m a b a n a n a s $
s
$



- This data structure is called a **suffix tree**.

- For any *Genome*, # nodes < 2|*Genome*|.
  - **# leaves** = |*Genome*|;
  - **# internal nodes** < |*Genome*| – 1

Complexity

- Runtime:
  - O(|*Genome*|^2) to construct the suffix tree.
  - O(|*Genome*| + |*Patterns*|) to find pattern matches.

- Memory:
  - O(|*Genome*|^2) to construct the suffix tree.
  - O(|*Genome*|) to store the suffix tree.

Complexity

- Runtime:
  - O(|*Genome*|) to construct the suffix tree ***directly.***
  - O(|*Genome*| + |*Patterns*|) to find pattern matches.
  - **Total: O(|*Genome*| + |*Patterns*|)**

- Memory:
  - O(|*Genome*|) to construct the suffix tree ***directly.***
  - O(|*Genome*|) to store the suffix tree.
  - **Total: O(|*Genome*| + |*Patterns*|)**

We are Not Finished Yet

- I am happy with the suffix tree, but I am not completely satisfied.
  - Runtime: O(|*Genome*| + |*Patterns*|)
  - Memory: O(|*Genome*|)

- However, big-O notation ignores constants!
  - The best known suffix tree implementations require ~ 20 times the length of |*Genome*|.
  - Can we reduce this constant factor?

Genome Compression

- Idea: decrease the amount of memory required to hold *Genome*.

- This indicates that we need methods of **compressing** a large genome, which is seemingly a separate problem.

Idea #1: Run-Length Encoding

- **Run-length encoding:** compresses a run of *n* identical symbols.

Genome

GGGGGGGGGGCCCCCCCCCCCAAAAAAATTTTTTTTTTTTTTTCCCCCG

↓

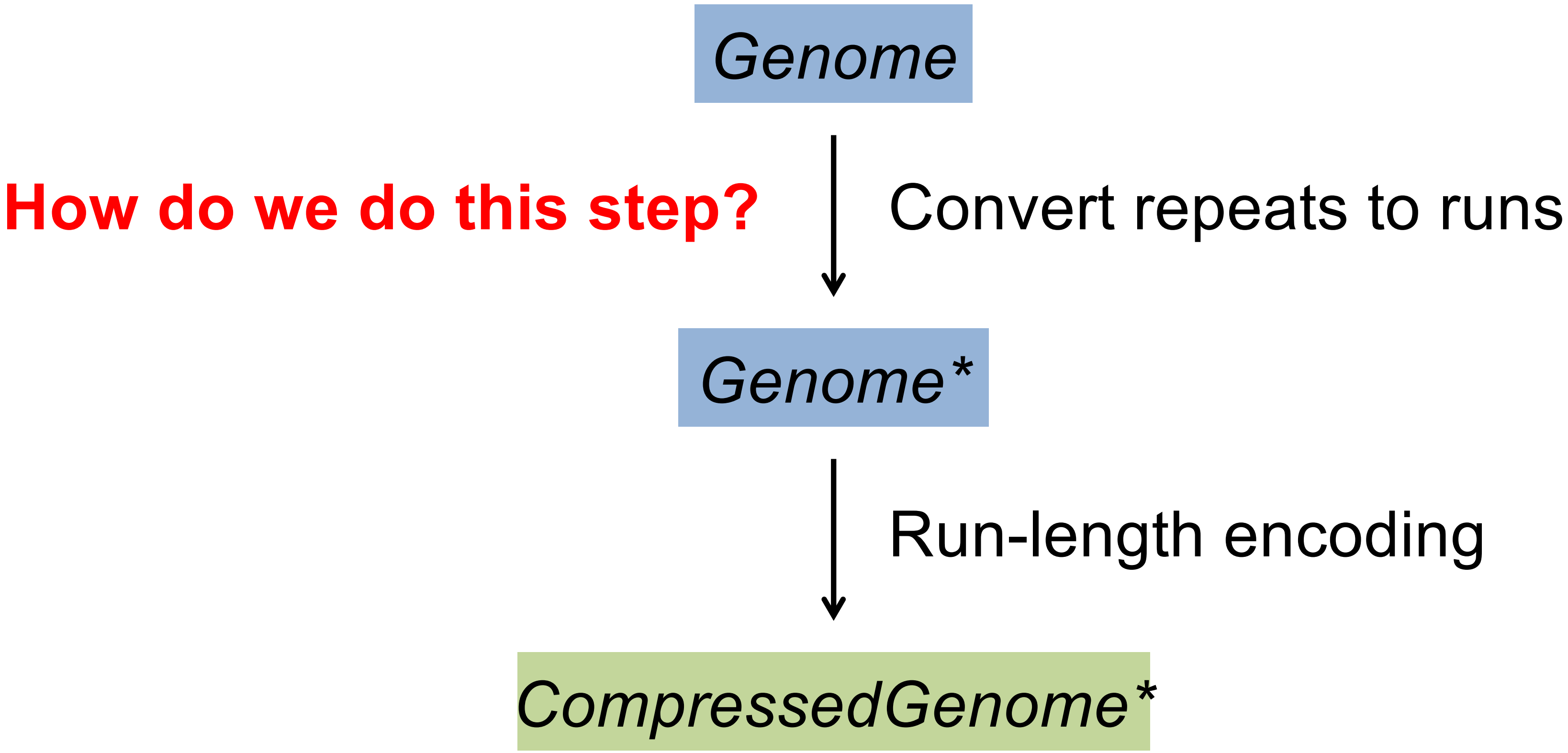
10G11C7A15T5C1G

Run-length encoding

- Problem: Genomes don’t have lots of runs…

Converting Repeats to Runs

- …but they do have lots of repeats!



Genome

How do we do this step?

Convert repeats to runs

*Genome**

Run-length encoding

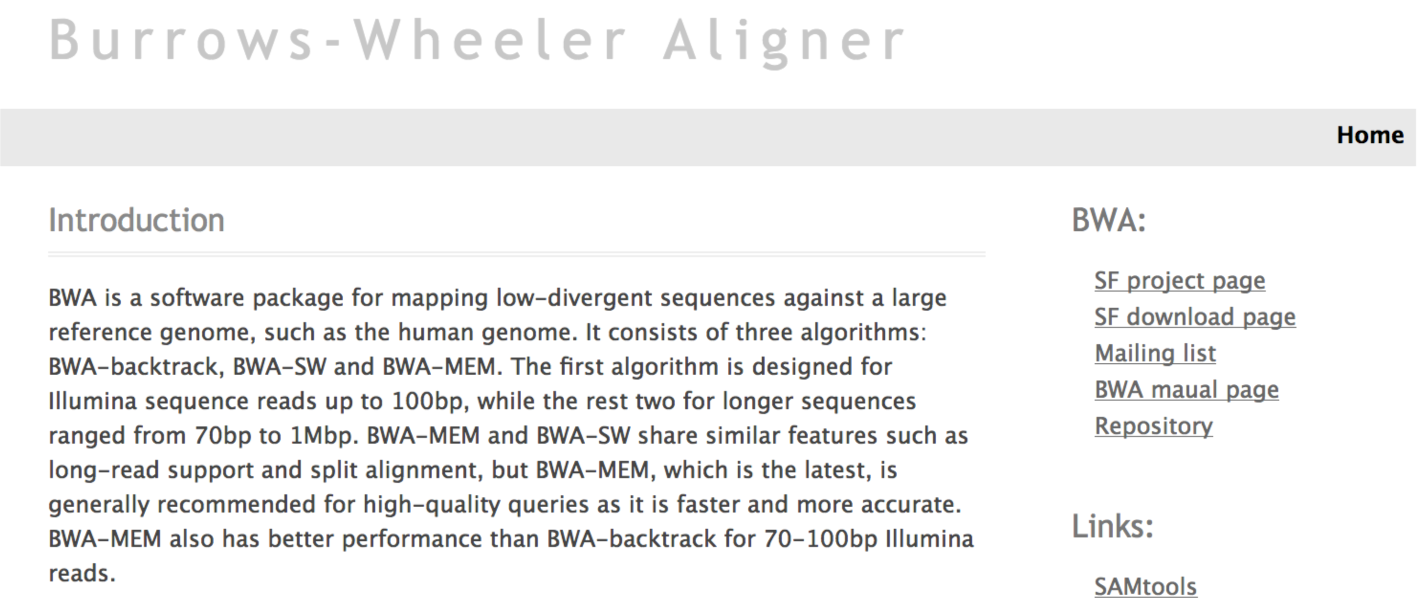
*CompressedGenome**

The Burrows – Wheeler Transform

Michael Burrows (left), David Wheeler (right)
both at the Computer Laboratory

Bowtie

An ultrafast memory-efficient short read aligner



Burrows-Wheeler Aligner

Introduction

BWA is a software package for mapping low-divergent sequences against a large reference genome, such as the human genome. It consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. The first algorithm is designed for Illumina sequence reads up to 100bp, while the rest two for longer sequences ranged from 70bp to 1Mbp. BWA-MEM and BWA-SW share similar features such as long-read support and split alignment, but BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate. BWA-MEM also has better performance than BWA-backtrack for 70-100bp Illumina reads.

BWA:

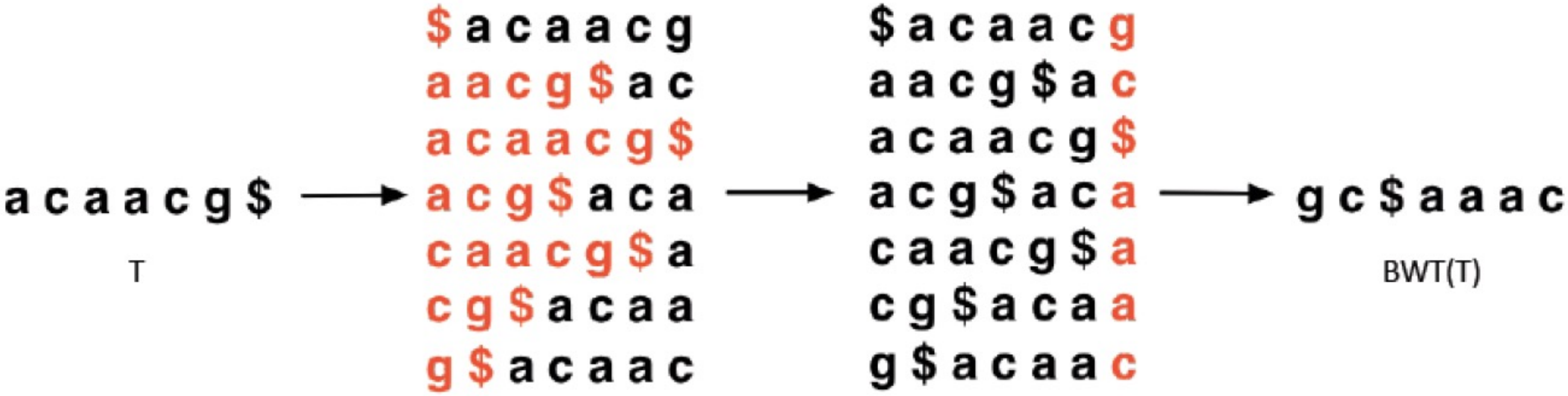
- SF project page
- SF download page
- Mailing list
- BWA maual page
- Repository

Links:

- SAMtools

The Burrows Wheeler Transform

Three steps: 1) Given a string T in input, we 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 the greatest character in the alphabet and occurs exactly once in the text); 2) We sort the matrix according to the alphabetic order. Note that the cycle and the sort procedures of the Burrows-Wheeler induce 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).



acaacg$

T

$acaacg
aacg$ac
acaacg$
acg$aca
caacg$a
cg$acaa
g$acaac

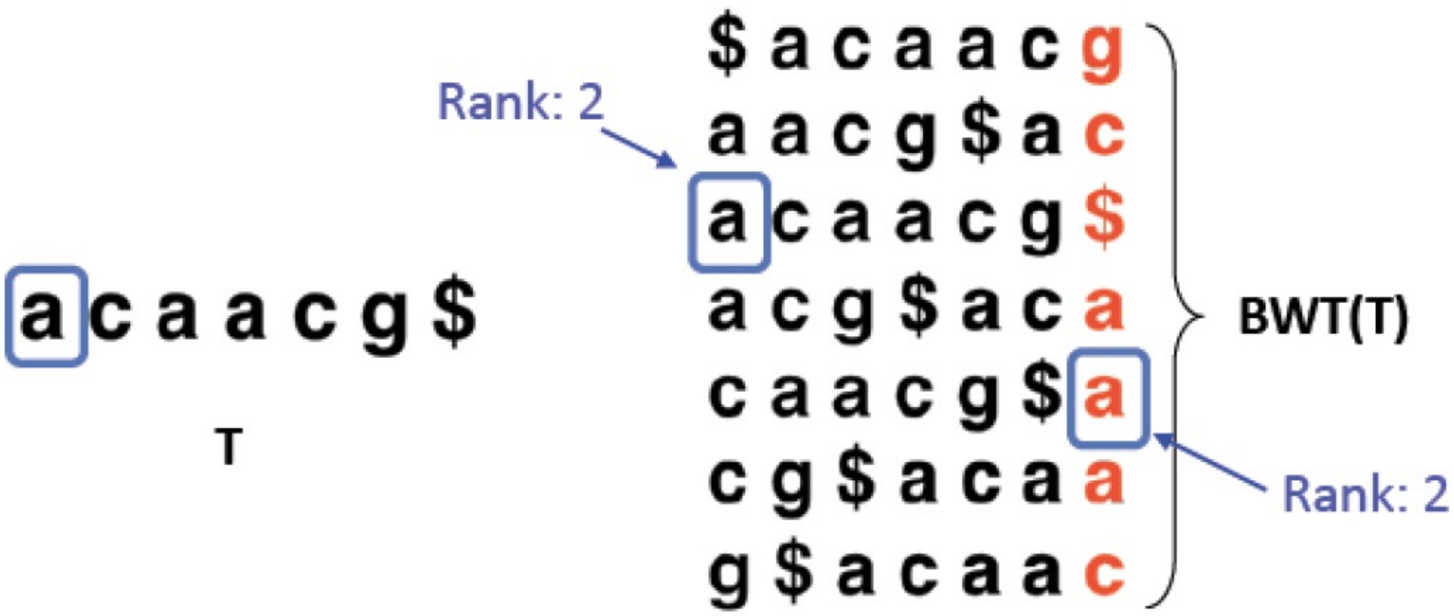
$acaacg
aacg$ac
acaacg$
acg$aca
caacg$a
cg$acaa
g$acaac

gc$aaac

BWT(T)

BWT

Property that makes BWT(T) reversible is LF Mapping: the i-th occurrence of a character in Last column is same text occurrence as the i-th occurrence in the First column (i.e. the sorting strategy preserves the relative order in both last column and first column).



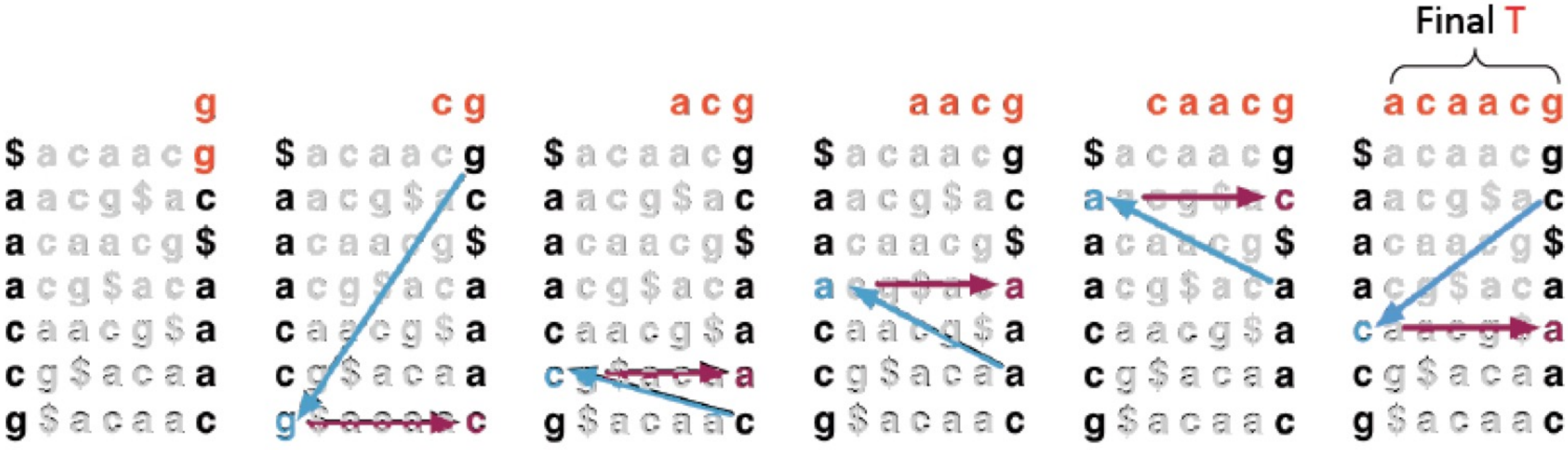
a c a a c g $

T

$ a c a a c g
a a c g $ a c
a c a a c g $
a c g $ a c a
c a a c g $ a
c g $ a c a a
g $ a c a a c

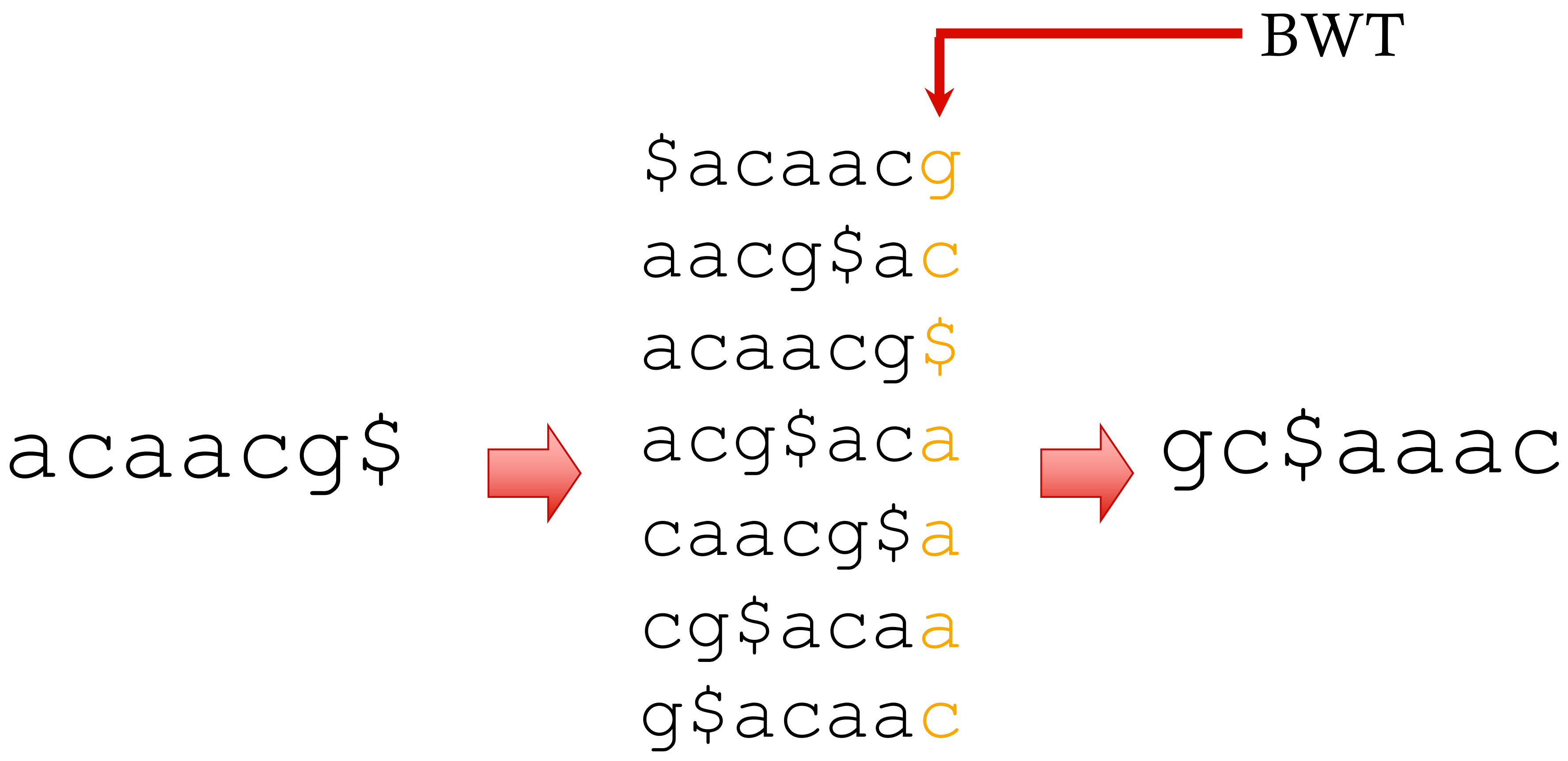
BWT

To recreate T from BWT(T), repeatedly apply the 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; $T = cg\$$.



```
      g          c g         a c g       a a c g     c a a c g   a c a a c g
$acaacg    $acaacg    $acaacg    $acaacg    $acaacg    $acaacg
aacg$ac    aacg$ac    aacg$ac    aacg$ac    aacg$ac    aacg$ac
acaacg$    acaacg$    acaacg$    acaacg$    acaacg$    acaacg$
acg$aca    acg$aca    acg$aca    acg$aca    acg$aca    acg$aca
caacg$a    caacg$a    caacg$a    caacg$a    caacg$a    caacg$a
cg$acaa    cg$acaa    cg$acaa    cg$acaa    cg$acaa    cg$acaa
g$acaac    g$acaac    g$acaac    g$acaac    g$acaac    g$acaac
```

Burrows-Wheeler Transform (BWT)



BWT

acaacg$

```
$acaacg
aacg$ac
acaacg$
acg$aca
caacg$a
cg$acaa
g$acaac
```

gc$aaac

Burrows-Wheeler Matrix (BWM)

Burrows-Wheeler Matrix

```
$acaacg
aacg$ac
acaacg$
acg$aca
caacg$a
cg$acaa
g$acaac
```

Burrows-Wheeler Matrix

	`$acaacg`
3	`aacg$ac`
1	`acaacg$`
4	`acg$aca`
2	`caacg$a`
5	`cg$acaa`
6	`g$acaac`

See the suffix array?

Key observation

$a^1c^1a^2a^3c^2g^1\$^1$

“last first (LF) mapping”

The *i*-th occurrence of character X in the last column corresponds to the same text character as the *i*-th occurrence of X in the first column.

$^1\$acaacg^1$

$^2aacg\$ac^1$

$^1acaacg\$^1$

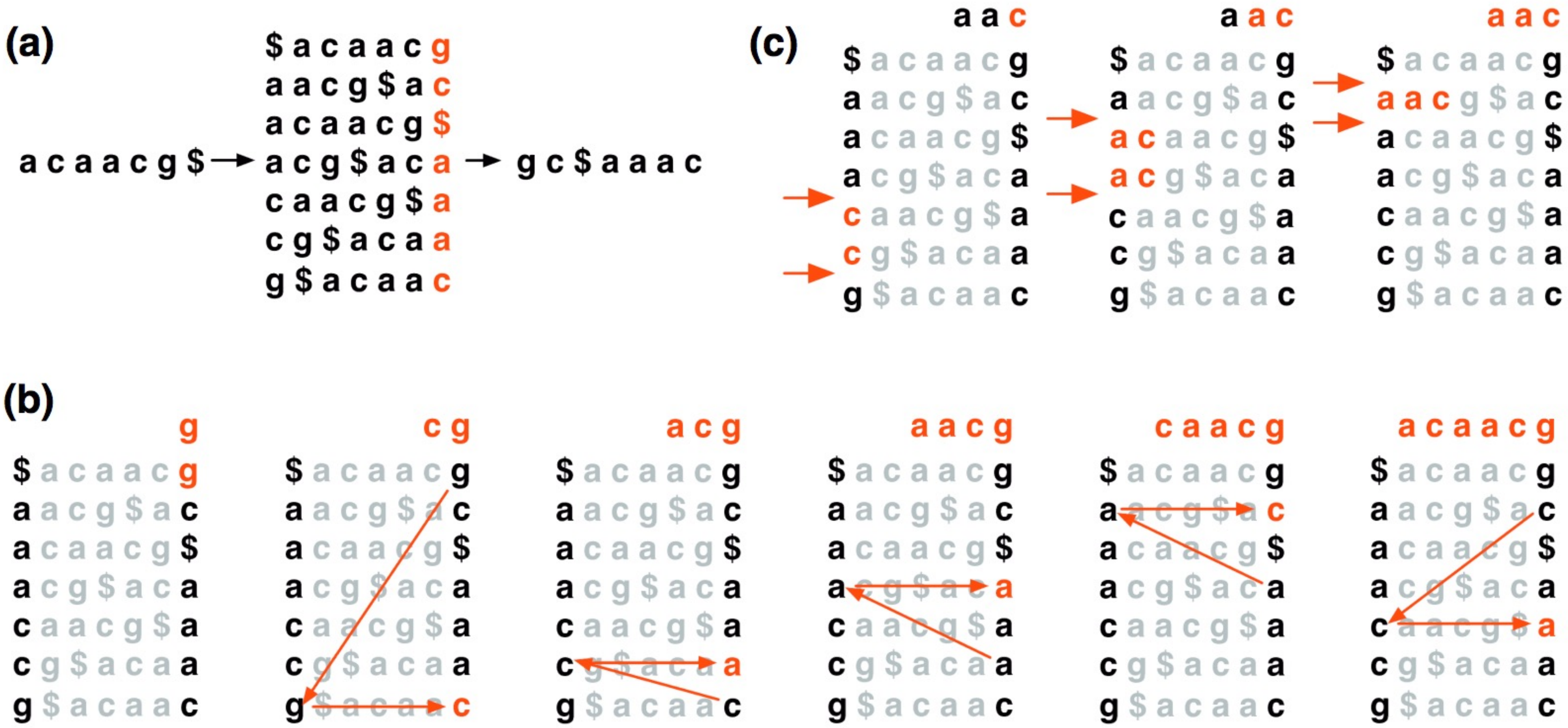
$^3acg\$aca^2$

$^1caacg\$a^1$

$^2cg\$acaa^3$

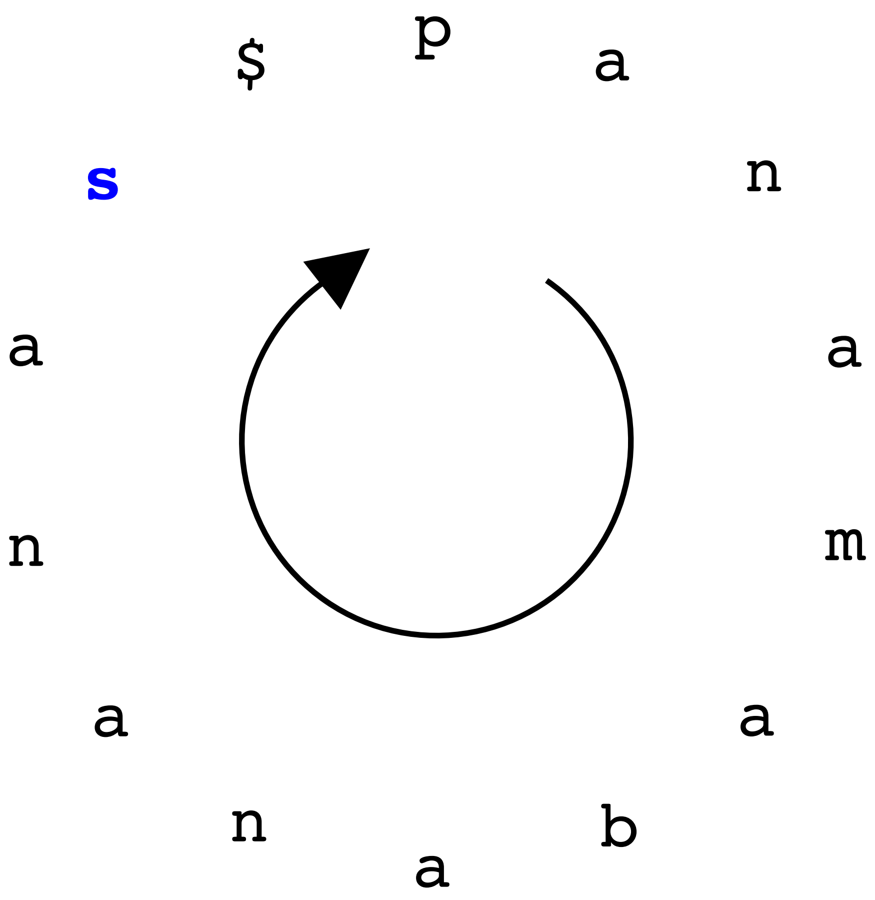
$^1g\$acaac^2$

Burrow Wheeler Transform



The Burrows-Wheeler Transform

```
panamabananas$
$panamabananas
s$panamabanana
```

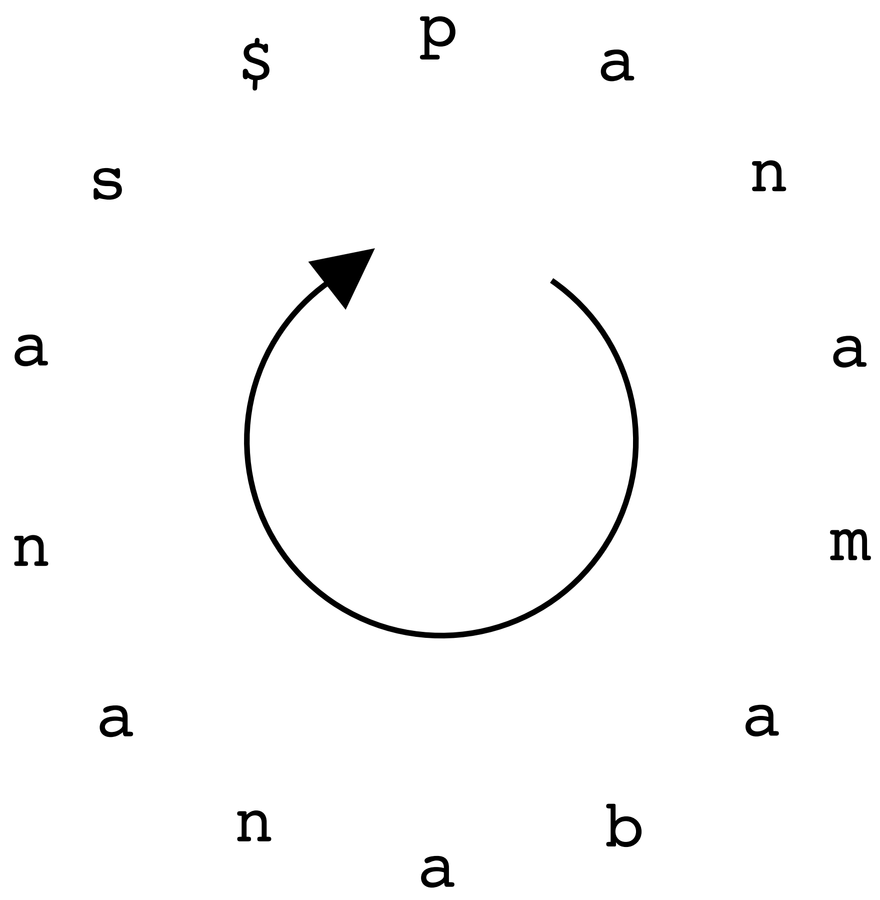


Form all cyclic rotations of “panamabananas$”

Burrows, Michael and Wheeler, David J. (1994), A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation
Li, H and Durbin, R (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics 25:1754-60.

The Burrows-Wheeler Transform

```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
amabananas$pan
namabananas$pa
anamabananas$p
```



Form all cyclic rotations of
"panamabananas$"

The Burrows-Wheeler Transform

```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
amabananas$pan
namabananas$pa
anamabananas$p
```

→

$panamabananas

abananas$panam

amabananas$pan

anamabananas$p

ananas$panamab

anas$panamaban

as$panamabanan

bananas$panama

mabananas$pana

namabananas$pa

nanas$panamaba

nas$panamabana

panamabananas$

s$panamabanana

Form all cyclic rotations of “panamabananas$”

Sort the strings lexicographically ($ comes first)

The Burrows-Wheeler Transform

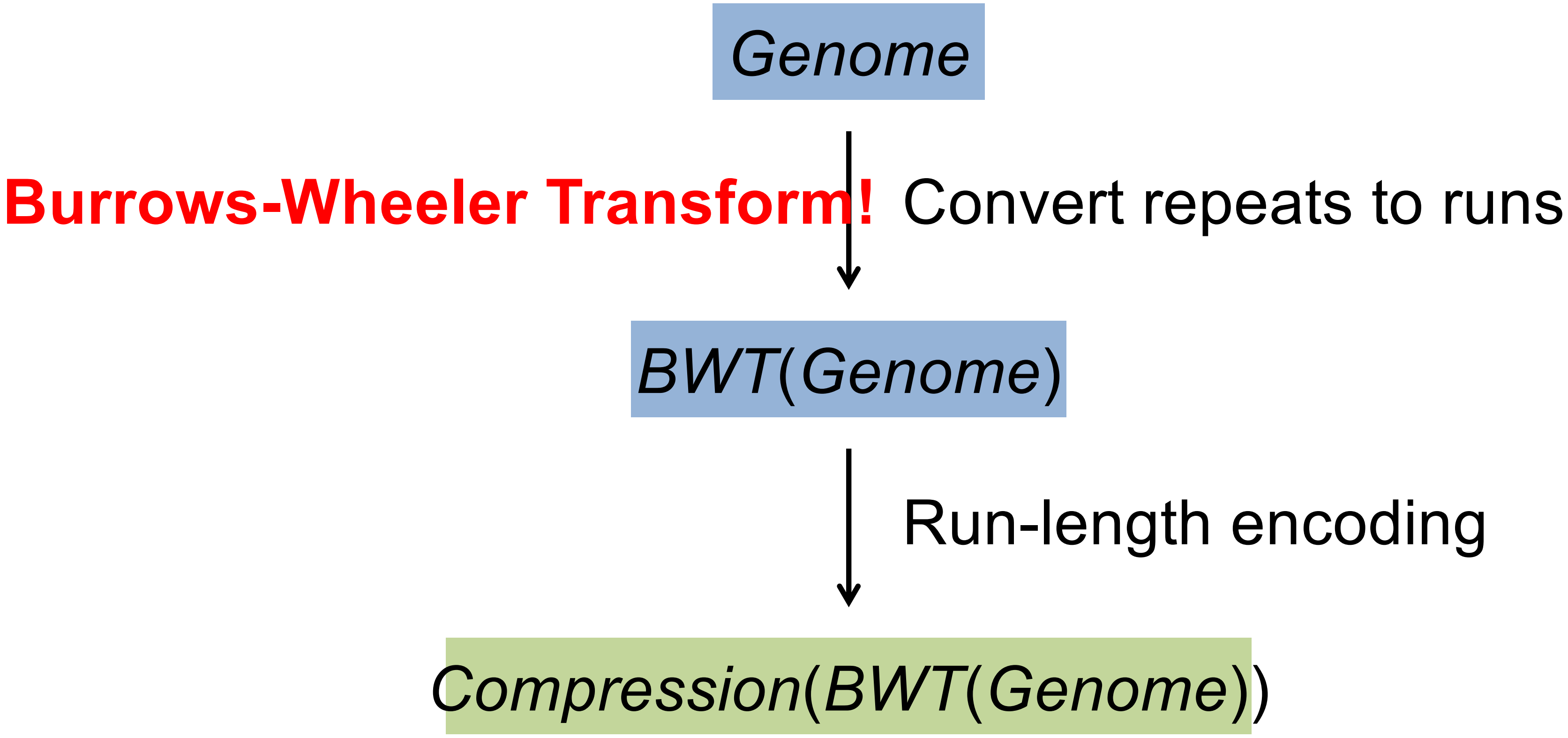
```
panamabananas$
$panamabananas
s$panamabanana
as$panamabanan
nas$panamabana
anas$panamaban
nanas$panamaba
ananas$panamab
bananas$panama
abananas$panam
mabananas$pana
amabananas$pan
namabananas$pa
anamabananas$p
```

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```

Form all cyclic rotations of
“panamabananas$”

Burrows-Wheeler Transform:
Last column =
smnpbnnaaaaa$a

BWT: Converting Repeats to Runs



Genome

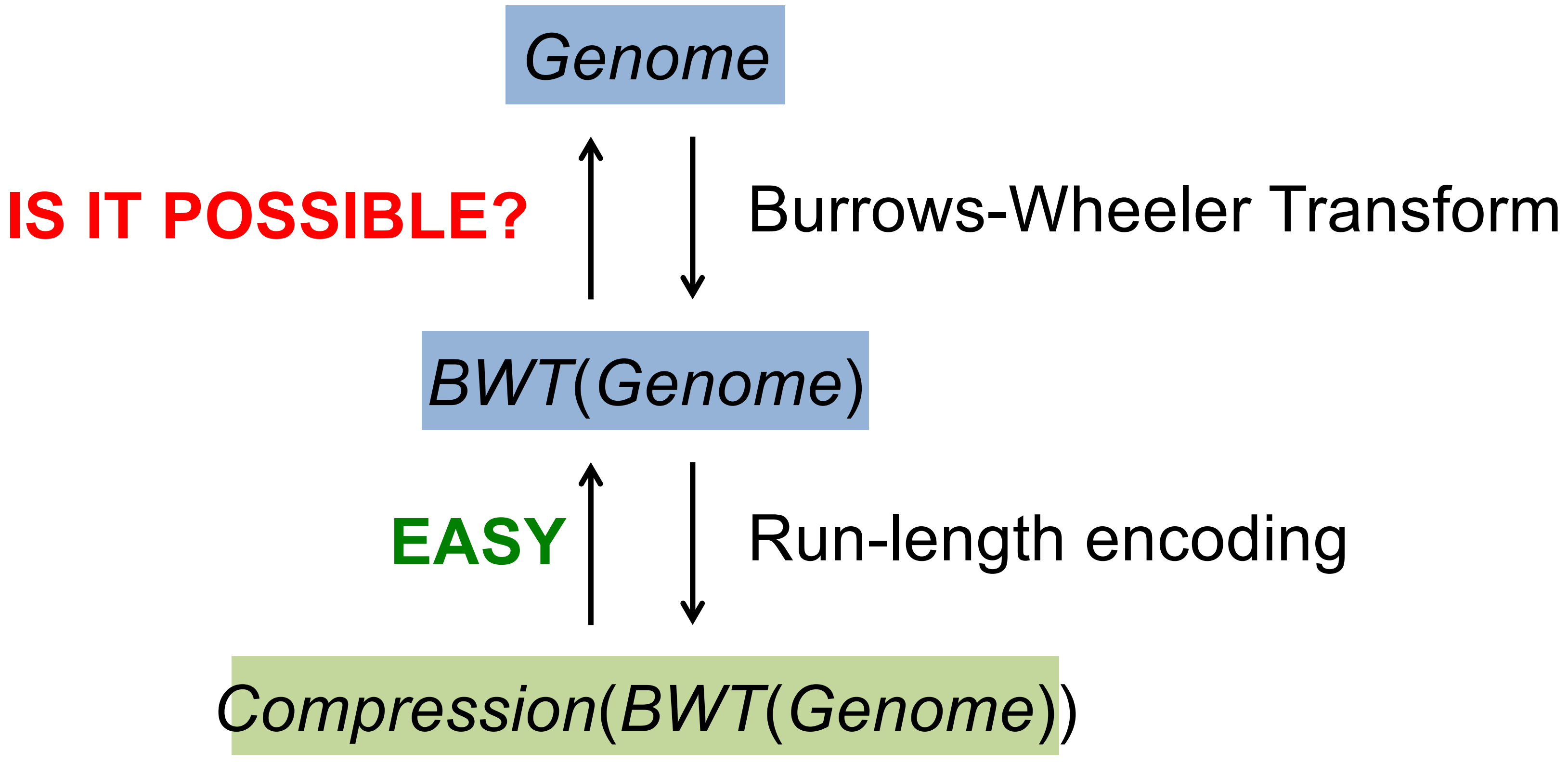
Burrows-Wheeler Transform! Convert repeats to runs

BWT(*Genome*)

Run-length encoding

Compression(*BWT*(*Genome*))

How Can We Decompress?



Genome

IS IT POSSIBLE?

Burrows-Wheeler Transform

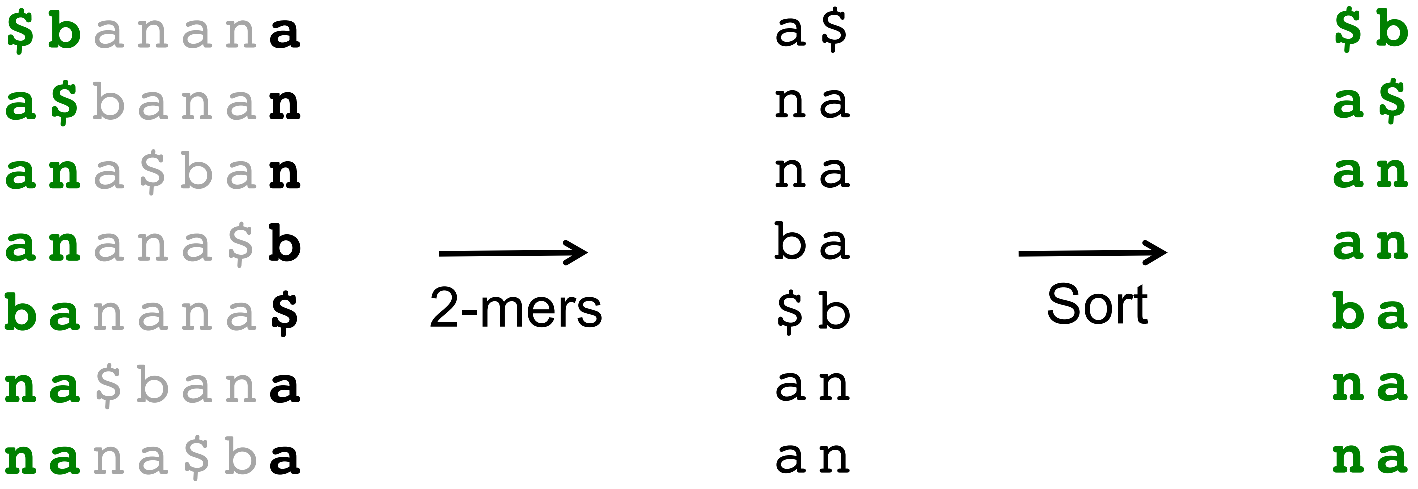
BWT(*Genome*)

EASY

Run-length encoding

Compression(*BWT*(*Genome*))

Reconstructing banana



```
$banana
a$banan
ana$ban
anana$b
banana$
na$bana
nana$ba
```

2-mers

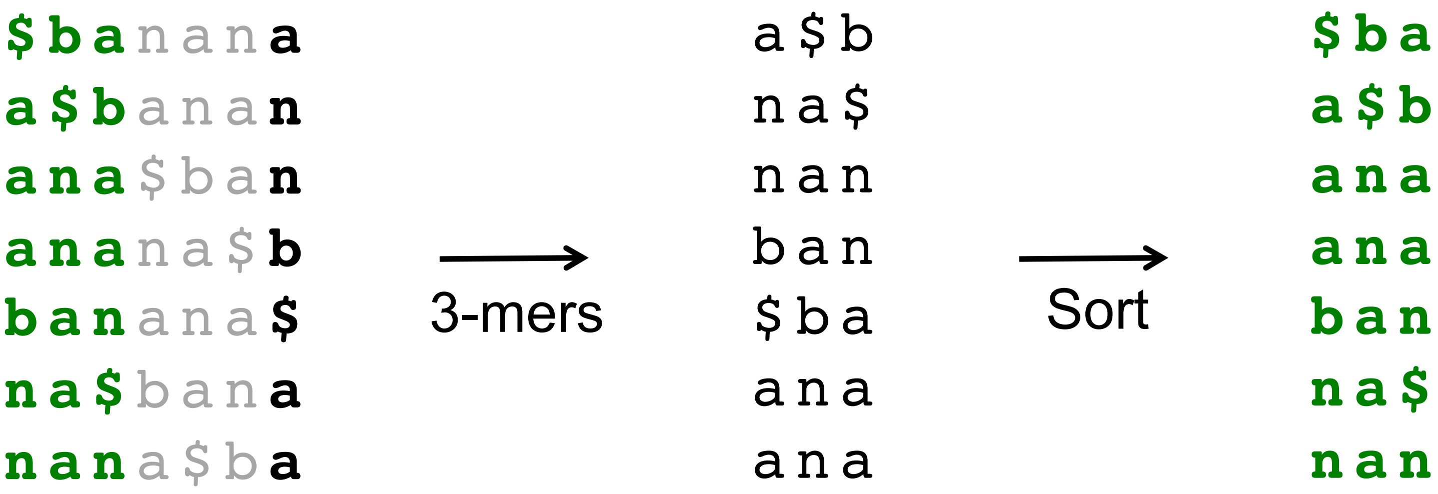
```
a$
na
na
ba
$b
an
an
```

Sort

```
$b
a$
an
an
ba
na
na
```

- We now know 2-mer composition of the circular string banana$
- Sorting gives us the first 2 columns of the matrix.

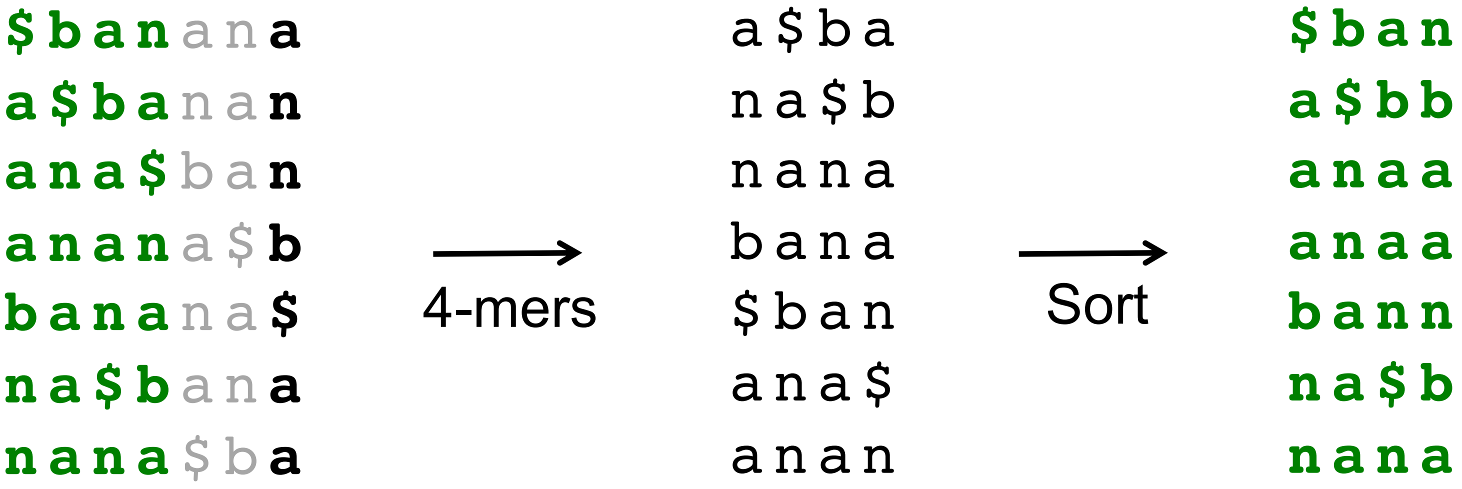
Reconstructing banana



$banana		a$b		$ba
a$banan		na$		a$b
ana$ban		nan		ana
anana$b	→	ban	→	ana
banana$	3-mers	$ba	Sort	ban
na$bana		ana		na$
nana$ba		ana		nan

- We now know 3-mer composition of the circular string banana$
- Sorting gives us the first 3 columns of the matrix.

Reconstructing banana



$banana
a$banan
ana$ban
anana$b
banana$
na$bana
nana$ba

4-mers

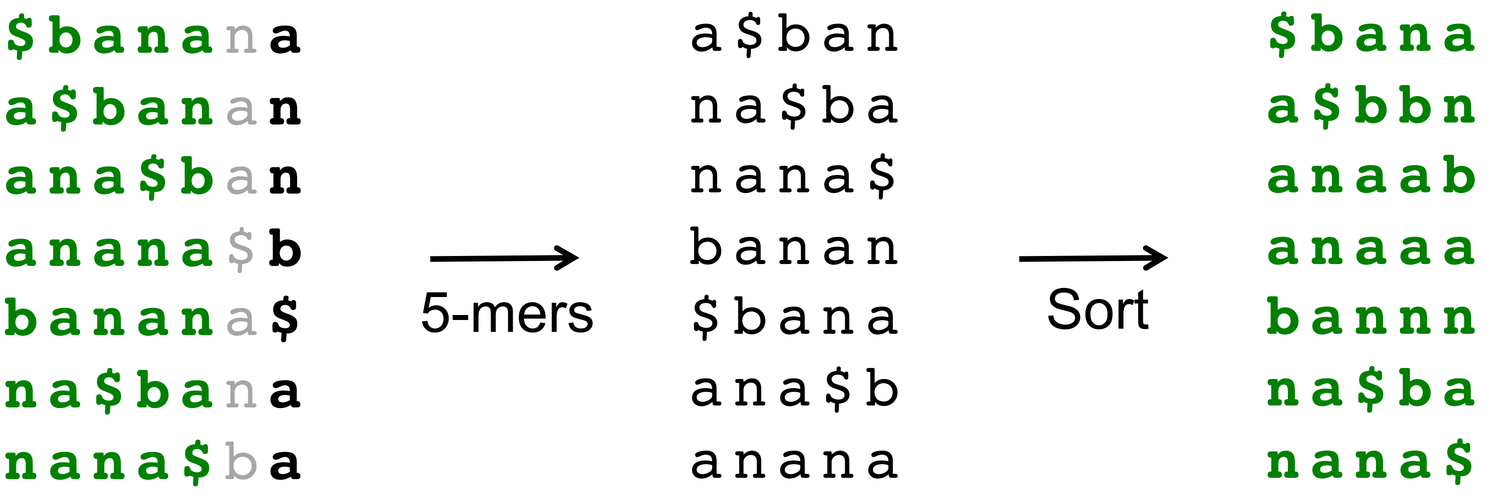
a$ba
na$b
nana
bana
$ban
ana$
anan

Sort

$ban
a$bb
anaa
anaa
bann
na$b
nana

- We now know 4-mer composition of the circular string banana$
- Sorting gives us the first 4 columns of the matrix.

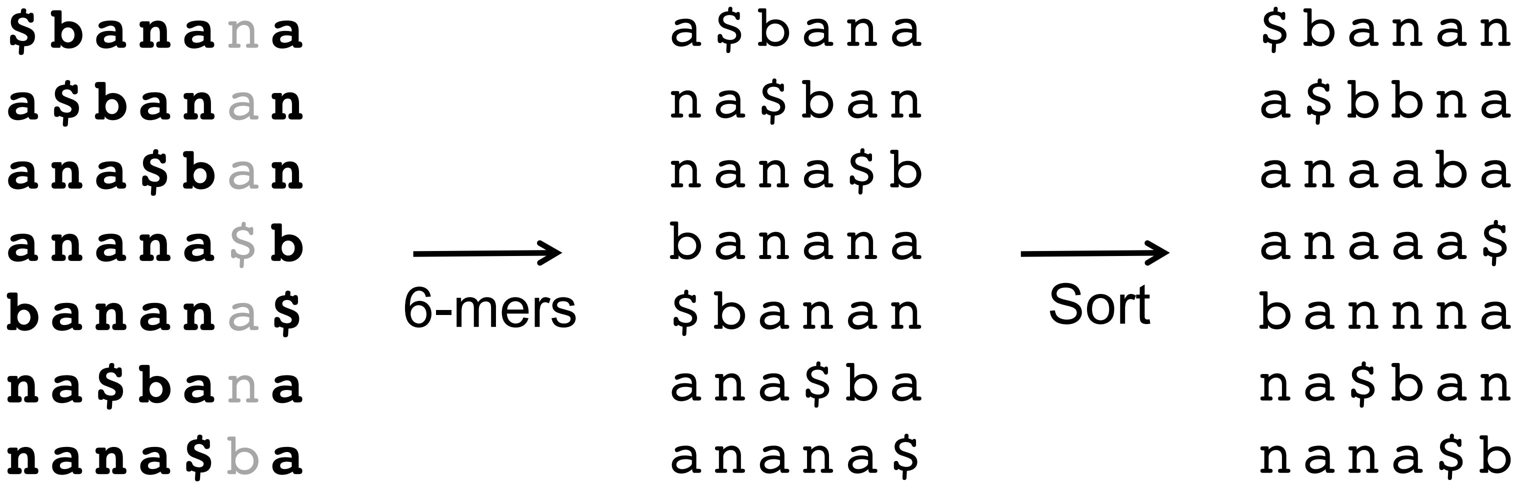
Reconstructing banana



```
$banana                  a$ban                   $bana
a$banan                  na$ba                   a$bbn
ana$ban                  nana$                   anaab
anana$b   ------->       banan      ------->     anaaa
banana$   5-mers         $bana      Sort         bannn
na$bana                  ana$b                   na$ba
nana$ba                  anana                   nana$
```

- We now know 5-mer composition of the circular string banana$
- Sorting gives us the first 5 columns of the matrix.

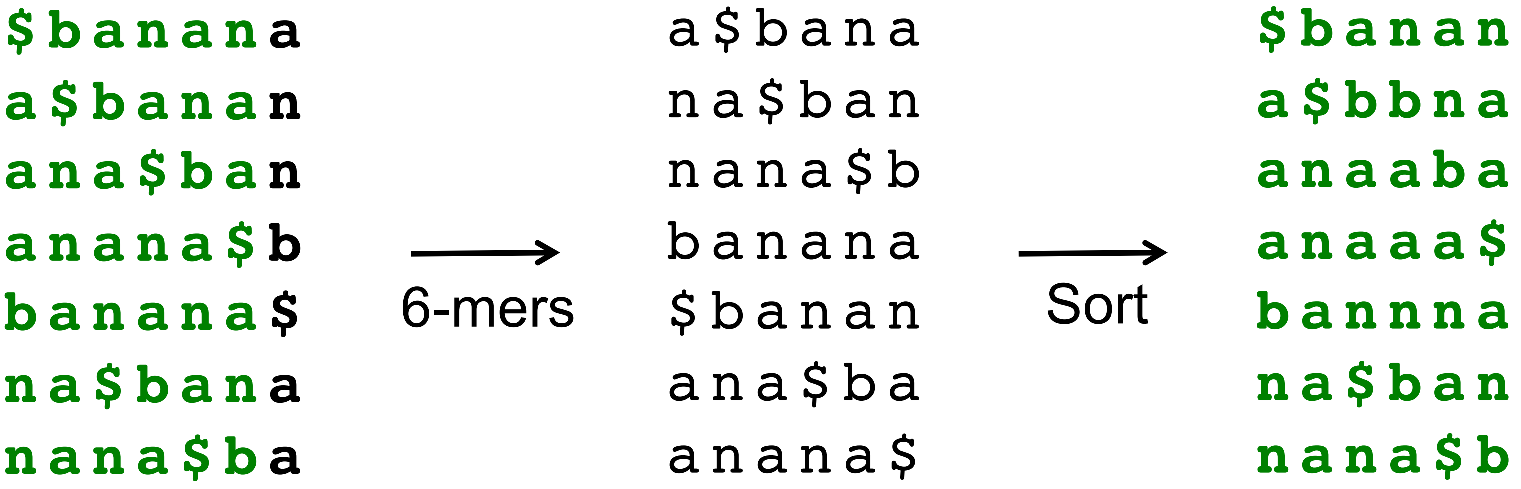
Reconstructing banana



```
$banana                a$bana                 $banan
a$banan                na$ban                 a$bbna
ana$ban                nana$b                 anaaba
anana$b   ------>      banana   ------>       anaaa$
banana$   6-mers       $banan   Sort          bannna
na$bana                ana$ba                 na$ban
nana$ba                anana$                 nana$b
```

- We now know 6-mer composition of the circular string banana$
- Sorting gives us the first 6 columns of the matrix.

Reconstructing banana



$banana
a$banan
ana$ban
anana$b
banana$
na$bana
nana$ba

6-mers

a$bana
na$ban
nana$b
banana
$banan
ana$ba
anana$

Sort

$banan
a$bbna
anaaba
anaaa$
bannna
na$ban
nana$b

- We now know 6-mer composition of the circular string banana$
- Sorting gives us the first 6 columns of the matrix.

Reconstructing banana

```
$banana
a$banan
ana$ban
anana$b
banana$
na$bana
nana$ba
```

- We now know the entire matrix!

- Taking all elements in the first row (after $) produces banana.

More Memory Issues

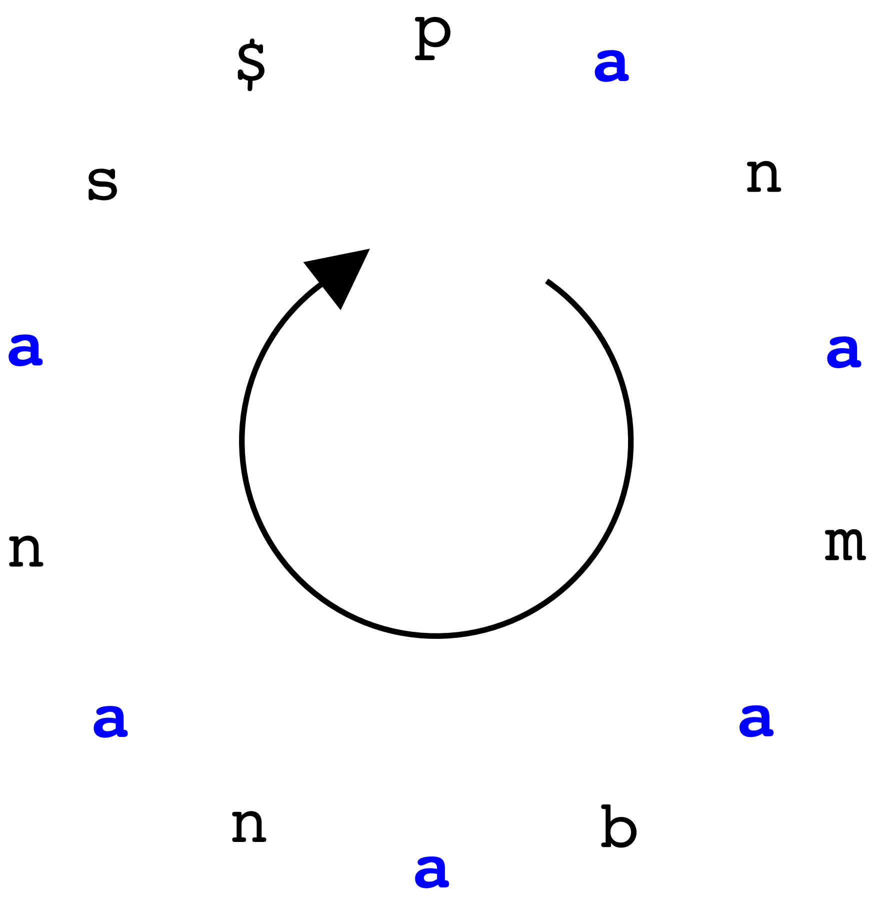
- Reconstructing *Genome* from *BWT*(*Genome*) required us to store |*Genome*| copies of |*Genome*|.

```
$banana
a$banan
ana$ban
anana$b
banana$
na$bana
nana$ba
```

- Can we invert BWT with less space?

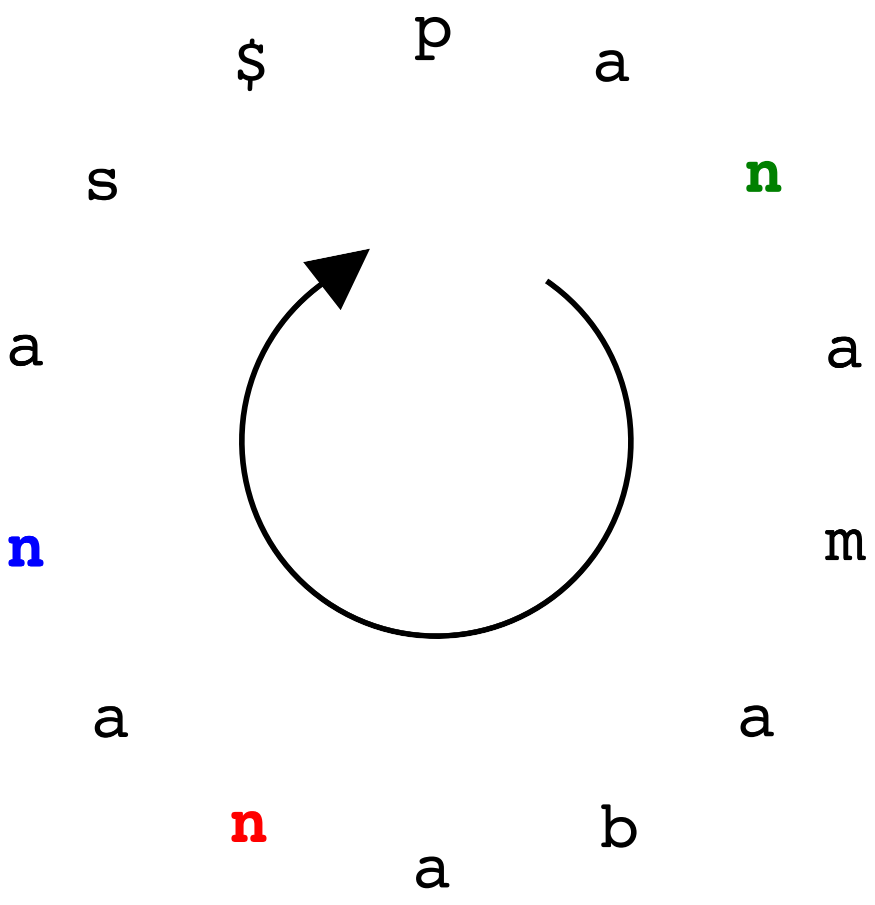
A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



A Strange Observation

```
$panamabananas
abananas$panam
amabananas$pan
anamabananas$p
ananas$panamab
anas$panamaban
as$panamabanan
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
s$panamabanana
```



Is It True in General?

```
   $panamabananas
1  abananas$panam
2  amabananas$pan
3  anamabananas$p
4  ananas$panamab
5  anas$panamaban
6  as$panamabanan
   bananas$panama
   mabananas$pana
   namabananas$pa
   nanas$panamaba
   nas$panamabana
   panamabananas$
   s$panamabanana
```

Chop off **a**

```
bananas$panam
mabananas$pan
namabananas$p
nanas$panamab
nas$panamaban
s$panamabanan
```

These strings are sorted

Is It True in General?

```
  $panamabananas
1 abananas$panam
2 amabananas$pan
3 anamabananas$p
4 ananas$panamab
5 anas$panamaban
6 as$panamabanan
  bananas$panama
  mabananas$pana
  namabananas$pa
  nanas$panamaba
  nas$panamabana
  panamabananas$
  s$panamabanana
```

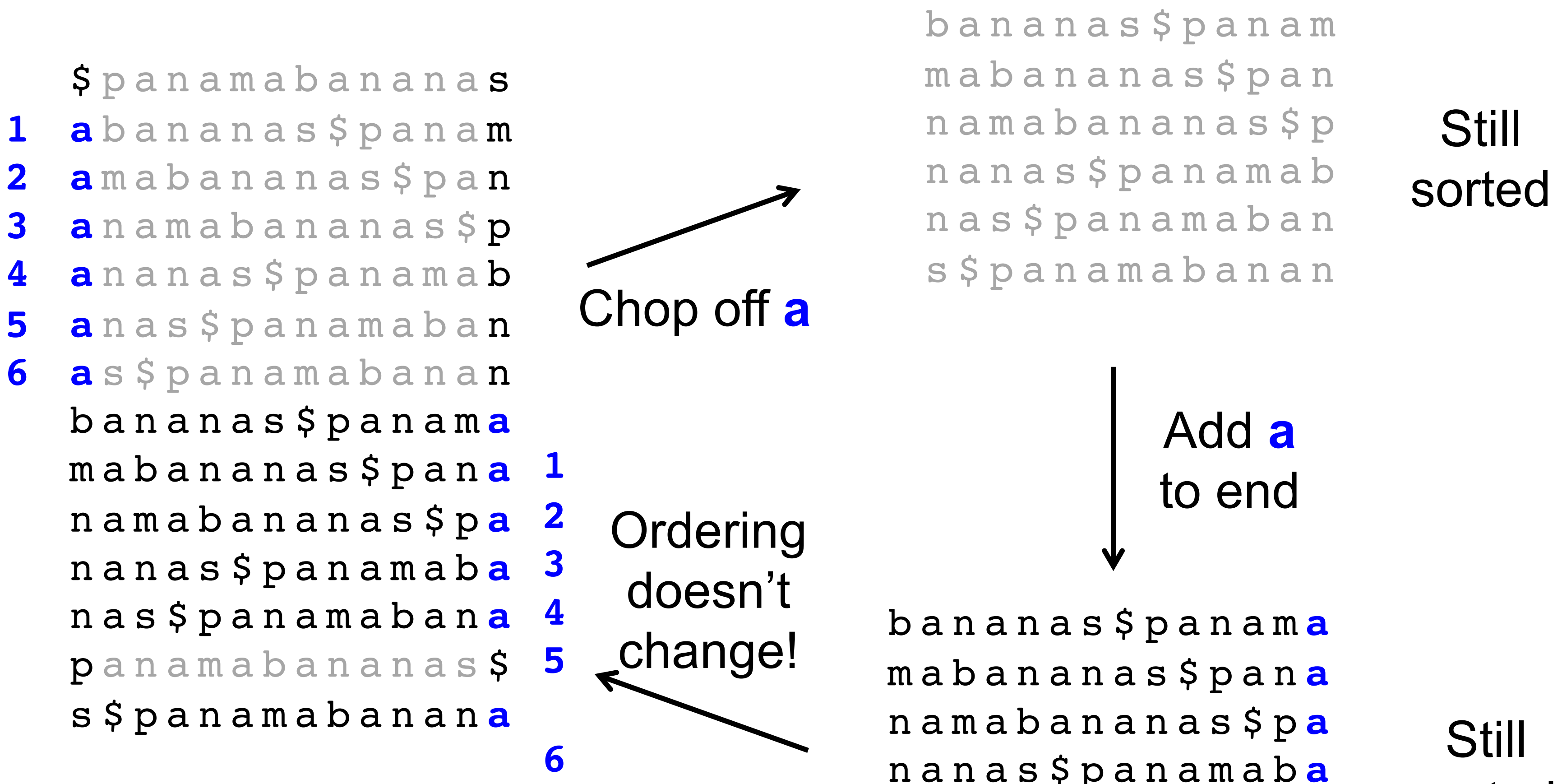
Chop off a

```
bananas$panam
mabananas$pan
namabananas$p
nanas$panamab
nas$panamaban
s$panamabanan
```

Still sorted

These strings are sorted

Is It True in General?



```
   $panamabananas
1  abananas$panam
2  amabananas$pan
3  anamabananas$p
4  ananas$panamab
5  anas$panamaban
6  as$panamabanan
   bananas$panama
   mabananas$pana  1
   namabananas$pa  2
   nanas$panamaba  3
   nas$panamabana  4
   panamabananas$  5
   s$panamabanana
                   6
```

Chop off a

```
bananas$panam
mabananas$pan
namabananas$p
nanas$panamab
nas$panamaban
s$panamabanan
```

Still sorted

Add a to end

```
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
s$panamabanana
```

Still sorted

Ordering doesn't change!

These strings are sorted

Is It True in General?

- **First-Last Property**: The *k*-th occurrence of *symbol* in *FirstColumn* and the *k*-th occurrence of *symbol* in *LastColumn* correspond to the same position of *symbol* in *Genome*.

$\$_1$panamabanana$s_1$

$a_1$bananas\$panam_1

$a_2$mabananas\$pan_1

$a_3$namabananas\$$p_1$

$a_4$nanas\$panamab_1

$a_5$nas\$panamaban_2

a_6s\$panamabanan_3

$b_1$ananas\$panama_1

$m_1$abananas\$pana_2

$n_1$amabananas\$pa_3

$n_2$anas\$panamaba_4

$n_3$as\$panamabana_5

$p_1$anamabananas$\$_1$

s_1\$panamabanana_6

More Efficient BWT Decompression

$\$_{1}$ panamabanana s_{1}

a_{1} bananas\$pana m_{1}

a_{2} mabananas\$pa n_{1}

a_{3} namabananas\$ p_{1}

a_{4} nanas\$panama b_{1}

a_{5} nas\$panamaba n_{2}

a_{6} s\$panamabana n_{3}

b_{1} ananas\$panam a_{1}

m_{1} abananas\$pan a_{2}

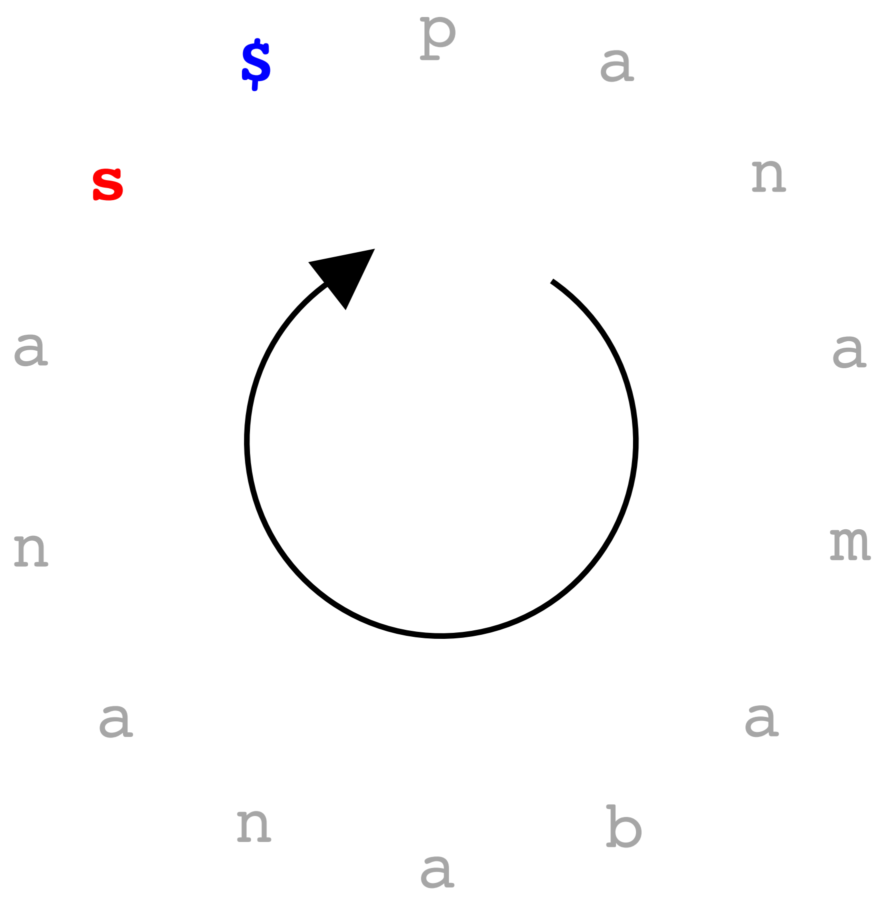
n_{1} amabananas\$p a_{3}

n_{2} anas\$panamab a_{4}

n_{3} as\$panamaban a_{5}

p_{1} anamabananas $\$_{1}$

s_{1} \$panamabanan a_{6}



More Efficient BWT Decompression

$\$_1$ panamabanana s_1

a_1 bananas\$pana m_1

a_2 mabananas\$pa n_1

a_3 namabananas\$ p_1

a_4 nanas\$panama b_1

a_5 nas\$panamaba n_2

a_6 s\$panamabana n_3

b_1 ananas\$panam a_1

m_1 abananas\$pan a_2

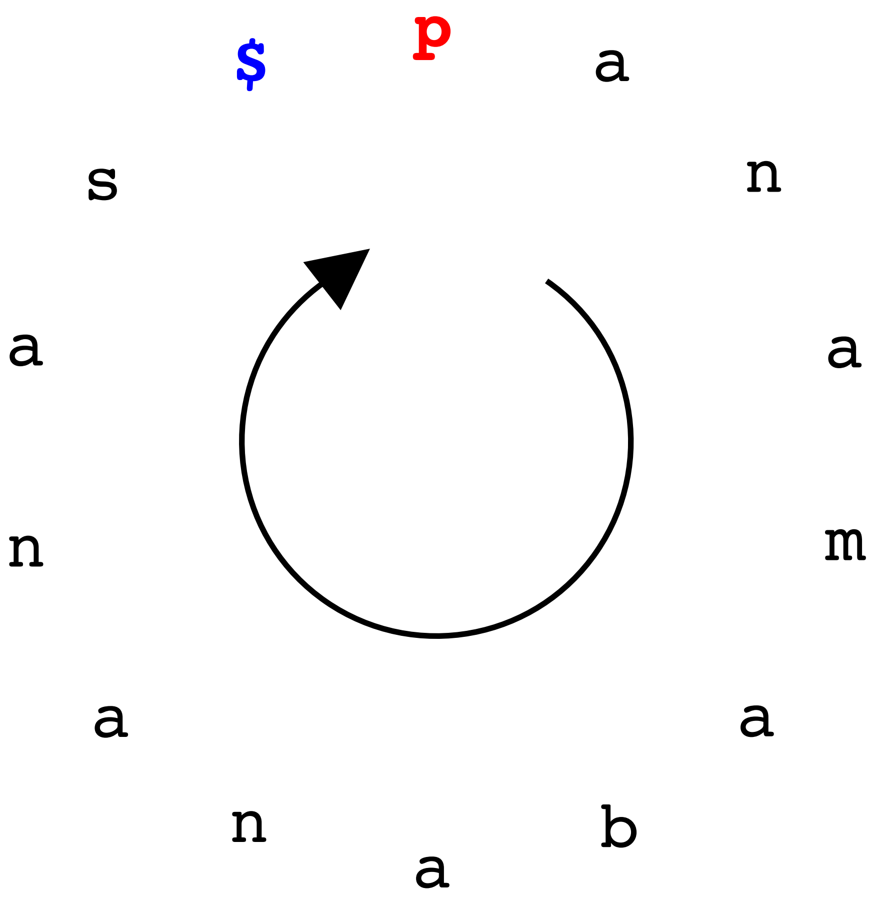
n_1 amabananas\$p a_3

n_2 anas\$panamab a_4

n_3 as\$panamaban a_5

p_1 anamabananas $\$_1$

s_1 \$panamabanan a_6



More Efficient BWT Decompression

$\$_1$ panamabanana s_1

a_1 bananas\$pana m_1

a_2 mabananas\$pa n_1

a_3 namabananas\$ p_1

a_4 nanas\$panama b_1

a_5 nas\$panamaba n_2

a_6 s\$panamabana n_3

b_1 ananas\$panam a_1

m_1 abananas\$pan a_2

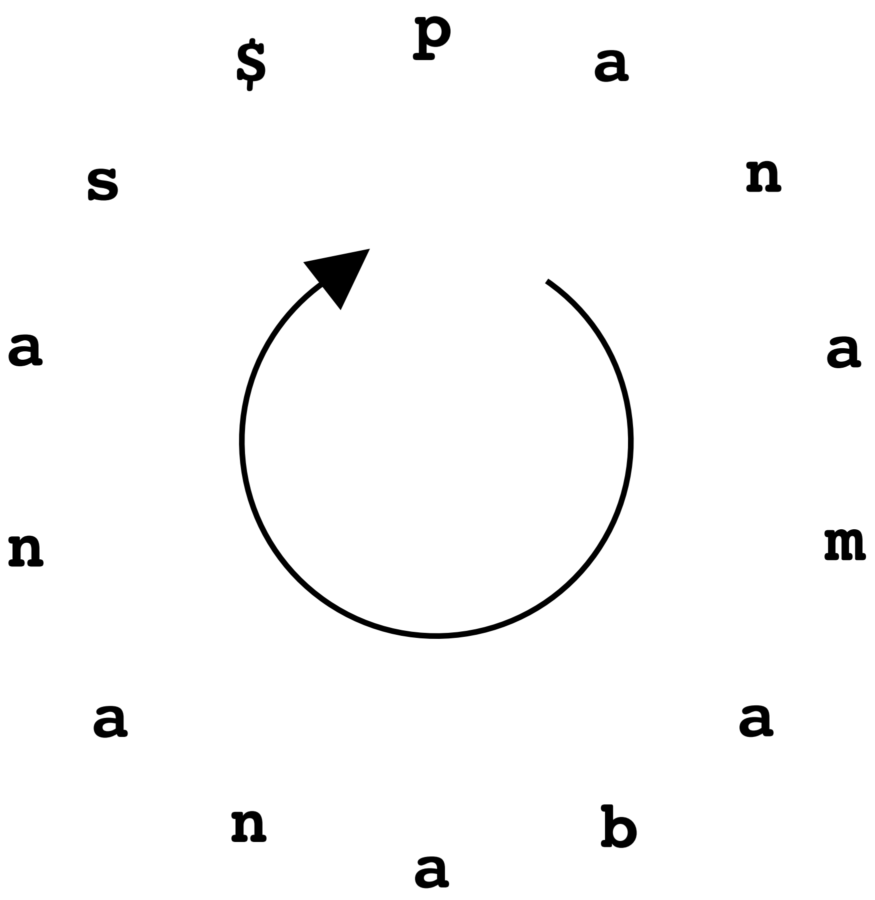
n_1 amabananas\$p a_3

n_2 anas\$panamab a_4

n_3 as\$panamaban a_5

p_1 anamabananas $\$_1$

s_1 \$panamabanan a_6



- Memory: 2|*Genome*| = O(|*Genome*|).

Recalling Our Goal

- Suffix Tree Pattern Matching:
  - Runtime: O(|*Genome*| + |*Patterns*|)
  - Memory: O(|*Genome*|)
  - Problem: suffix tree takes 20 x |*Genome*| space

- Can we use BWT(*Genome*) as our data structure instead?

Finding Pattern Matches Using BWT

- Searching for **ana** in p**ana**mab**ananana**s

```
$_1 panamabanana s_1
a_1 bananas$pana m_1
a_2 mabananas$pa n_1
a_3 na mabananas$ p_1
a_4 na nas$panama b_1
a_5 na s$panamaba n_2
a_6 s$panamabana n_3
b_1 ananas$panam a_1
m_1 abananas$pan a_2
n_1 amabananas$p a_3
n_2 anas$panamab a_4
n_3 as$panamaban a_5
p_1 anamabananas $_1
s_1 $panamabanan a_6
```

Finding Pattern Matches Using BWT

- Searching for `ana` in `panamabananas`

$\$_1$	panamabanana	s_1
a_1	bananas$panam	m_1
a_2	mabananas$pan	n_1
a_3	namabananas$	p_1
a_4	nanas$panama	b_1
a_5	nas$panamaban	n_2
a_6	s$panamabanan	n_3
b_1	ananas$panam	a_1
m_1	abananas$pan	a_2
n_1	amabananas$p	a_3
n_2	anas$panamab	a_4
n_3	as$panamaban	a_5
p_1	anamabananas	$\$_1$
s_1	$panamabanan	a_6

Finding Pattern Matches Using BWT

- Searching for **ana** in **panamabananas**

```
$1 panamabanana s1
a1 bananas$pana m1
a2 mabananas$pa n1
a3 namabananas$ p1
a4 nanas$panama b1
a5 nas$panamaba n2
a6 s$panamabana n3
b1 ananas$panam a1
m1 abananas$pan a2
n1 amabananas$p a3
n2 anas$panamab a4
n3 as$panamaban a5
p1 anamabananas $1
s1 $panamabanan a6
```

Finding Pattern Matches Using BWT

- Searching for **ana** in panamabananas

$\$_1$panamabanana$s_1$

$a_1$bananas\$panam_1

$a_2$mabananas\$pan_1

$a_3$namabananas\$$p_1$

$a_4$nanas\$panamab_1

$a_5$nas\$panamaban_2

a_6s\$panamabanan_3

$b_1$ananas\$panama_1

$m_1$abananas\$pana_2

$n_1$amabananas\$pa_3

$n_2$anas\$panamaba_4

$n_3$as\$panamabana_5

$p_1$anamabananas$\$_1$

s_1\$panamabanana_6

Where Are the Matches?

- **Multiple Pattern Matching Problem:**
  - **Input:** A collection of strings *Patterns* and a string *Genome.*
  - **Output:** All **positions** in *Genome* where one of *Patterns* appears as a substring.

- Where are the **positions**? BWT has not revealed them.

Where Are the Matches?

- Example: We know that **ana** occurs 3 times, but where?

$\$_1\texttt{panamabananas}s_1$

$a_1\texttt{bananas\$panam}m_1$

$a_2\texttt{mabananas\$pan}n_1$

$\mathbf{a_3\texttt{na}}\texttt{mabananas\$}p_1$

$\mathbf{a_4\texttt{na}}\texttt{nas\$panama}b_1$

$\mathbf{a_5\texttt{na}}\texttt{s\$panamaba}n_2$

$a_6\texttt{s\$panamabana}n_3$

$b_1\texttt{ananas\$panam}a_1$

$m_1\texttt{abananas\$pan}a_2$

$n_1\texttt{amabananas\$p}a_3$

$n_2\texttt{anas\$panamab}a_4$

$n_3\texttt{as\$panamaban}a_5$

$p_1\texttt{anamabananas}\$_1$

$s_1\texttt{\$panamabanan}a_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

$\$_1$ panamabanana s_1

a_1 bananas\$pana m_1

a_2 mabananas\$pa n_1

a_3 namabananas\$ p_1

a_4 nanas\$panama b_1

a_5 nas\$panamaba n_2

a_6 s\$panamabana n_3

b_1 ananas\$panam a_1

m_1 abananas\$pan a_2

n_1 amabananas\$p a_3

n_2 anas\$panamab a_4

n_3 as\$panamaban a_5

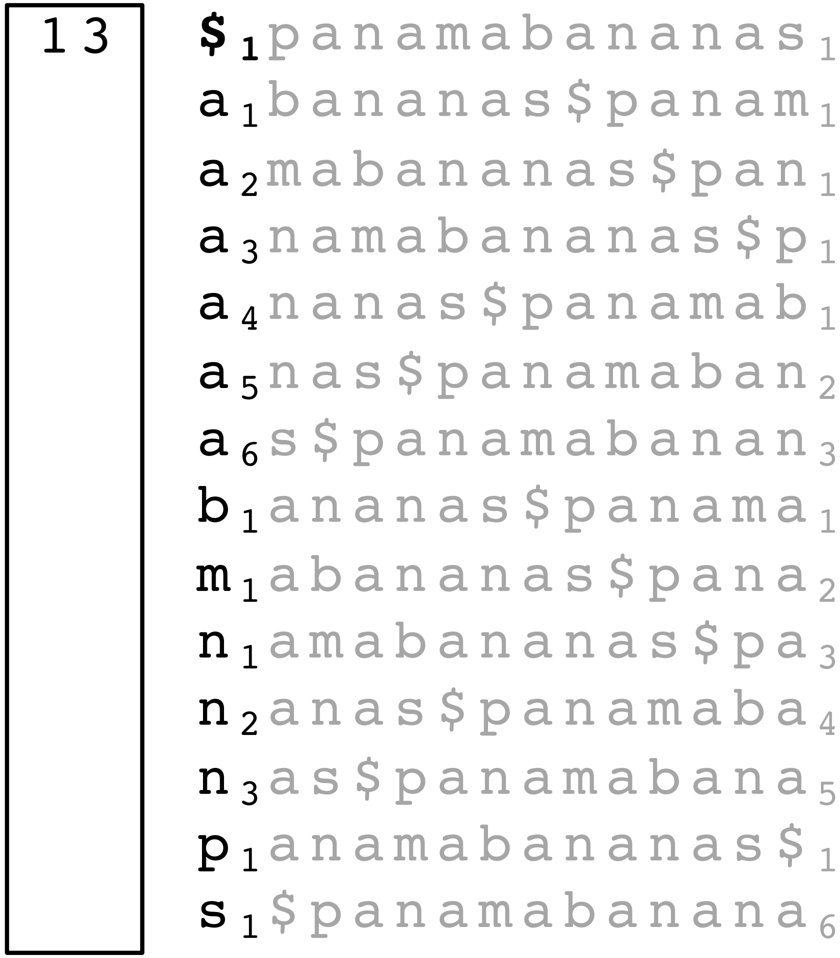
p_1 anamabananas $\$_1$

s_1 \$panamabanan a_6

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabananas$



13

$\mathtt{\$_1 panamabananas_1}$

$\mathtt{a_1 bananas\$panam_1}$

$\mathtt{a_2 mabananas\$pan_1}$

$\mathtt{a_3 namabananas\$p_1}$

$\mathtt{a_4 nanas\$panamab_1}$

$\mathtt{a_5 nas\$panamaban_2}$

$\mathtt{a_6 s\$panamabanan_3}$

$\mathtt{b_1 ananas\$panama_1}$

$\mathtt{m_1 abananas\$pana_2}$

$\mathtt{n_1 amabananas\$pa_3}$

$\mathtt{n_2 anas\$panamaba_4}$

$\mathtt{n_3 as\$panamabana_5}$

$\mathtt{p_1 anamabananas\$_1}$

$\mathtt{s_1 \$panamabanana_6}$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panam**abananas$**

13	$\$_1$panamabanana$s_1$
5	**$a_1$bananas$\$$**panam_1
	$a_2$mabananaspan_1$
	$a_3$namabananas$$p_1$
	$a_4$nanas$panama$b_1$
	$a_5$nas$panamaba$n_2$
	a_6s$panamabana$n_3$
	$b_1$ananas$panam$a_1$
	$m_1$abananaspana_2$
	$n_1$amabananas$p$a_3$
	$n_2$anas$panamab$a_4$
	$n_3$as$panamaban$a_5$
	$p_1$anamabananas$\$_1$
	$s_1$$panamabanana_6

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

pan**amabananas$**

13	**$\$_1$**panamabananas$_1$
5	**a$_1$bananas$**panam$_1$
3	**a$_2$mabananas$**pan$_1$
	a$_3$namabananas$p$_1$
	a$_4$nanas$panamab$_1$
	a$_5$nas$panamaban$_2$
	a$_6$s$panamabanan$_3$
	b$_1$ananas$panama$_1$
	m$_1$abananas$pana$_2$
	n$_1$amabananaspa_3$
	n$_2$anas$panamaba$_4$
	n$_3$as$panamabana$_5$
	p$_1$anamabananas$$_1$
	s$_1$$panamabanana$_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

p**anamabananas$**

13	$\$_{1}\text{panamabananas}_{1}$
5	$\text{a}_{1}\text{bananas\$panam}_{1}$
3	$\text{a}_{2}\text{mabananas\$pan}_{1}$
1	$\text{a}_{3}\text{namabananas\$p}_{1}$
	$\text{a}_{4}\text{nanas\$panamab}_{1}$
	$\text{a}_{5}\text{nas\$panamaban}_{2}$
	$\text{a}_{6}\text{s\$panamabanan}_{3}$
	$\text{b}_{1}\text{ananas\$panama}_{1}$
	$\text{m}_{1}\text{abananas\$pana}_{2}$
	$\text{n}_{1}\text{amabananas\$pa}_{3}$
	$\text{n}_{2}\text{anas\$panamaba}_{4}$
	$\text{n}_{3}\text{as\$panamabana}_{5}$
	$\text{p}_{1}\text{anamabananas\$}_{1}$
	$\text{s}_{1}\text{\$panamabanana}_{6}$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamab**ananas$**

13	$\mathbf{\$_1}\mathrm{panamabananas}_1$
5	$\mathbf{a_1bananas\$}\mathrm{panam}_1$
3	$\mathbf{a_2mabananas\$}\mathrm{pan}_1$
1	$\mathbf{a_3namabananas\$}\mathrm{p}_1$
7	$\mathbf{a_4nanas\$}\mathrm{panamab}_1$
	$a_5\mathrm{nas\$panamaban}_2$
	$a_6\mathrm{s\$panamabanan}_3$
	$b_1\mathrm{ananas\$panama}_1$
	$m_1\mathrm{abananas\$pana}_2$
	$n_1\mathrm{amabananas\$pa}_3$
	$n_2\mathrm{anas\$panamaba}_4$
	$n_3\mathrm{as\$panamabana}_5$
	$p_1\mathrm{anamabananas\$}_1$
	$s_1\mathrm{\$panamabanana}_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamaban**anas$**

13	$\mathbf{\$_1}\mathrm{panamabananas}_1$
5	$\mathbf{a_1bananas\$}\mathrm{panam}_1$
3	$\mathbf{a_2mabananas\$}\mathrm{pan}_1$
1	$\mathbf{a_3namabananas\$}\mathrm{p}_1$
7	$\mathbf{a_4nanas\$}\mathrm{panamab}_1$
9	$\mathbf{a_5nas\$}\mathrm{panamaban}_2$
	$\mathrm{a}_6\mathrm{s\$panamabanan}_3$
	$\mathrm{b}_1\mathrm{ananas\$panama}_1$
	$\mathrm{m}_1\mathrm{abananas\$pana}_2$
	$\mathrm{n}_1\mathrm{amabananas\$pa}_3$
	$\mathrm{n}_2\mathrm{anas\$panamaba}_4$
	$\mathrm{n}_3\mathrm{as\$panamabana}_5$
	$\mathrm{p}_1\mathrm{anamabananas\$}_1$
	$\mathrm{s}_1\mathrm{\$panamabanana}_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabanan**as$**

13	$\mathbf{\$_1}\text{panamabananas}_1$
5	$\mathbf{a_1bananas\$}\text{panam}_1$
3	$\mathbf{a_2mabananas\$}\text{pan}_1$
1	$\mathbf{a_3namabananas\$}\text{p}_1$
7	$\mathbf{a_4nanas\$}\text{panamab}_1$
9	$\mathbf{a_5nas\$}\text{panamaban}_2$
11	$\mathbf{a_6s\$}\text{panamabanan}_3$
	$b_1\text{ananas\$panama}_1$
	$m_1\text{abananas\$pana}_2$
	$n_1\text{amabananas\$pa}_3$
	$n_2\text{anas\$panamaba}_4$
	$n_3\text{as\$panamabana}_5$
	$p_1\text{anamabananas\$}_1$
	$s_1\text{\$panamabanana}_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panama**bananas$**

13	$\mathtt{\$_1panamabananas_1}$
5	$\mathtt{a_1bananas\$panam_1}$
3	$\mathtt{a_2mabananas\$pan_1}$
1	$\mathtt{a_3namabananas\$p_1}$
7	$\mathtt{a_4nanas\$panamab_1}$
9	$\mathtt{a_5nas\$panamaban_2}$
11	$\mathtt{a_6s\$panamabanan_3}$
6	$\mathtt{b_1ananas\$panama_1}$
	$\mathtt{m_1abananas\$pana_2}$
	$\mathtt{n_1amabananas\$pa_3}$
	$\mathtt{n_2anas\$panamaba_4}$
	$\mathtt{n_3as\$panamabana_5}$
	$\mathtt{p_1anamabananas\$_1}$
	$\mathtt{s_1\$panamabanana_6}$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabana**nas$**

13	$\mathbf{\$_1}\text{panamabananas}_1$
5	$\mathbf{a_1bananas\$}\text{panam}_1$
3	$\mathbf{a_2mabananas\$}\text{pan}_1$
1	$\mathbf{a_3namabananas\$}\text{p}_1$
7	$\mathbf{a_4nanas\$}\text{panamab}_1$
9	$\mathbf{a_5nas\$}\text{panamaban}_2$
11	$\mathbf{a_6s\$}\text{panamabanan}_3$
6	$\mathbf{b_1ananas\$}\text{panama}_1$
4	$\mathbf{m_1abananas\$}\text{pana}_2$
2	$\mathbf{n_1amabananas\$}\text{pa}_3$
8	$\mathbf{n_2anas\$}\text{panamaba}_4$
10	$\mathbf{n_3as\$}\text{panamabana}_5$
	$\text{p}_1\text{anamabananas\$}_1$
	$\text{s}_1\text{\$panamabanana}_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabananas$

13	$\mathtt{\$_{1}panamabananas_{1}}$
5	$\mathtt{a_{1}bananas\$panam_{1}}$
3	$\mathtt{a_{2}mabananas\$pan_{1}}$
1	$\mathtt{a_{3}namabananas\$p_{1}}$
7	$\mathtt{a_{4}nanas\$panamab_{1}}$
9	$\mathtt{a_{5}nas\$panamaban_{2}}$
11	$\mathtt{a_{6}s\$panamabanan_{3}}$
6	$\mathtt{b_{1}ananas\$panama_{1}}$
4	$\mathtt{m_{1}abananas\$pana_{2}}$
2	$\mathtt{n_{1}amabananas\$pa_{3}}$
8	$\mathtt{n_{2}anas\$panamaba_{4}}$
10	$\mathtt{n_{3}as\$panamabana_{5}}$
0	$\mathtt{p_{1}anamabananas\$_{1}}$
	$\mathtt{s_{1}\$panamabanana_{6}}$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabanana**s$**

13	$\texttt{\$}_1\texttt{panamabananas}_1$
5	$\texttt{a}_1\texttt{bananas\$panam}_1$
3	$\texttt{a}_2\texttt{mabananas\$pan}_1$
1	$\texttt{a}_3\texttt{namabananas\$p}_1$
7	$\texttt{a}_4\texttt{nanas\$panamab}_1$
9	$\texttt{a}_5\texttt{nas\$panamaban}_2$
11	$\texttt{a}_6\texttt{s\$panamabanan}_3$
6	$\texttt{b}_1\texttt{ananas\$panama}_1$
4	$\texttt{m}_1\texttt{abananas\$pana}_2$
2	$\texttt{n}_1\texttt{amabananas\$pa}_3$
8	$\texttt{n}_2\texttt{anas\$panamaba}_4$
10	$\texttt{n}_3\texttt{as\$panamabana}_5$
0	$\texttt{p}_1\texttt{anamabananas\$}_1$
12	$\texttt{s}_1\texttt{\$panamabanana}_6$

Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

panamabananas$

13	$\$_1\text{panamabanana}s_1$
5	$a_1\text{bananas\$pana}m_1$
3	$a_2\text{mabananas\$pa}n_1$
1	$a_3\text{namabananas\$}p_1$
7	$a_4\text{nanas\$panama}b_1$
9	$a_5\text{nas\$panamaba}n_2$
11	$a_6\text{s\$panamabana}n_3$
6	$b_1\text{ananas\$panam}a_1$
4	$m_1\text{abananas\$pan}a_2$
2	$n_1\text{amabananas\$p}a_3$
8	$n_2\text{anas\$panamab}a_4$
10	$n_3\text{as\$panamaban}a_5$
0	$p_1\text{anamabananas}\$_1$
12	$s_1\text{\$panamabanan}a_6$

Using the Suffix Array to Find Matches

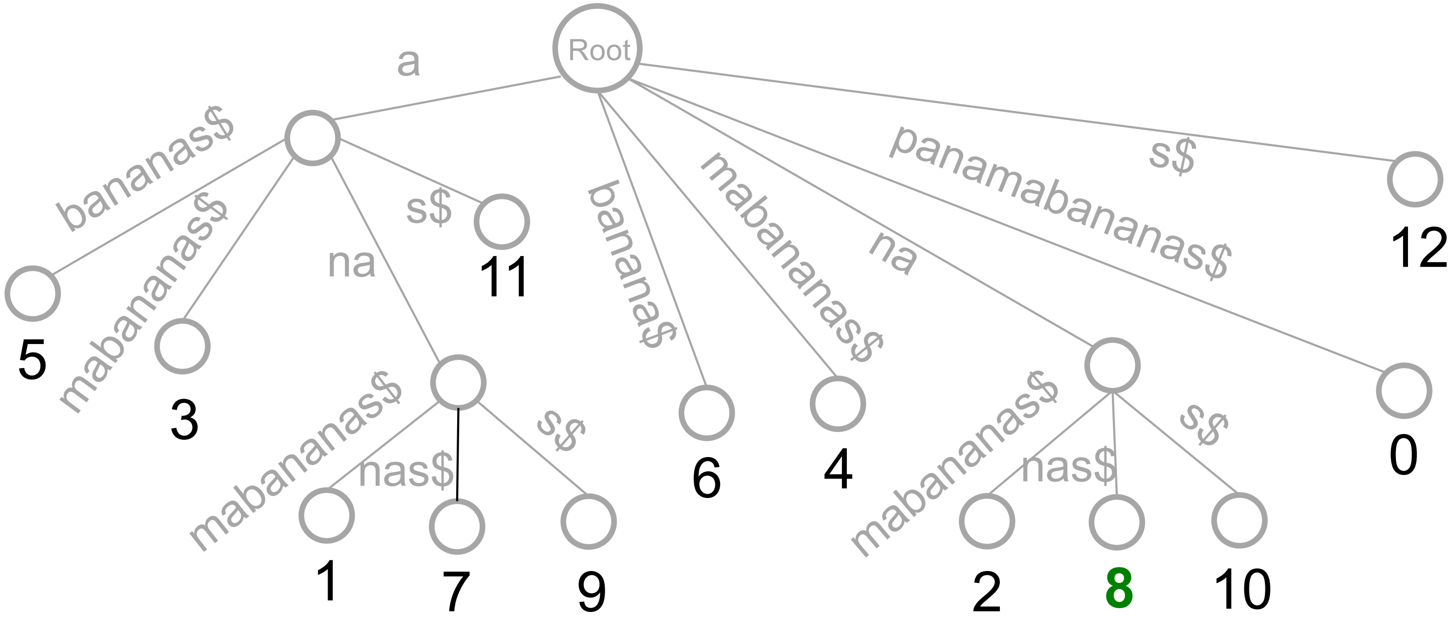
- **Suffix array:** holds starting position of each suffix beginning a row.

- Thus, **ana** occurs at positions **1**, **7**, **9** of panamabananas$.

13	$\$_1\texttt{panamabanana}s_1$
5	$a_1\texttt{bananas\$pana}m_1$
3	$a_2\texttt{mabananas\$pa}n_1$
1	$a_3\texttt{namabananas\$}p_1$
7	$a_4\texttt{nanas\$panama}b_1$
9	$a_5\texttt{nas\$panamaba}n_2$
11	$a_6\texttt{s\$panamabana}n_3$
6	$b_1\texttt{ananas\$panam}a_1$
4	$m_1\texttt{abananas\$pan}a_2$
2	$n_1\texttt{amabananas\$p}a_3$
8	$n_2\texttt{anas\$panamab}a_4$
10	$n_3\texttt{as\$panamaban}a_5$
0	$p_1\texttt{anamabananas}\$_1$
12	$s_1\texttt{\$panamabanan}a_6$

The Suffix Array: Memory Once Again

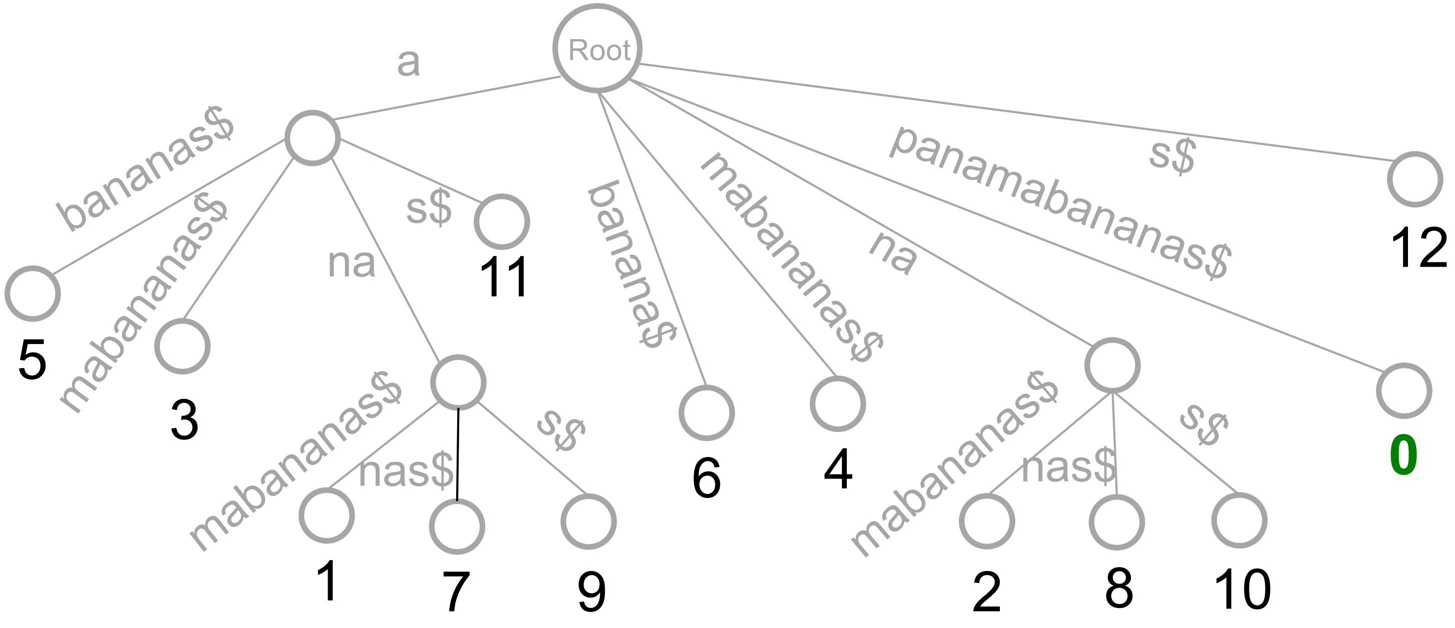
- Memory: ~ 4 x |*Genome*|.



[13 5 3 1 7 9 11 6 4 2 **8** 10 0 12]

The Suffix Array: Memory Once Again

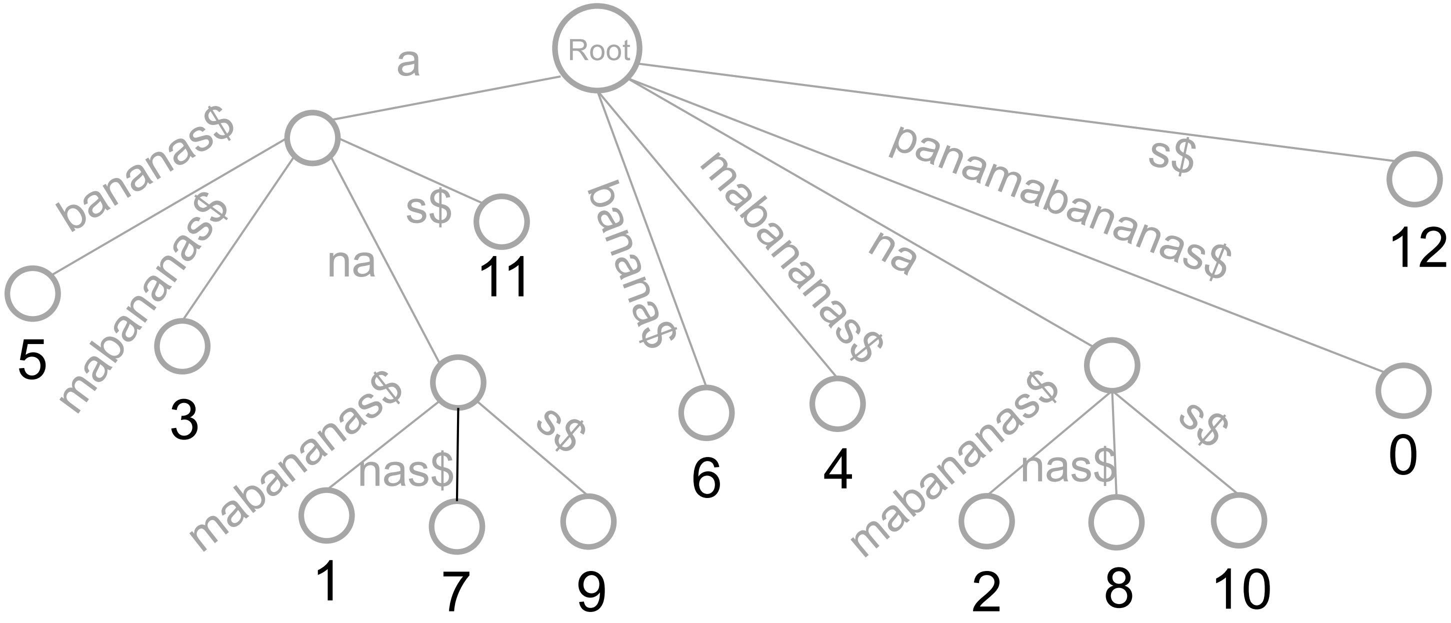
- Memory: ~ 4 x |*Genome*|.



[13 5 3 1 7 9 11 6 4 2 8 10 0 12]

The Suffix Array: Memory Once Again

- Memory: ~ 4 x |*Genome*|.



[13 5 3 1 7 9 11 6 4 2 8 10 0 12]

Returning to Our Original Problem

- We need to look at INEXACT matching in order to find variants.

- **Approx. Pattern Matching Problem**:
  - **Input**: A string *Pattern*, a string *Genome*, and an integer *d*.
  - **Output:** All positions in *Genome* where *Pattern* appears as a substring with at most *d* mismatches.

Returning to Our Original Problem

- We need to look at INEXACT matching in order to find variants.

- Multiple **Approx. Pattern Matching Problem**:
  - **Input**: A **collection** of strings *Patterns*, a string *Genome*, and an integer *d*.
  - **Output:** All positions in *Genome* where a string from *Patterns* appears as a substring with at most *d* mismatches.

Method 1: Seeding

- Say that *Pattern* appears in *Genome* with 1 mismatch:

```
Pattern          actt|ggct
Genome   ...ggcacacta|ggctcc...
```

Method 1: Seeding

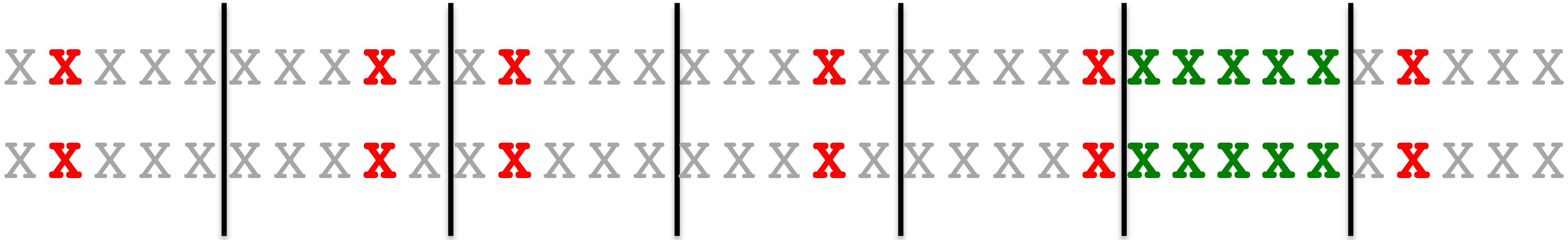
- Say that *Pattern* appears in *Genome* with 1 mismatch:

```
Pattern         actt|ggct
Genome    …ggcacacta|ggctcc…
```

- One of the substrings must match!

Method 1: Seeding

- **Theorem:** If *Pattern* occurs in *Genome* with *d* mismatches, then we can divide *Pattern* into *d* + 1 “equal” pieces and find at least one exact match.



Method 1: Seeding

- Say we are looking for at most *d* mismatches.
- Divide each of our strings into *d* + 1 smaller pieces, called **seeds**.
- Check if each *Pattern* has a seed that matches *Genome* exactly.
- If so, check the entire *Pattern* against *Genome*.

Method 2: BWT Saves the Day Again

- Recall: searching for a**na** in panamabananas

Now we extend all strings with at most 1 mismatch.

	# Mismatches
$\mathtt{\$}_1\mathtt{panamabanana}\mathtt{s}_1$	
$\mathtt{a}_1\mathtt{bananas\$pana}\mathtt{m}_1$	1
$\mathtt{a}_2\mathtt{mabananas\$pa}\mathtt{n}_1$	0
$\mathtt{a}_3\mathtt{namabananas\$}\mathtt{p}_1$	1
$\mathtt{a}_4\mathtt{nanas\$panama}\mathtt{b}_1$	1
$\mathtt{a}_5\mathtt{nas\$panamaba}\mathtt{n}_2$	0
$\mathtt{a}_6\mathtt{s\$panamabana}\mathtt{n}_3$	0
$\mathtt{b}_1\mathtt{ananas\$panam}\mathtt{a}_1$	
$\mathtt{m}_1\mathtt{abananas\$pan}\mathtt{a}_2$	
$\mathtt{n}_1\mathtt{amabananas\$p}\mathtt{a}_3$	
$\mathtt{n}_2\mathtt{anas\$panamab}\mathtt{a}_4$	
$\mathtt{n}_3\mathtt{as\$panamaban}\mathtt{a}_5$	
$\mathtt{p}_1\mathtt{anamabananas}\mathtt{\$}_1$	
$\mathtt{s}_1\mathtt{\$panamabanan}\mathtt{a}_6$	

Method 2: BWT Saves the Day Again

- Recall: searching for **ana** in panamabananas

One string produces a second mismatch (the $), so we discard it.

	# Mismatches
$\$_1$ panamabanana s_1	
a_1 bananas\$pana m_1	
a_2 mabananas\$pa n_1	
a_3 namabananas\$ p_1	
a_4 nanas\$panama b_1	
a_5 nas\$panamaba n_2	
a_6 s\$panamabana n_3	
b_1 ananas\$panam a_1	1
m_1 abananas\$pan a_2	1
n_1 amabananas\$p a_3	0
n_2 anas\$panamab a_4	0
n_3 as\$panamaban a_5	0
p_1 anamabananas $\$_1$	2
s_1 \$panamabanan a_6	

Method 2: BWT Saves the Day Again

- Recall: searching for **ana** in panamabananas

In the end, we have five 3-mers with at most 1 mismatch.

```
                                  # Mismatches
$1 panamabanana s1
a1 banasnas$panam m1  
```

Method 2: BWT Saves the Day Again

- Recall: searching for **ana** in panamab**ana**nas

In the end, we have five 3-mers with at most 1 mismatch.

			Suffix Array
$\$_1$	panamabanana	s_1	
a_1	bananas\$pana	m_1	5
a_2	mabananas\$pa	n_1	3
a_3	namabananas\$	p_1	1
a_4	**na**nas\$panama	b_1	**7**
a_5	nas\$panamaba	n_2	9
a_6	s\$panamabana	n_3	
b_1	ananas\$panam	a_1	
m_1	abananas\$pan	a_2	
n_1	amabananas\$p	a_3	
n_2	anas\$panamab	a_4	
n_3	as\$panamaban	a_5	
p_1	anamabananas	$\$_1$	
s_1	\$panamabanan	a_6	

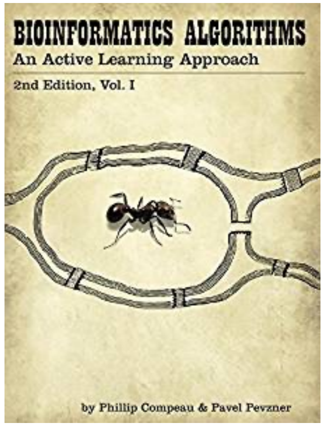
Method 2: BWT Saves the Day Again

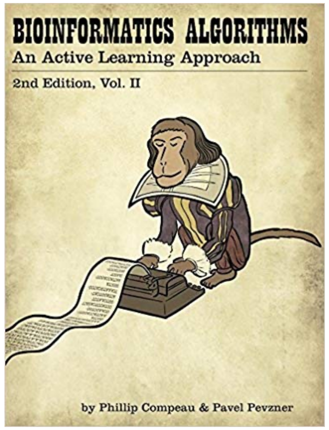
- Recall: searching for **ana** in panamabanan**ana**s

In the end, we have five 3-mers with at most 1 mismatch.

			Suffix Array
$\$_1$	panamabanana	s_1	
a_1	bananas\$panam	m_1	5
a_2	mabananas\$pan	n_1	3
a_3	namabananas\$p	p_1	1
a_4	nanas\$panamab	b_1	7
a_5	**na**s\$panamaban	n_2	**9**
a_6	s\$panamabanan	n_3	
b_1	ananas\$panama	a_1	
m_1	abananas\$pana	a_2	
n_1	amabananas\$pa	a_3	
n_2	anas\$panamaba	a_4	
n_3	as\$panamabana	a_5	
p_1	anamabananas\$	$\$_1$	
s_1	\$panamabanana	a_6	

Reference for this section


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. I
by Phillip Compeau & Pavel Pevzner


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. II
by Phillip Compeau & Pavel Pevzner

- Chapter 9 Vol 2 (pag 120-173)

Useful link for computing BWT:
http://www.allisons.org/ll/AlgDS/Strings/BWT/

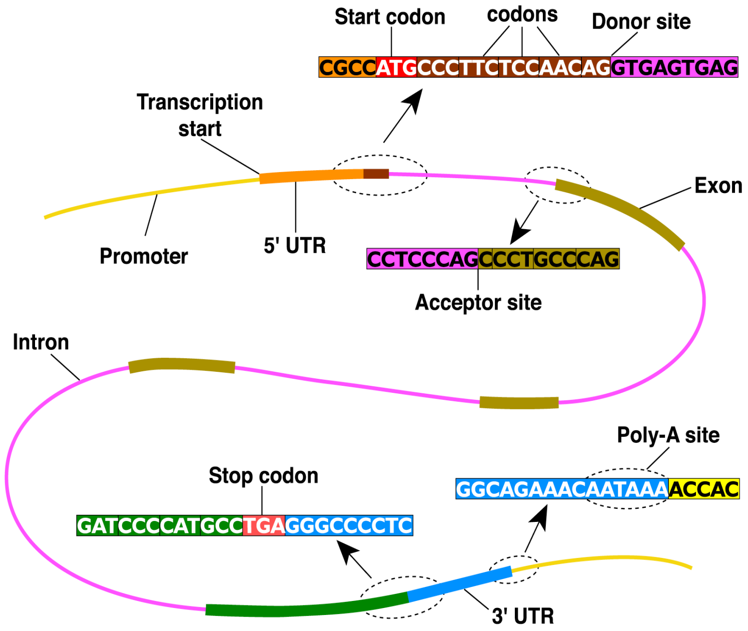
Section 7

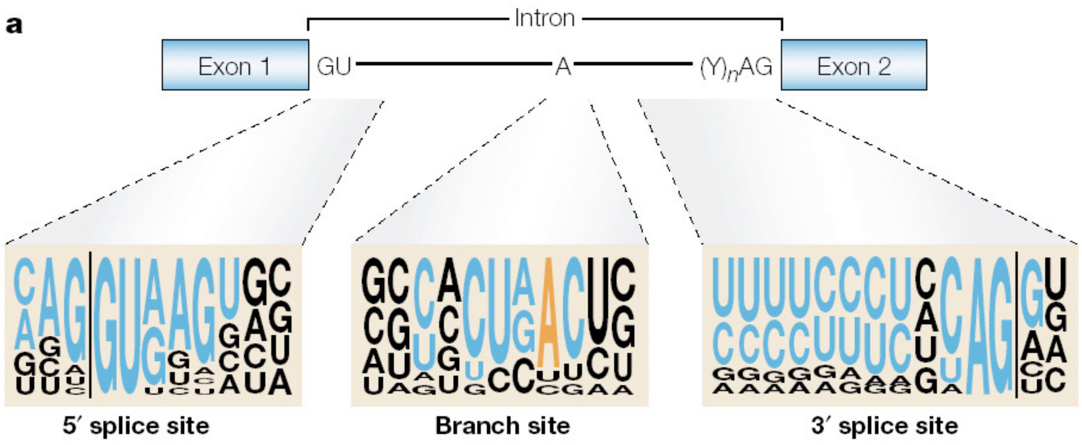
Algorithms to find parts

- Algorithm: Viterbi
- Algorithm: Forward
- Algorithm: Backward

Biologists need algorithms to identify genes and gene parts

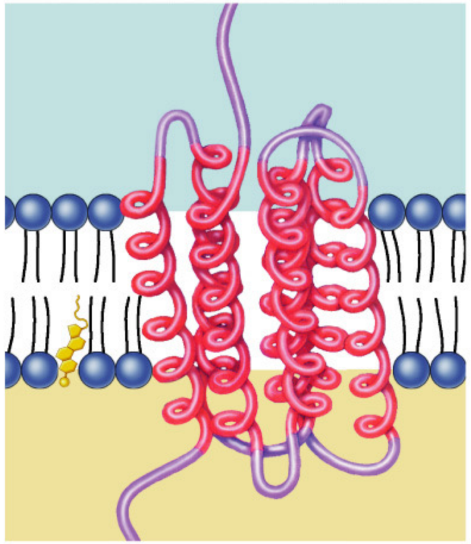
The gene information starts with the promoter, which is followed by a transcribed (i.e. RNA) but non-coding (i.e. not translated) region called 5’ untranslated region (5’ UTR). The initial exon contains the start codon which is usually ATG. There is an alternating series of introns and exons, followed 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 the amino acid 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.

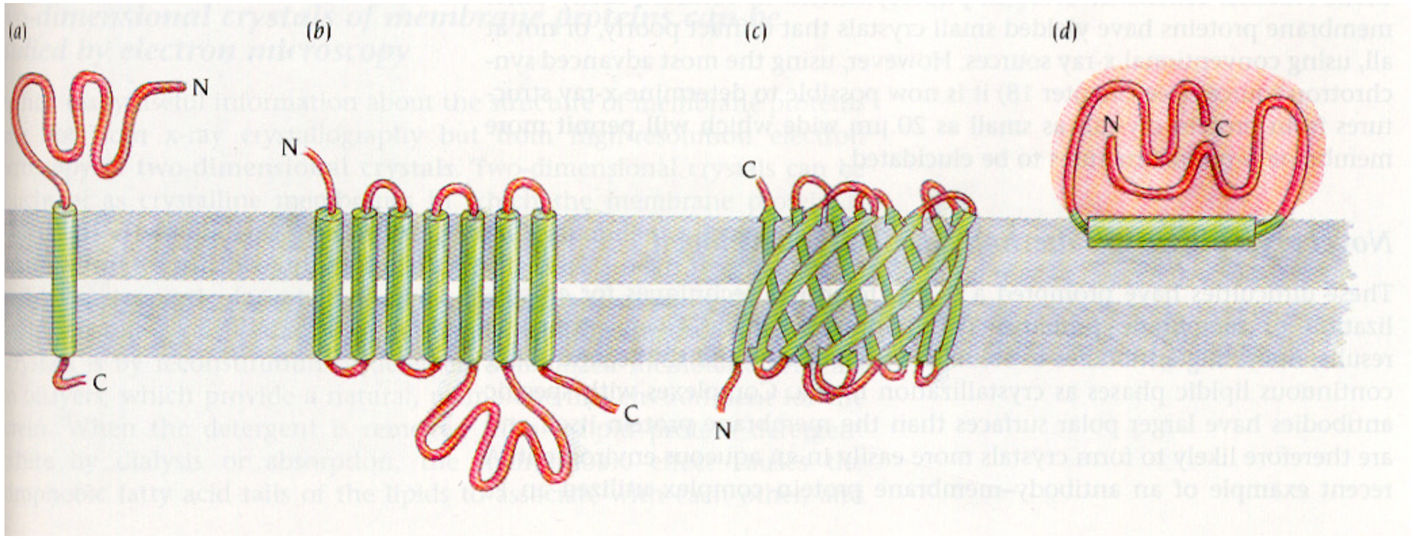




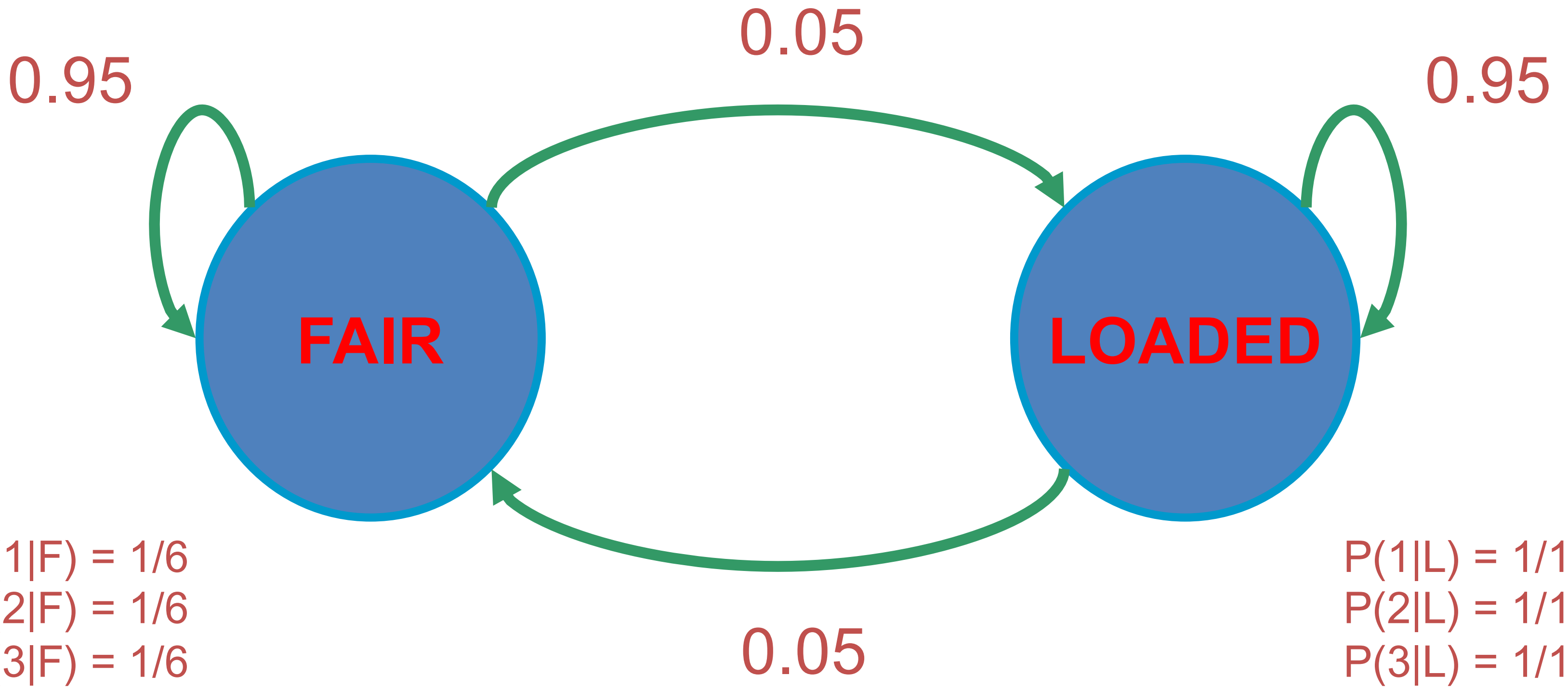
Biologists need algorithms to identify protein parts

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 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 on right describes a 7 helix membrane protein forming a sort of a cylinder (porus) across the cell membrane





The dishonest casino model



$P(1|F) = 1/6$
$P(2|F) = 1/6$
$P(3|F) = 1/6$
$P(4|F) = 1/6$
$P(5|F) = 1/6$
$P(6|F) = 1/6$

$P(1|L) = 1/10$
$P(2|L) = 1/10$
$P(3|L) = 1/10$
$P(4|L) = 1/10$
$P(5|L) = 1/10$
$P(6|L) = 1/2$

HMM

Definition: A hidden Markov model (HMM)

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

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

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

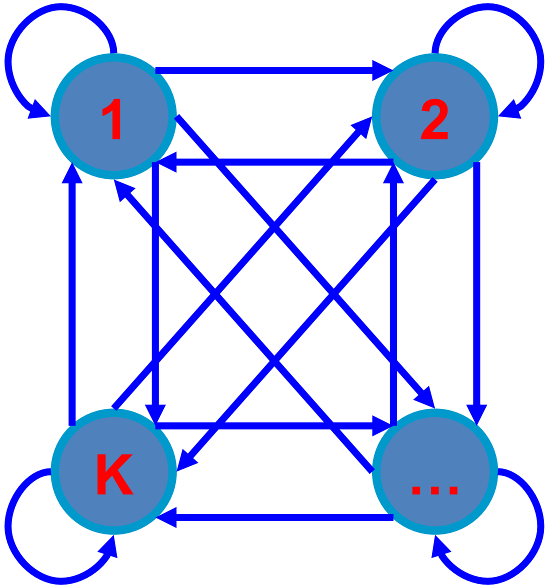
- Start probabilities a_{0i}

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

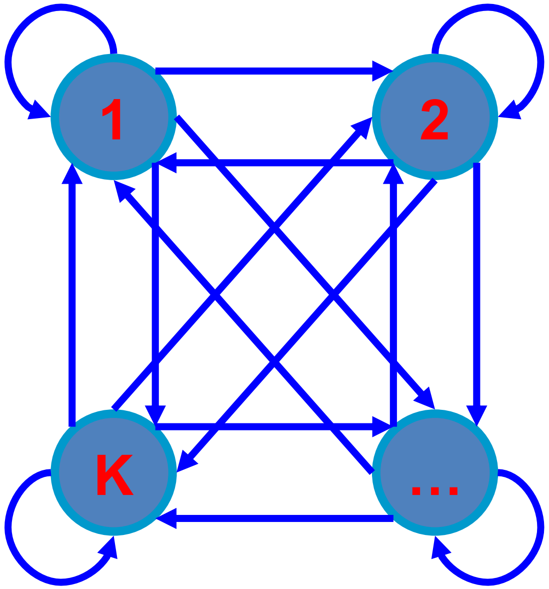
- Emission probabilities within each state

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

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



A Hidden Markov Model is memory-less



At each time step t,
the only thing that affects future states
is the current state π_t

$P(\pi_{t+1} = k \mid \text{“whatever happened so far”}) =$

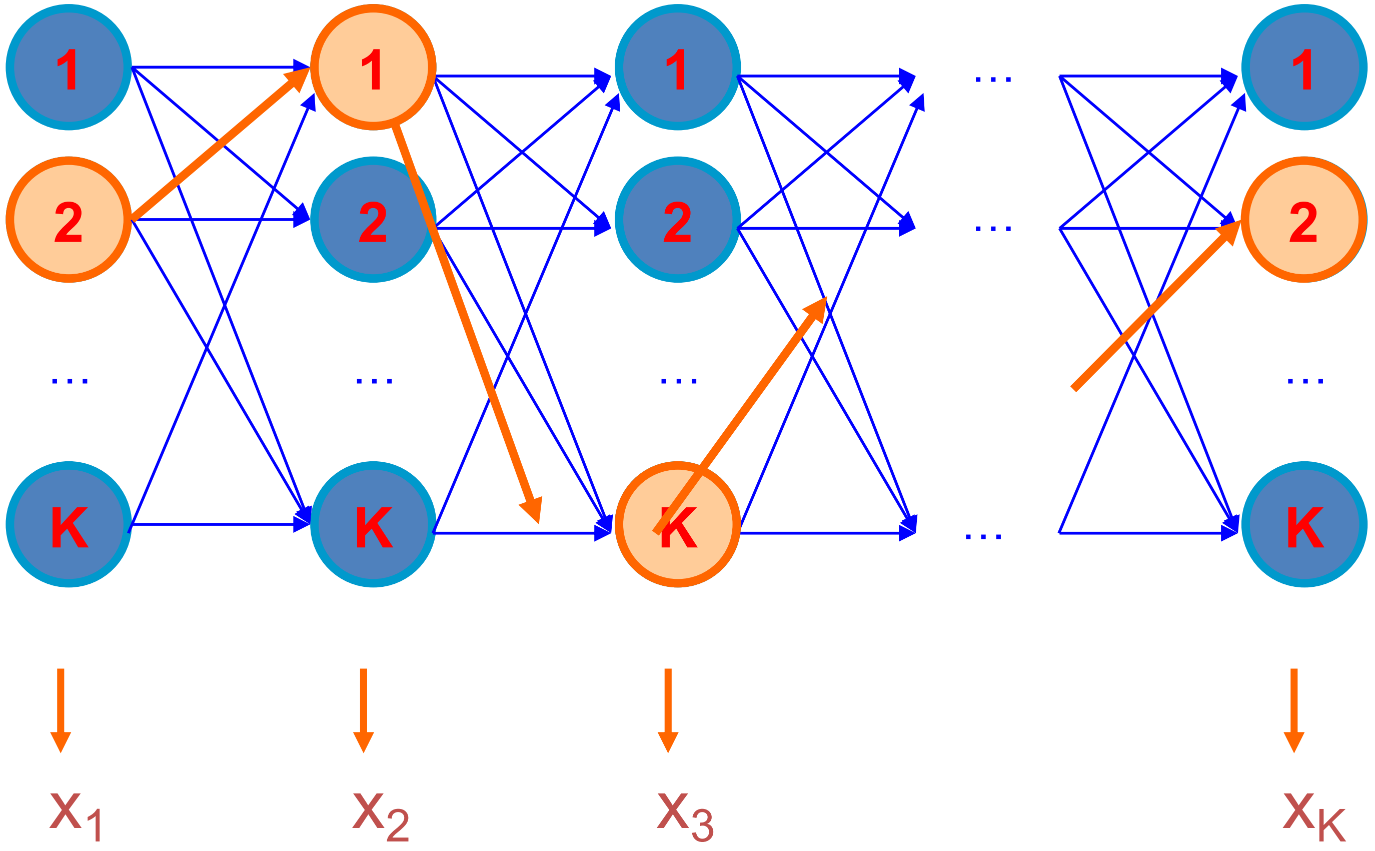
$P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) =$

$P(\pi_{t+1} = k \mid \pi_t)$

A parse of a sequence

Given a sequence $x = x_1……x_N$,

A <u>parse</u> of x is a sequence of states $\pi = \pi_1, ……, \pi_N$



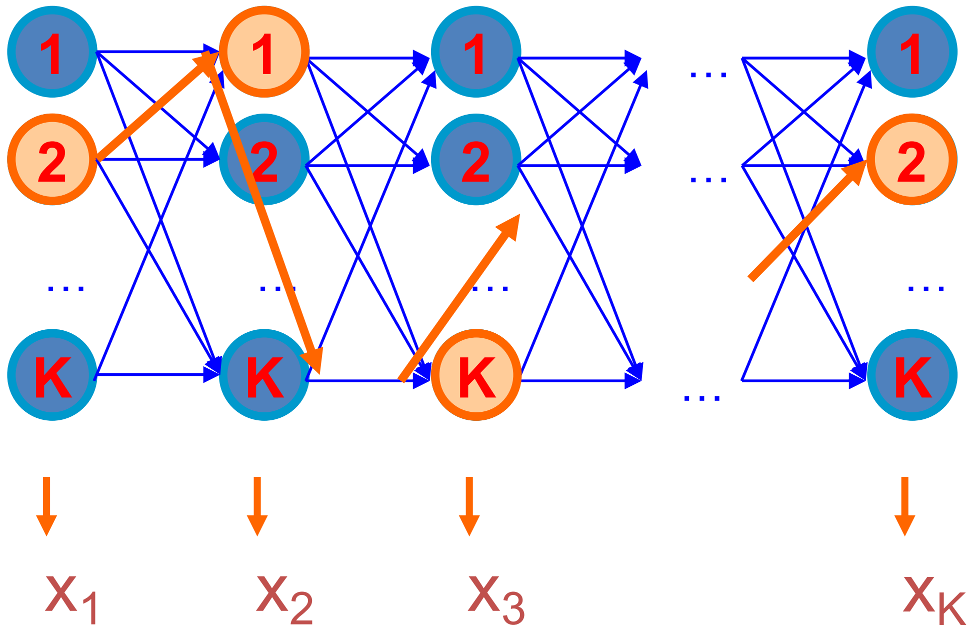
Likelihood of a parse

Given a sequence $x = x_1 \ldots\ldots x_N$

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

To find how likely is the parse:

(given our HMM)



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

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

$P(x_1, \pi_1) =$

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

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

Example: the dishonest casino

Let the sequence of rolls be:

x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4

Then, what is the likelihood of

π = Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair?

(say initial probs a_{0Fair} = ½, $a_{oLoaded}$ = ½)

½ × P(1 | Fair) P(Fair | Fair) P(2 | Fair) P(Fair | Fair) … P(4 | Fair) =

$\frac{1}{2} \times (1/6)^{10} \times (0.95)^{9} = .00000000521158647211 = 0.5 \times 10^{-9}$

Example: the dishonest casino

So, the likelihood the die is fair in all this run

is just 0.521 $\times$ 10^{-9}

OK, but what is the likelihood of

= Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?

½ $\times$ P(1 | Loaded) P(Loaded, Loaded) … P(4 | Loaded) =

½ $\times$ $(1/10)^8 \times (1/2)^2 (0.95)^9$ = .00000000078781176215 = 7.9 $\times$ 10^{-10}

Therefore, it is after all 6.59 times more likely that the die is fair all the way, than that it is loaded all the way.

Example: the dishonest casino

Let the sequence of rolls be:

x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6

Now, what is the likelihood π = F, F, …, F?

$\frac{1}{2} \times (1/6)^{10} \times (0.95)^{9} = 0.5 \times 10^{-9}$, same as before

What is the likelihood

π = L, L, …, L?

$\frac{1}{2} \times (1/10)^{4} \times (1/2)^{6} (0.95)^{9} = .00000049238235134735 = 0.5 \times 10^{-7}$

So, it is 100 times more likely the die is loaded

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 θ = (e_i(.), a_{ij}) that maximize P[x | θ]

Let's not be confused by notation

$P[\, x \mid M \,]$: The probability that sequence x was generated by the model

The model is: architecture (#states, etc)

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

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

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

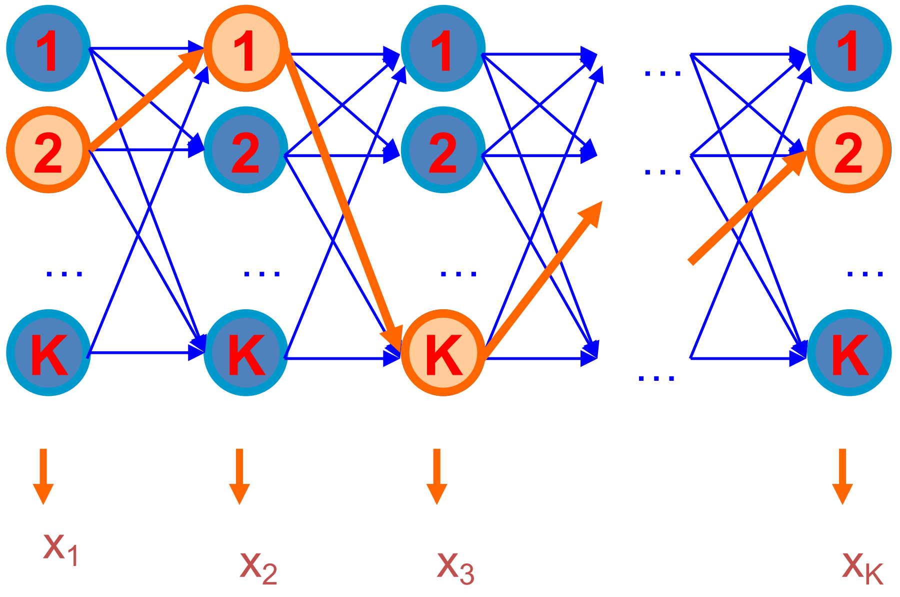
In the LEARNING problem we always write $P[\, x \mid \theta \,]$ to emphasize that we are seeking the θ that maximizes $P[\, x \mid \theta \,]$

Decoding

GIVEN x = $x_1x_2......x_N$

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

$\pi^* = \text{argmax}_\pi$ P[x, π]



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,

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

What is $V_k(i+1)$?

From definition,

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

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

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

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

The Viterbi Algorithm

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

Andrew Viterbi

Initialization:

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

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

Iteration:

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

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

Termination:

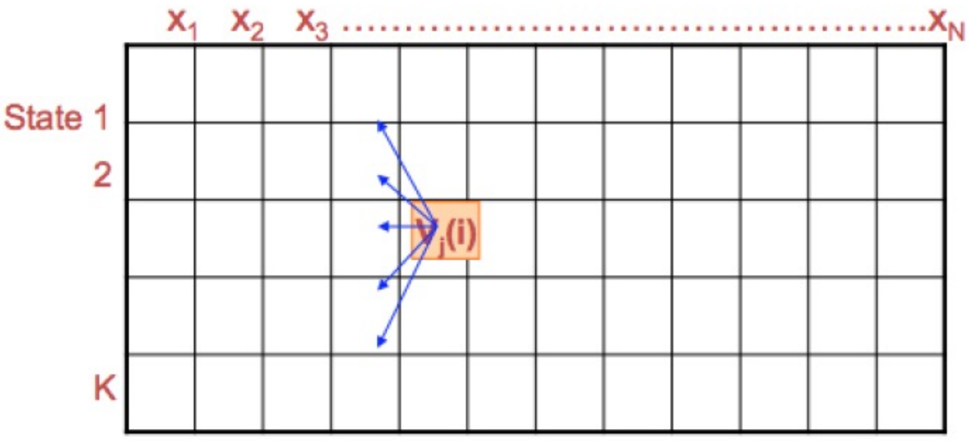
$P(x, \pi^*) = \max_k V_k(N)$

Traceback:

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

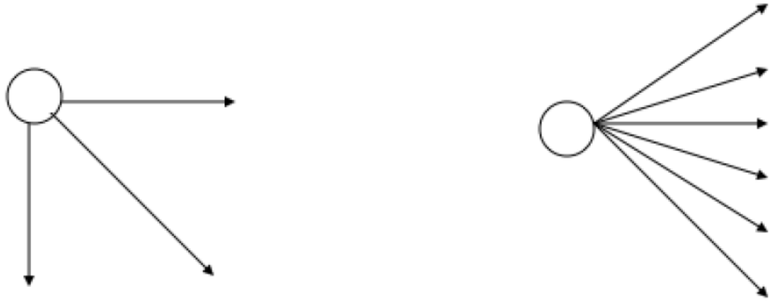
$\pi_{i-1}^* = \text{Ptr}_{\pi i}(i)$

The Viterbi Algorithm: complexity



left: Similar to “aligning” a set of states to a sequence,

Time: $O(K^2N)$; **Space:** $O(KN)$; bottom right : comparison of valid directions in the alignment and decoding problem.



Viterbi Algorithm – a practical detail

Underflows are a significant problem

$P[\, x_1, \ldots, x_i, \pi_1, \ldots, \pi_i \,] = a_{0\pi 1}\, a_{\pi 1 \pi 2} \ldots\ldots a_{\pi i}\, e_{\pi 1}(x_1) \ldots\ldots e_{\pi i}(x_i)$

These numbers become extremely small – underflow

Solution: Take the logs of all values

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

Examples

Let x be a sequence with a portion of ~ 1/6 6’s, followed by a portion of ~ ½ 6’s…

x = 123456123456…12345 6626364656…1626364656

Then, it is not hard to show that optimal parse is (exercise):

FFF…………………...F LLL………………………...L

6 nucleotides “123456” parsed as F, contribute $.95^6\times(1/6)^6 = 1.6\times10^{-5}$

parsed as L, contribute $.95^6\times(1/2)^1\times(1/10)^5 = 0.4\times10^{-5}$

“162636” parsed as F, contribute $.95^6\times(1/6)^6 = 1.6\times10^{-5}$

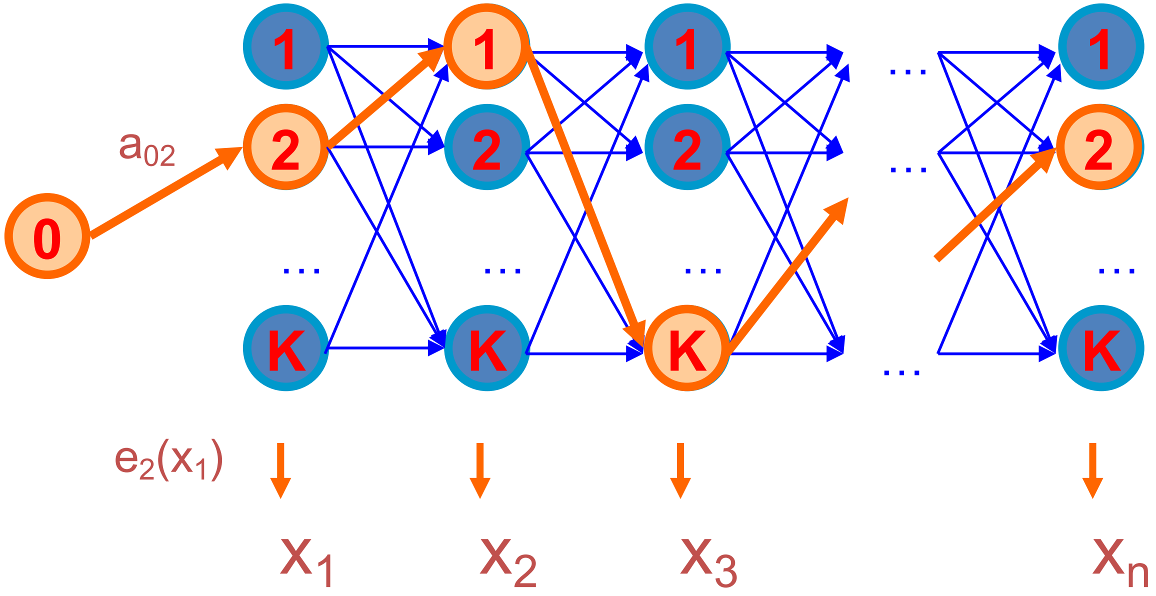
parsed as L, contribute $.95^6\times(1/2)^3\times(1/10)^3 = 9.0\times10^{-5}$

Generating a sequence by the model

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

Start at state π_1 according to prob $a_{0\pi_1}$

1. Emit letter x_1 according to prob $e_{\pi_1}(x_1)$
2. Go to state π_2 according to prob $a_{\pi_1\pi_2}$
3. … until emitting x_n



A couple of questions

Given a sequence x,

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

Example: the dishonest casino

Say x = 12341623162616364616234161221341

Most likely path: π = FF……F

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

Evaluation

We will develop algorithms that allow us to compute:

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

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

$P(\pi_l = k \mid x)$ 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:

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

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

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

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

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 $= a_{0k}$)

Relation between Forward and Viterbi

VITERBI

Initialization:

$V_0(0) = 1$

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

Iteration:

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

Termination:

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

FORWARD

Initialization:

$f_0(0) = 1$

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

Iteration:

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

Termination:

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

Motivation for the Backward Algorithm

We want to compute

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

the probability distribution on the ith position, given x

We start by computing

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

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:

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

Termination:

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

Computational Complexity

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

Time: $O(K^2N)$

Space: $O(KN)$

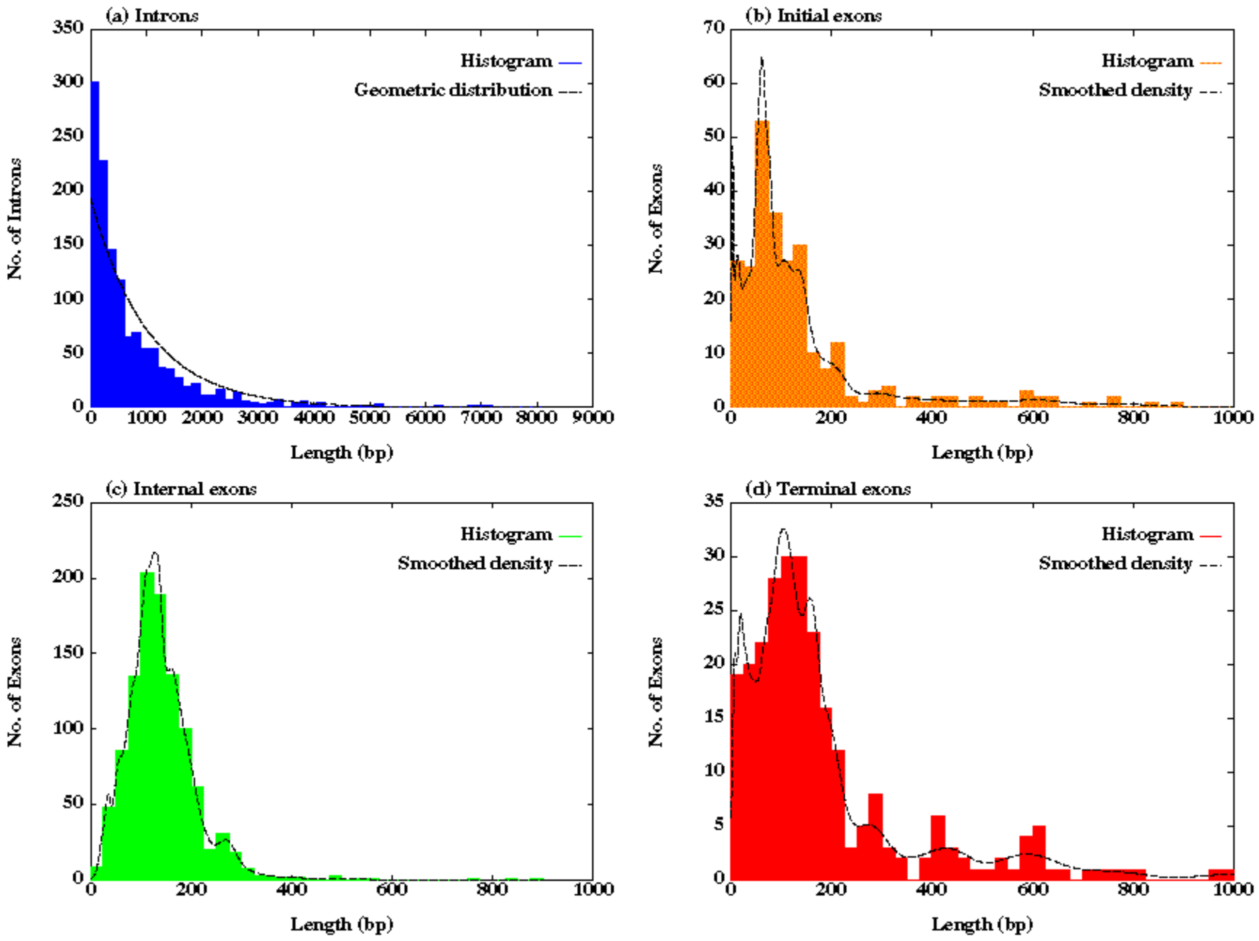
Useful implementation technique to avoid underflows

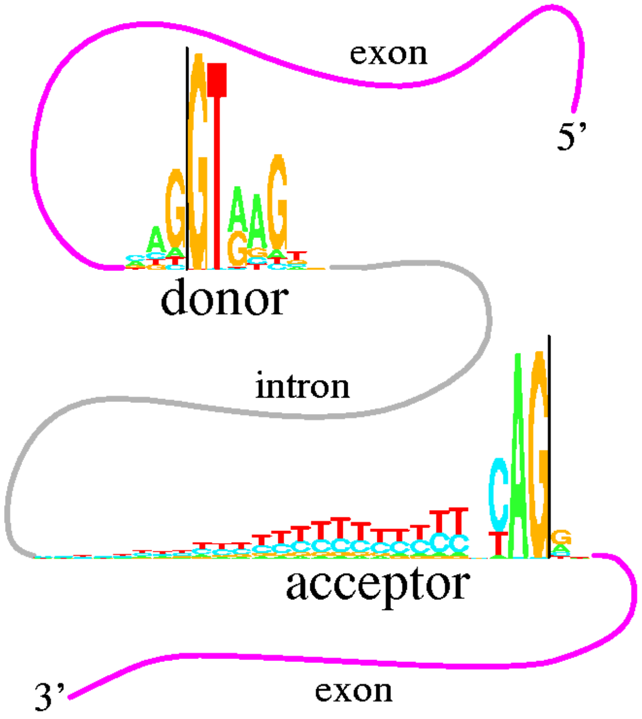
Viterbi: sum of logs

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

Application: GenScan

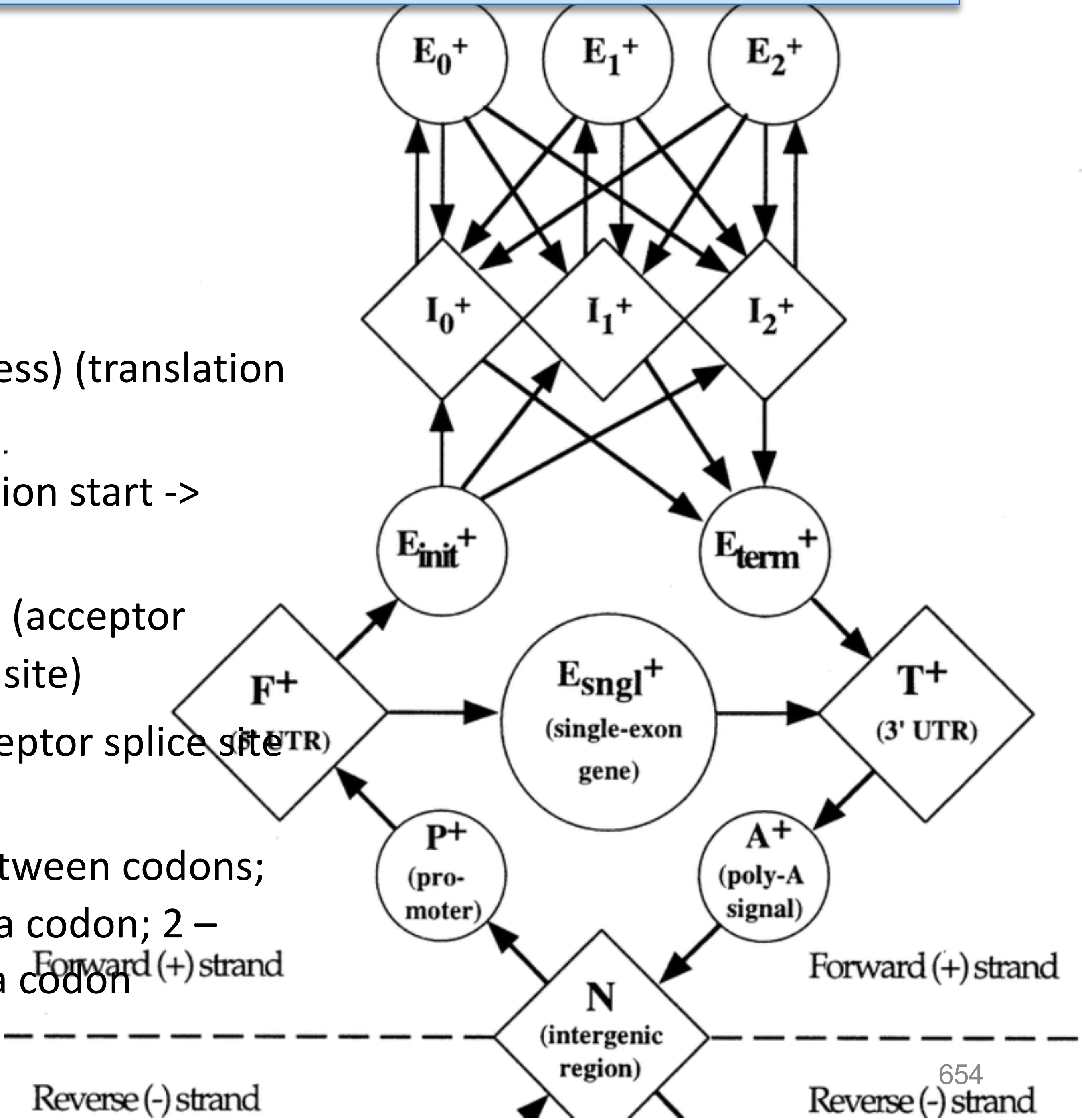
Length distributions of human introns and initial, internal and terminal exons



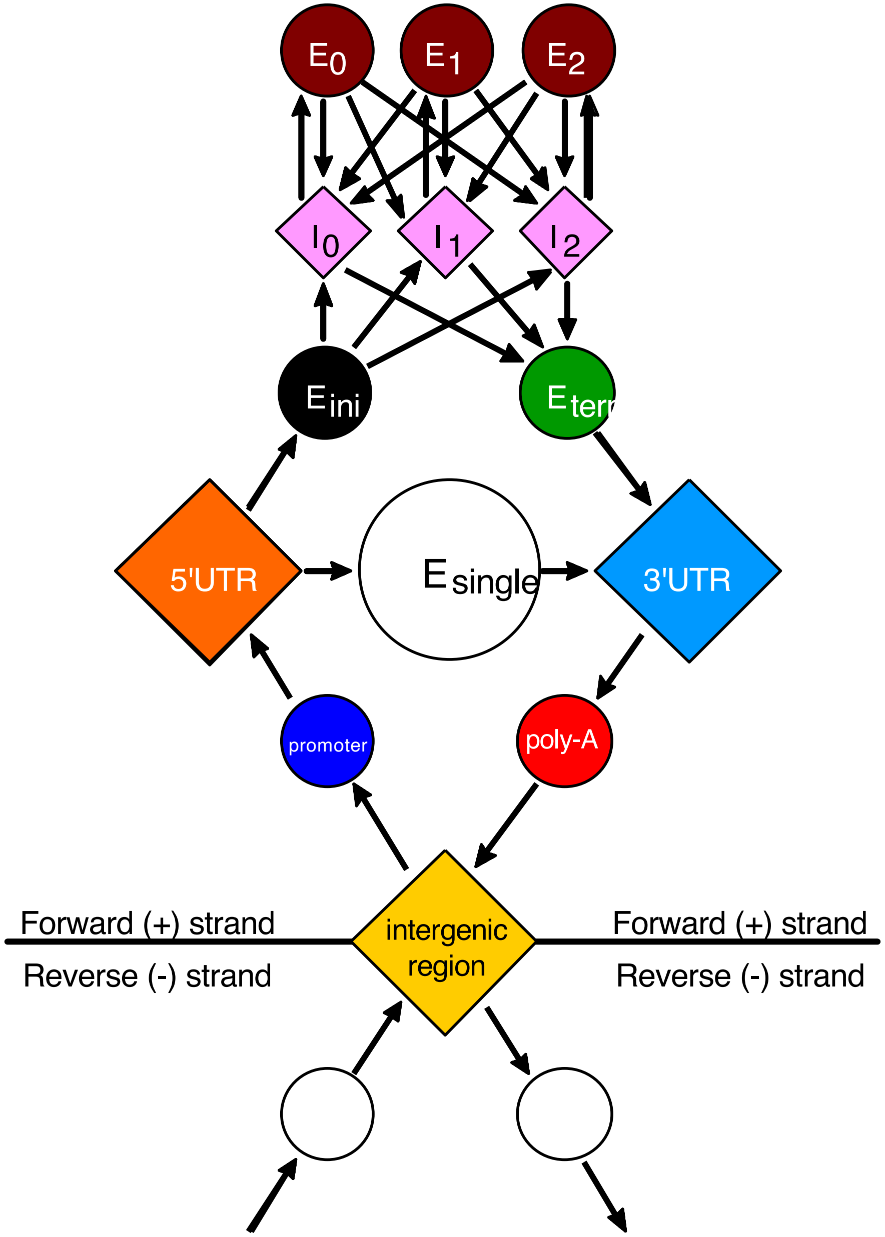


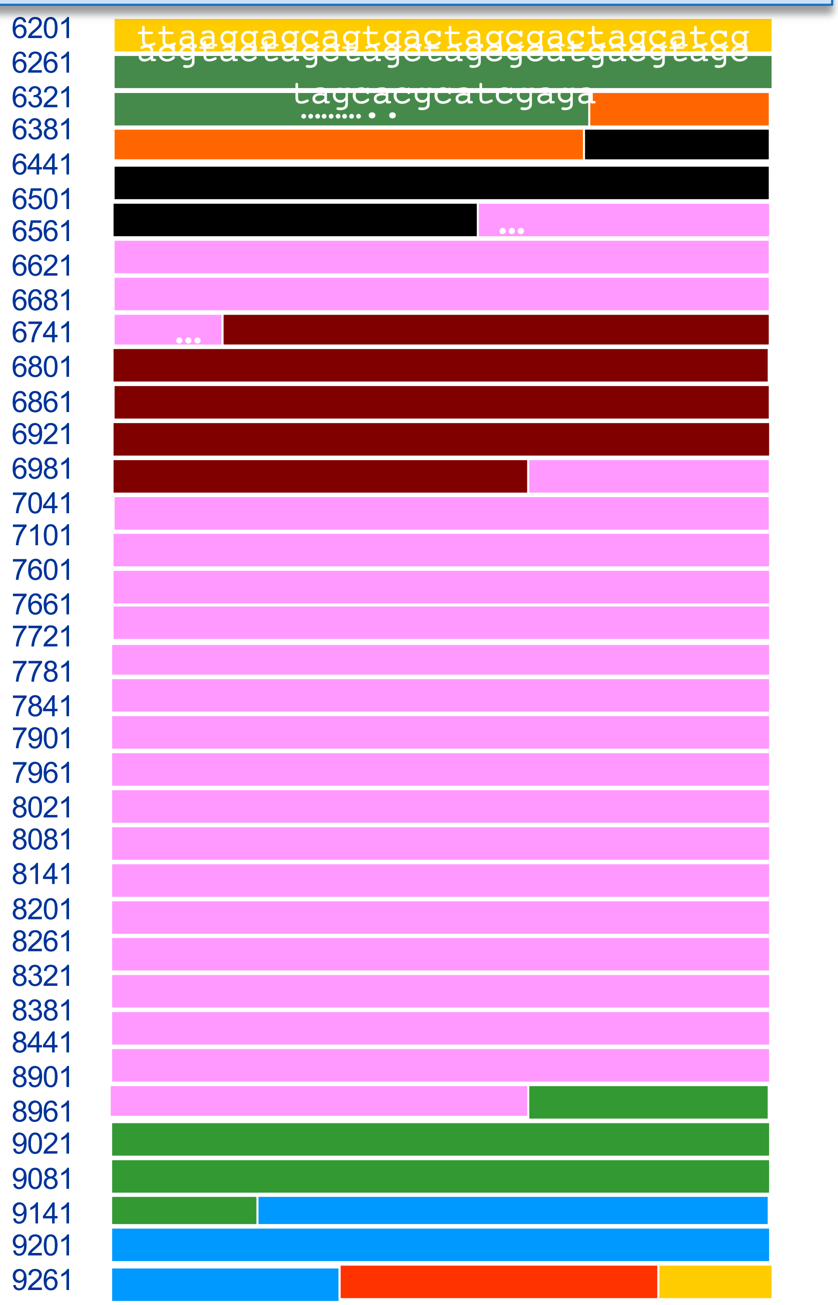
(continue) GenScan

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



GenScan


E0
E1
E2
I0
I1
I2
Eini
Eterm
5'UTR
Esingle
3'UTR
promoter
poly-A
intergenic region
Forward (+) strand
Reverse (-) strand
Forward (+) strand
Reverse (-) strand

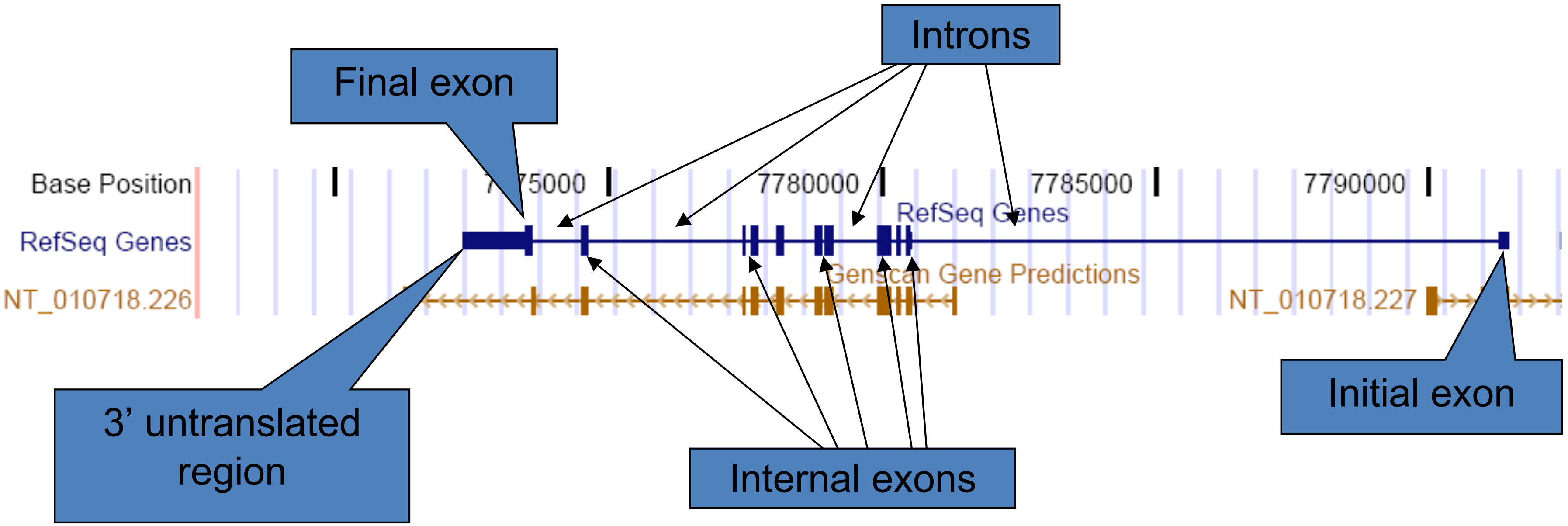

6201
6261
6321
6381
6441
6501
6561
6621
6681
6741
6801
6861
6921
6981
7041
7101
7601
7661
7721
7781
7841
7901
7961
8021
8081
8141
8201
8261
8321
8381
8441
8901
8961
9021
9081
9141
9201
9261
ttaaggagcagtgactagcgactagcatcg
acgtactagctagctagcgcatgacgtagc
tagcacgcatcgaga
....
...
...

Genscan model

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

Performance
> 80% correct exon predictions, and > 90% correct coding/non coding predictions by bp.
BUT - the ability to predict the whole gene correctly is much lower

Example result: exons, introns prediction



Introns

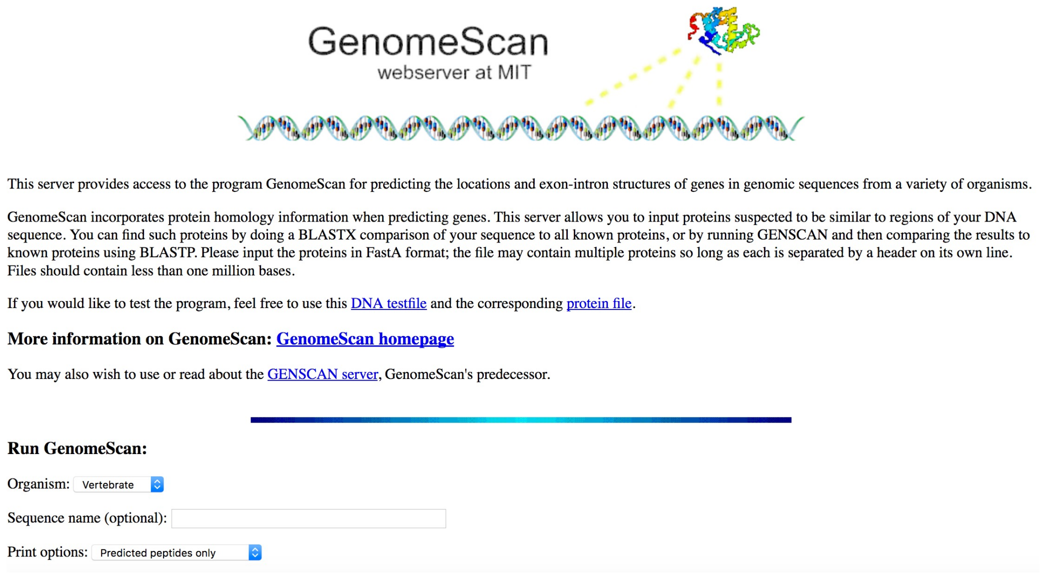
Final exon

3’ untranslated region

Internal exons

Initial exon

Human p53 tumor suppressor gene -chromosome 17



This server provides access to the program GenomeScan for predicting the locations and exon-intron structures of genes in genomic sequences from a variety of organisms.

GenomeScan incorporates protein homology information when predicting genes. This server allows you to input proteins suspected to be similar to regions of your DNA sequence. You can find such proteins by doing a BLASTX comparison of your sequence to all known proteins, or by running GENSCAN and then comparing the results to known proteins using BLASTP. Please input the proteins in FastA format; the file may contain multiple proteins so long as each is separated by a header on its own line. Files should contain less than one million bases.

If you would like to test the program, feel free to use this DNA testfile and the corresponding protein file.

More information on GenomeScan: GenomeScan homepage

You may also wish to use or read about the GENSCAN server, GenomeScan's predecessor.

Run GenomeScan:

Organism: Vertebrate

Sequence name (optional):

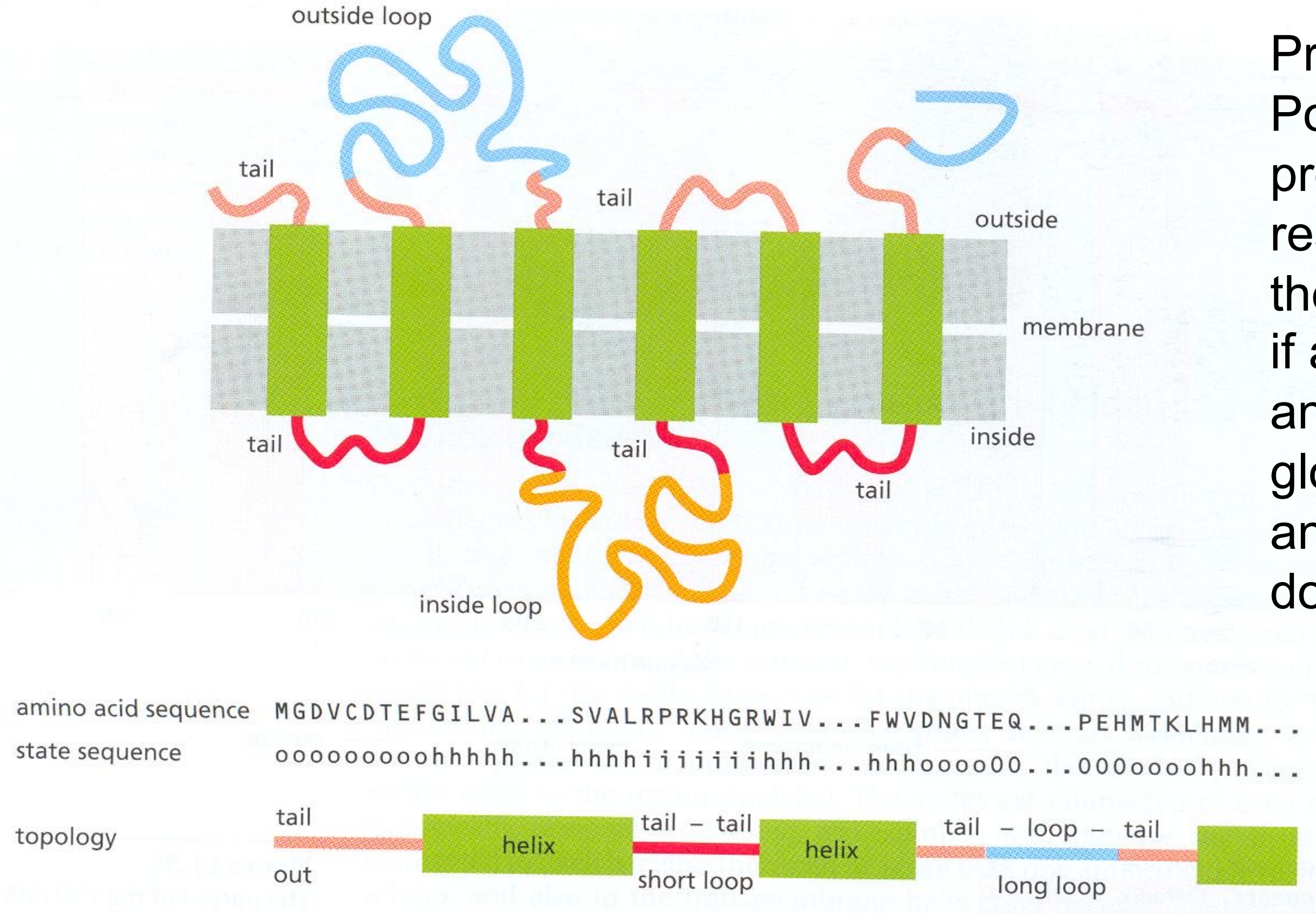
Print options: Predicted peptides only

TMHMM: Prediction of transmembrane topology of protein sequence

Model consists of submodels for:

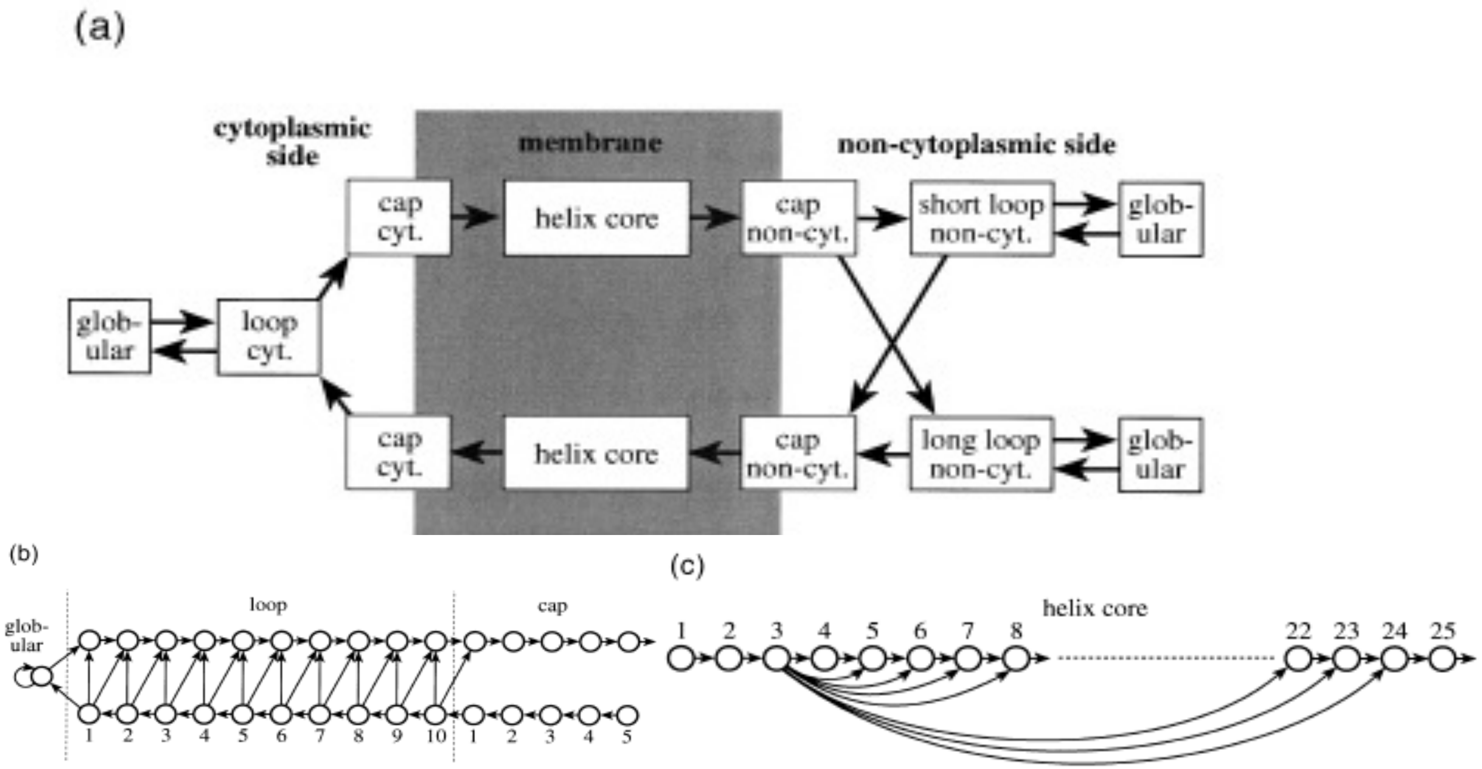
- helix core and cap regions (cytoplasmic and extracellular)
- cytoplasmic and extracellular loop regions
- globular domain regions

Trained form 160 proteins with experimentally determined transmembrane



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

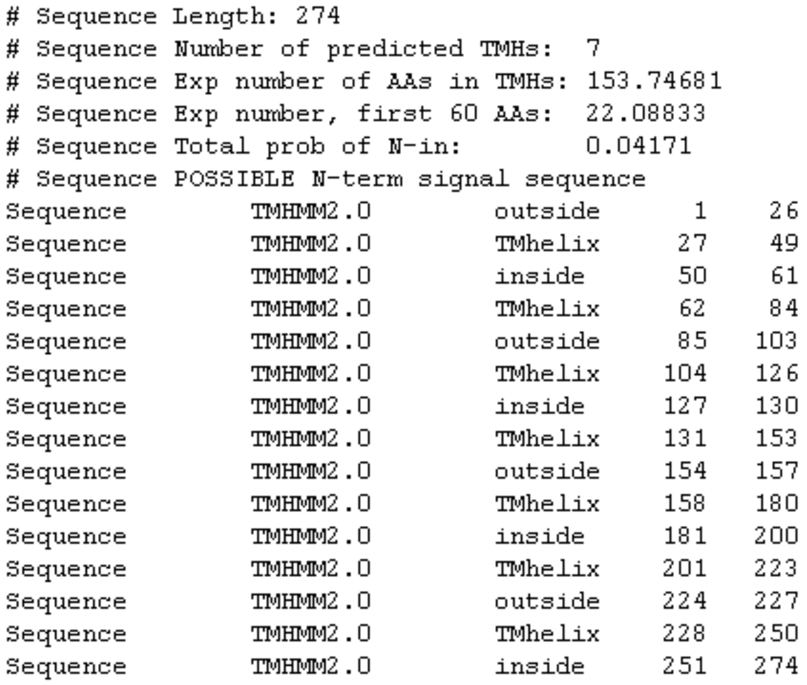
Model architecture of TMHMM



TMHMM: uses cyclic model with 7 states for

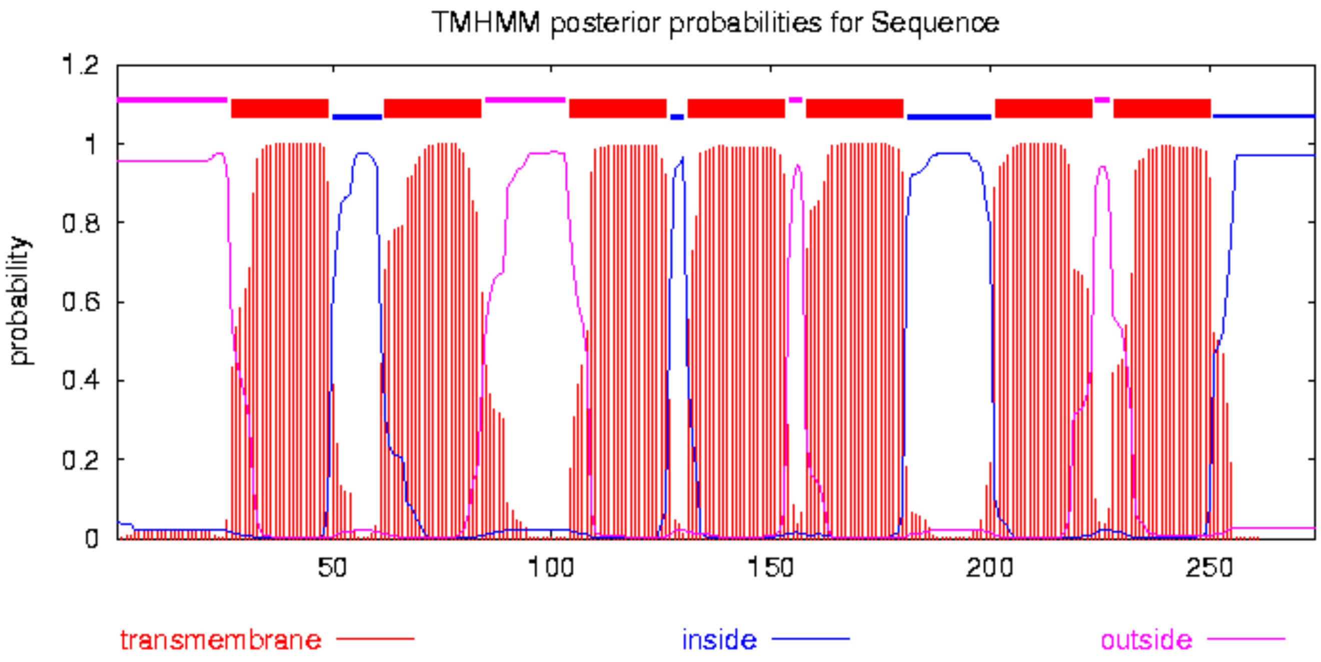
- TM helix core
- TM helix caps on the N- and C-terminal side
- non-membrane region on the cytoplasmic side
- 2 non-membrane regions on the non-cytoplasmic side (for short and long loops to account for different membrane insertion mechanism)
- a globular domain state in the middle of each non-membrane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

Example result: TMHMM-Output

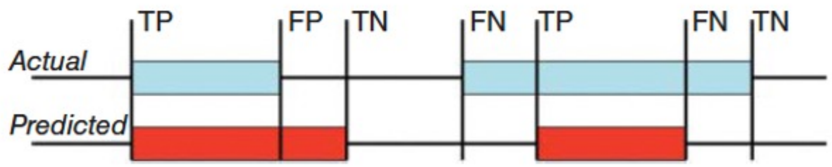


```
# Sequence Length: 274
# Sequence Number of predicted TMHs:  7
# Sequence Exp number of AAs in TMHs: 153.74681
# Sequence Exp number, first 60 AAs:  22.08833
# Sequence Total prob of N-in:        0.04171
# Sequence POSSIBLE N-term signal sequence
Sequence        TMHMM2.0        outside      1   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

http://www.cbs.dtu.dk/services/TMHMM-2.0/

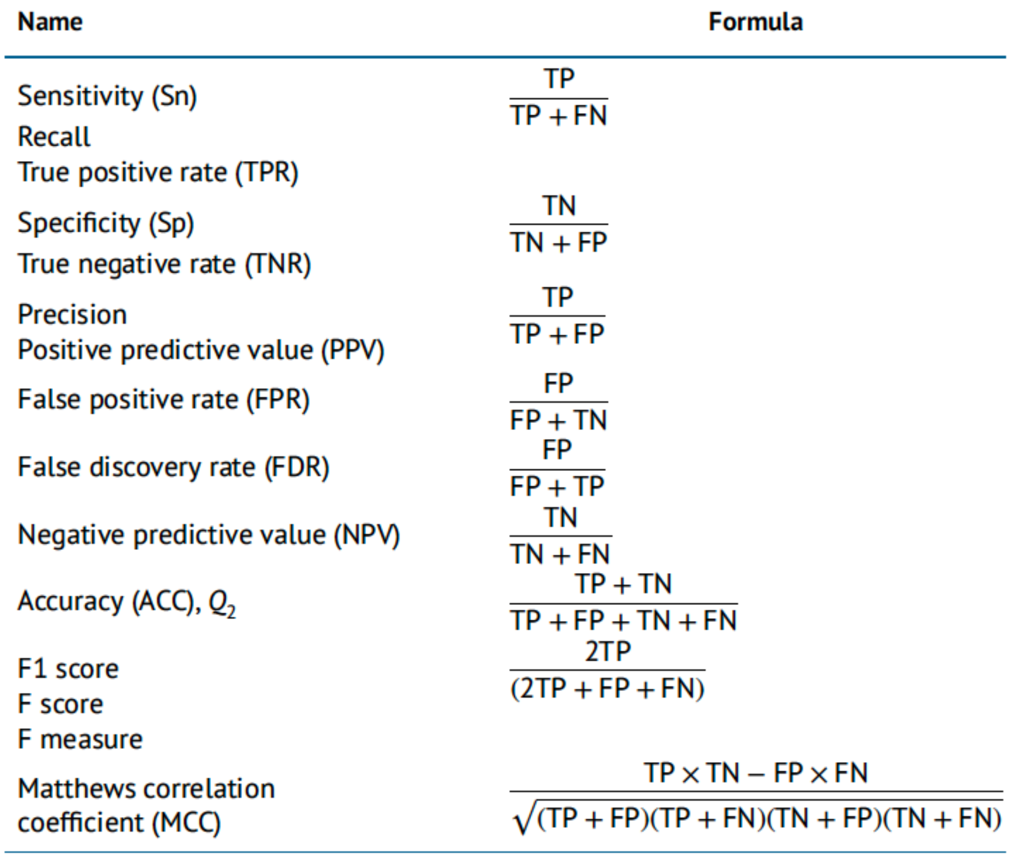


Validation for exons, introns, genes, protein parts etc



Sensitivity $Sn = TP/(TP + FN)$

Specificity $Sp = TN/(TN + FP)$



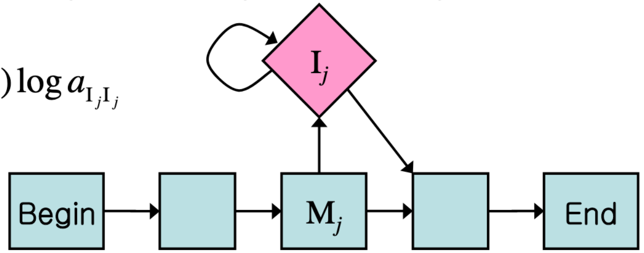
Name	Formula
Sensitivity (Sn) Recall True positive rate (TPR)	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$
Specificity (Sp) True negative rate (TNR)	$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$
Precision Positive predictive value (PPV)	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$
False positive rate (FPR)	$\frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$
False discovery rate (FDR)	$\frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TP}}$
Negative predictive value (NPV)	$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}}$
Accuracy (ACC), Q_2	$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{TN}+\mathrm{FN}}$
F1 score F score F measure	$\frac{2\mathrm{TP}}{(2\mathrm{TP}+\mathrm{FP}+\mathrm{FN})}$
Matthews correlation coefficient (MCC)	$\frac{\mathrm{TP}\times\mathrm{TN}-\mathrm{FP}\times\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$

Sensitivity (Sn, recall, or TPR) measures the proportion of actual positives that are correctly identified as such, while specificity (Sp or TNR) measures the proportion of actual negatives that are correctly identified as such. Precision (PPV) is the proportion of positive results that are true positive results, while NPV is the proportion of negative results that are true negative results. FDR is the binary (not the multiple testing) measure of false positives divided by all positive predictions. Accuracy or ACC (for binary classification) is defined as the number of correct predictions made divided by the total number of predictions made. ACC is one of the best ways of assessing binary test or predictor accuracy. The F1 score is another measure of test accuracy and is defined as the harmonic average of precision (PPV) and recall (Sn). MCC is a popular measure of test or predictor accuracy. It is essentially a chi-squared statistic for a standard 2 × 2 contingency table. In effect, MCC is the correlation coefficient between the observed and predicted binary classifications.

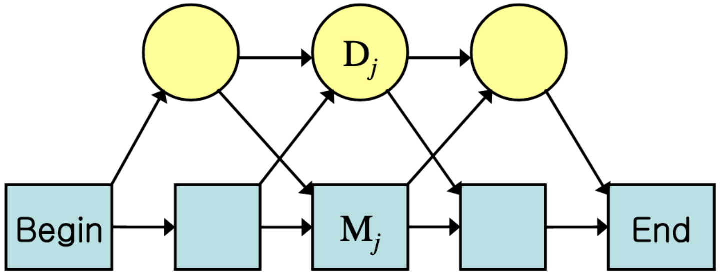
Application of HMMs: Model representing the consensus for the alignment of sequence from the same family of sequences, not the sequence of any particular member

- Insert states $e_{\mathrm{I}_i}(a)$
  - Emission prob.
    - Usually back ground distribution q_a.
  - Transition prob.
    - M_i to I_i, I_i to itself, I_i to M_{i+1}
  - Log-odds score for a gap of length k (no log-odds from emission)

$$\log a_{\mathrm{M}_j\mathrm{I}_j} + \log a_{\mathrm{I}_j\mathrm{M}_{j+1}} + (k-1)\log a_{\mathrm{I}_j\mathrm{I}_j}$$



- Delete states
  - No emission prob.
  - Cost of a deletion
    - M→D, D→D, D→M
    - Each D→D might be different



- Parameters
  - the probabilities values : trivial if many of independent alignment sequences are given.

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \qquad e_k(a) = \frac{E_k(a)}{\sum_{a'} E_k(a')}$$

profile HMM parameterization
– Aim: making the higher probability for sequences from the family

(continue from previous slide) profile HMMs:

Detecting potential sequences in a family

Matching a sequence to the profile HMMs

- Viterbi algorithm or forward algorithm

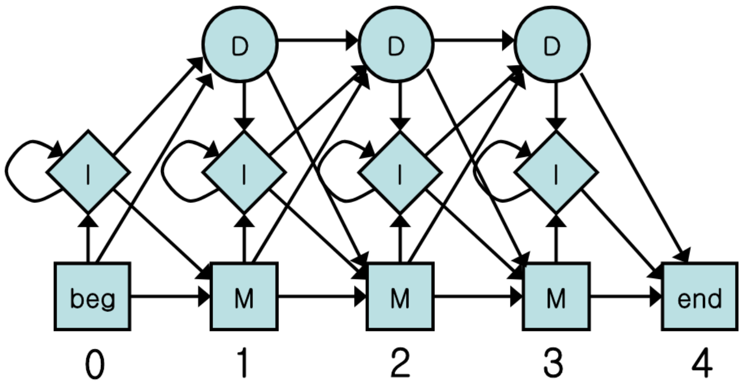
Comparing the resulting probability with random model

$$P(x \mid R) = \prod_i q_{x_i}$$

(a) Multiple alignment:

```
        x x . . . x
bat     A G - - - C
rat     A - A G - C
cat     A G - A A -
gnat    - - A A A C
goat    A G - - - C
        1 2 . . . 3
```

(b) Profile-HMM architecture:

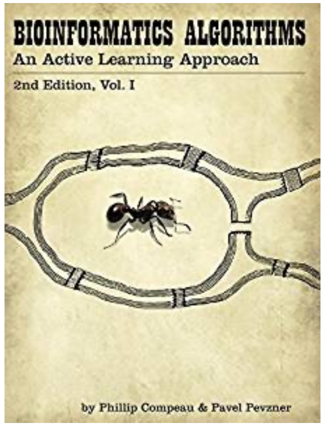


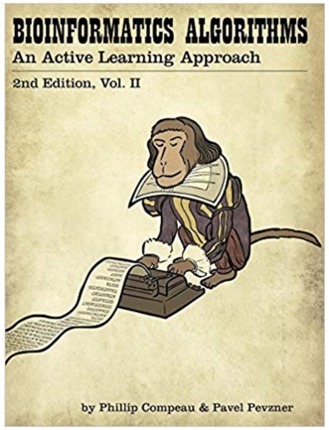
(c) Observed emission/transition counts

		0	1	2	3
match emissions	A	-	4	0	0
	C	-	0	0	4
	G	-	0	3	0
	T	-	0	0	0
insert emissions	A	0	0	6	0
	C	0	0	0	0
	G	0	0	1	0
	T	0	0	0	0
state transitions	M-M	4	3	2	4
	M-D	1	1	0	0
	M-I	0	0	1	0
	I-M	0	0	2	0
	I-D	0	0	1	0
	I-I	0	0	4	0
	D-M	-	0	0	1
	D-D	-	1	0	0
	D-I	-	0	2	0

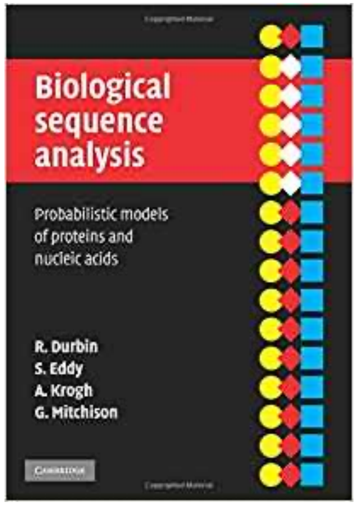
S R Eddy Profile hidden Markov models Bioinformatics 1998;14:755-63; Lemoine et al. COVID-Align: Accurate online alignment of hCoV-19 genomes using a profile HMM, Bioinformatics 2020

Reference for this section


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. I
by Phillip Compeau & Pavel Pevzner


BIOINFORMATICS ALGORITHMS
An Active Learning Approach
2nd Edition, Vol. II
by Phillip Compeau & Pavel Pevzner

- Chapter 10 Vol 2 (pag 185-225)


Biological sequence analysis
Probabilistic models of proteins and nucleic acids
R. Durbin
S. Eddy
A. Krogh
G. Mitchison
CAMBRIDGE

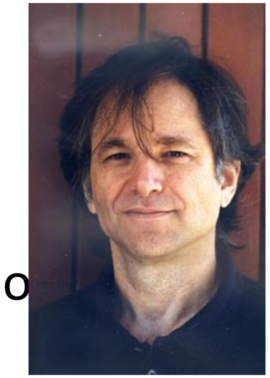
- Chapter 3

Section 8

Algorithms for DNA computing and storage

- Algorithm: Adleman DNA Computing
- Algorithm: Random access in large-scale DNA data storage

DNA for computing



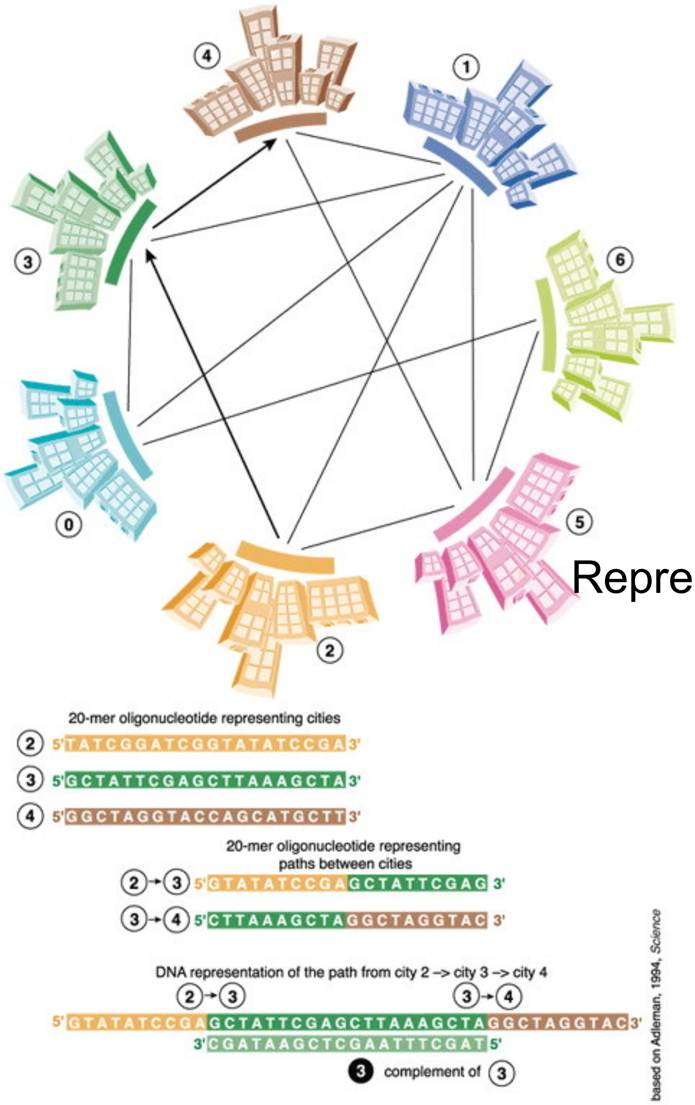
Adleman's DNA computation approach (1994) solved a Hamilton problem o seven cities.   He used DNA techniques to assemble itineraries at random; Select itineraries from initial city to final city. The correct number of cities must be visited. No city can be left out.

Each city is represented by a unique sequence of bases. Connections between two cities are created from a combination of the complement of the first half of the sequence of one city, and the complement of the second half of the sequence of a connected city. In this way DNA representing the trip will be created with one strand representing a sequence of cities and the complementing strand representing a series of connections.

The next step is filtering out trips that start and end in the correct cities, then filtering trips with the correct number of cities, and finally filtering out trips that contain each city only once.

DNA for computing

Represent Each City By A DNA Strand of 20 Bases


4
1
3
6
0
5
2
20-mer oligonucleotide representing cities
2 5'TATCGGATCGGTATATCCGA3'
3 5'GCTATTCGAGCTTAAAGCTA3'
4 5'GGCTAGGTACCAGCATGCTT3'
20-mer oligonucleotide representing paths between cities
2→3 5'GTATATCCGAGCTATTCGAG 3'
3→4 5'CTTAAAGCTAGGCTAGGTAC 3'
DNA representation of the path from city 2 –> city 3 –> city 4
2→3
3→4
5'GTATATCCGAGCTATTCGAGCTTAAAGCTAGGCTAGGTAC3'
3'CGATAAGCTCGAATTTCGAT5'
3 complement of 3

based on Adleman, 1994, *Science*

City1	ATGCTCAGCTACTATAGCGA
City2	TGCGATGTACTAGCATATAT
City3	GCATATGGTACACTGTACAA
City4	TTATTAGCGTGCGGCCTATG
City5	CCGCGATAGTCTAGATTTCC

Etc.

Represent Each Air Route By Mixed Complementary Strands

City 1->2	TGATATCGCTACGCTACATG
City 2->3	ATCGTATATACGTATACCAT
City 3->4	GTGACATGTTAATAATCGCA
City 4->5	CGCCGGATACGGCGCTATCA
City 5->6	GATCTAAAGGTATGCATACG

Etc.

L. Adelman, *Scientific American*, pp. 54-61 (Aug 1998);

Hamiltonian problem: list of steps

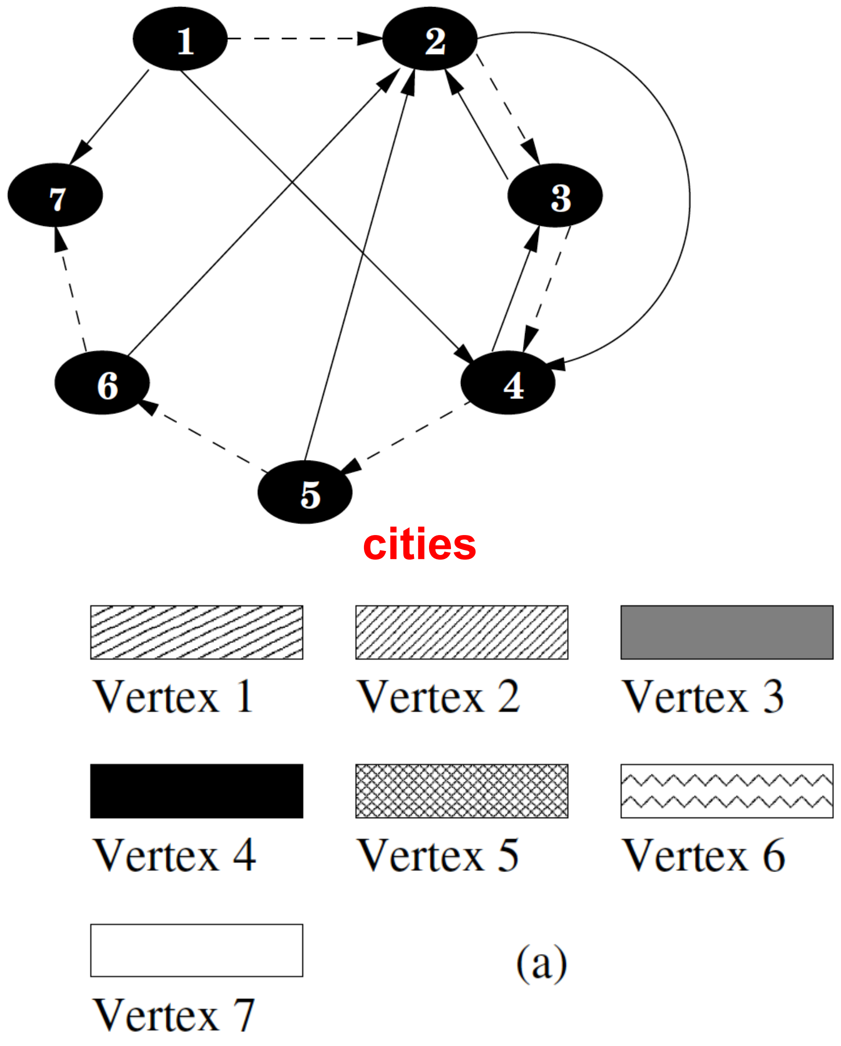
The challenge is finding a route between various cities, passing through each only once.

Adleman first generated all the possible itineraries and then selected the correct itinerary.

Specifically, the method based on Adleman's experiment would be as follows:

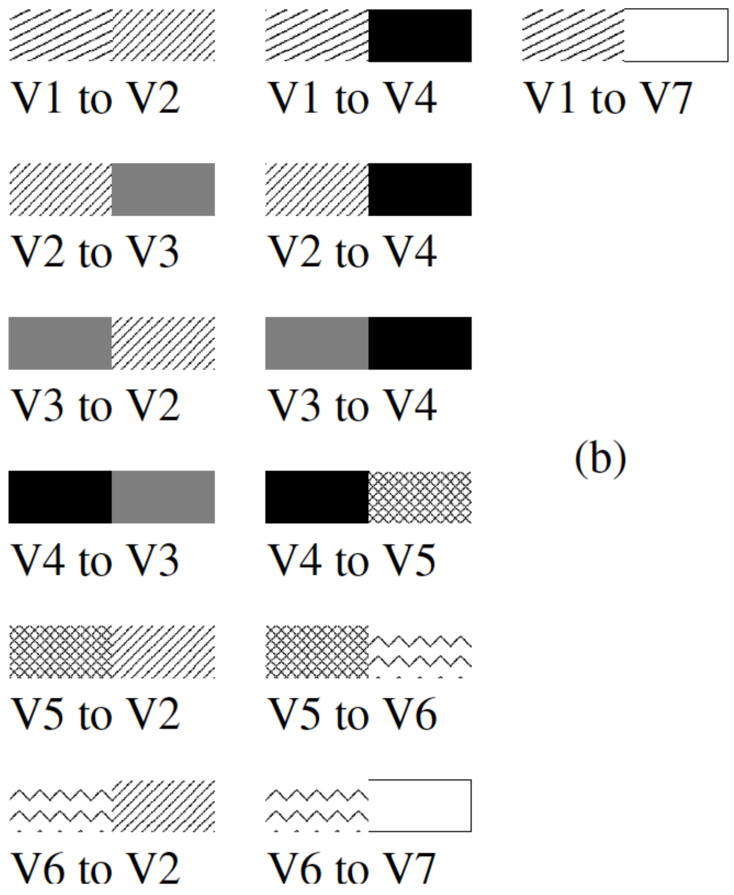
- 1 Generate all possible routes.
- 2 Select itineraries that start with the proper city and end with the final city.
- 3 Select itineraries with the correct number of cities.
- 4 Select itineraries that contain each city only once.
- All of the above steps can be accomplished with standard molecular biology techniques.

Step 1: Generate all possible routes



cities

routes



Step 1: Technique for Generating Routes Strategy

Encode city names in short DNA sequences. Encode itineraries by connecting the city sequences for which routes exist.

Synthesizing short single stranded DNA is now a routine process, so encoding the city strings is straightforward. Itineraries can then be produced from the city encodings by linking them together in proper order.

To accomplish this you can take advantage of the fact that DNA hybridizes (=binds) with its complimentary sequence (complementary strands of DNA bind each other).

For example, you can encode the routes between cities by encoding the complement of the second half (last n letters) of the departure city and the first half (first n letters) of the arrival city.

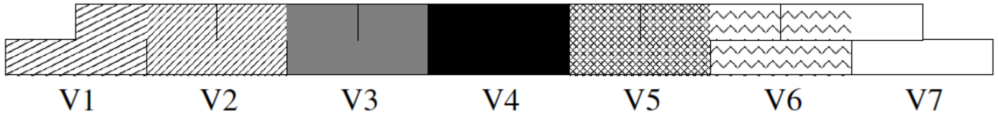
For example the route between Miami (CTACGG) and NY (ATGCCG) can be made by taking the second half of the coding for Miami (CGG) and the first half of the coding for NY (ATG). This gives CGGATG.

By taking the complement of this you get, GCCTAC, which not only uniquely represents the route from Miami to NY, but will connect the DNA representing Miami and NY by hybridizing itself to the second half of the code representing Miami (...CGG) and the first half of the code representing NY (ATG…).

Random itineraries can be made by mixing city encodings with the route encodings. Finally, the DNA strands can be connected together by an enzyme called ligase (ligases are enzymes, i.e. proteins connecting strings). What we are left with are strands of DNA representing itineraries with a random number of cities and random set of routes.

Step 2,3: Sort the DNA by length and select the DNA whose length corresponds to 7 cities

selection for length and initial/end points

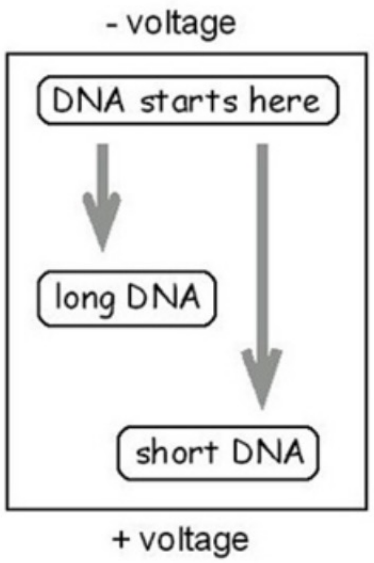


A test tube is now filled with DNA encoded itineraries that start with LA and end with NY, where the number of cities in between LA and NY varies.

We now want to select those itineraries that are seven cities long. To accomplish this we can use a technique called Gel Electrophoresis, which is a common procedure used to resolve the size of DNA.

Step 2,3: Sort the DNA by length and select the DNA whose length corresponds to 7 cities (tech details)

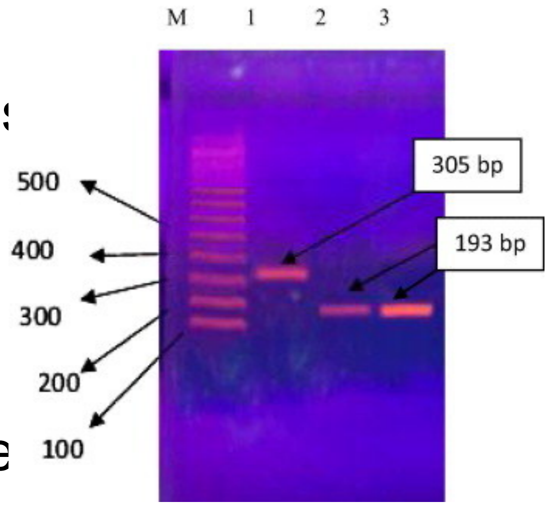
DNA is a negatively charged molecule, so if placed in an electric field it will be attracted to the positive potential. The basic principle behind Gel Electrophoresis is to force DNA through a gel matrix by using an electric field.



The gel is made up of a polymer that forms a meshwork of linked strands. The DNA now is forced to thread its way through the tiny spaces, which slows down the DNA at different rates depending on its length.

What we typically end up with after running a gel is a series of DNA bands, with each band corresponding to a certain length.

We can then simply cut out the band of interest to isolate DNA of a specific length. We know that each city is encode with a certain number of base pairs of DNA, knowing the length of the itinerary gives us the number of cities.



UV shows DNA position

Step 4: itineraries Selection: Start and End with Correct Cities (using PCR)

Strategy: Selectively copy and amplify only the section of the DNA that starts with LA and ends with NY by using the Polymerase Chain Reaction (PCR). See next slide.

After generating the routes, we now have a test tube full of various lengths of DNA that encode possible routes between cities.

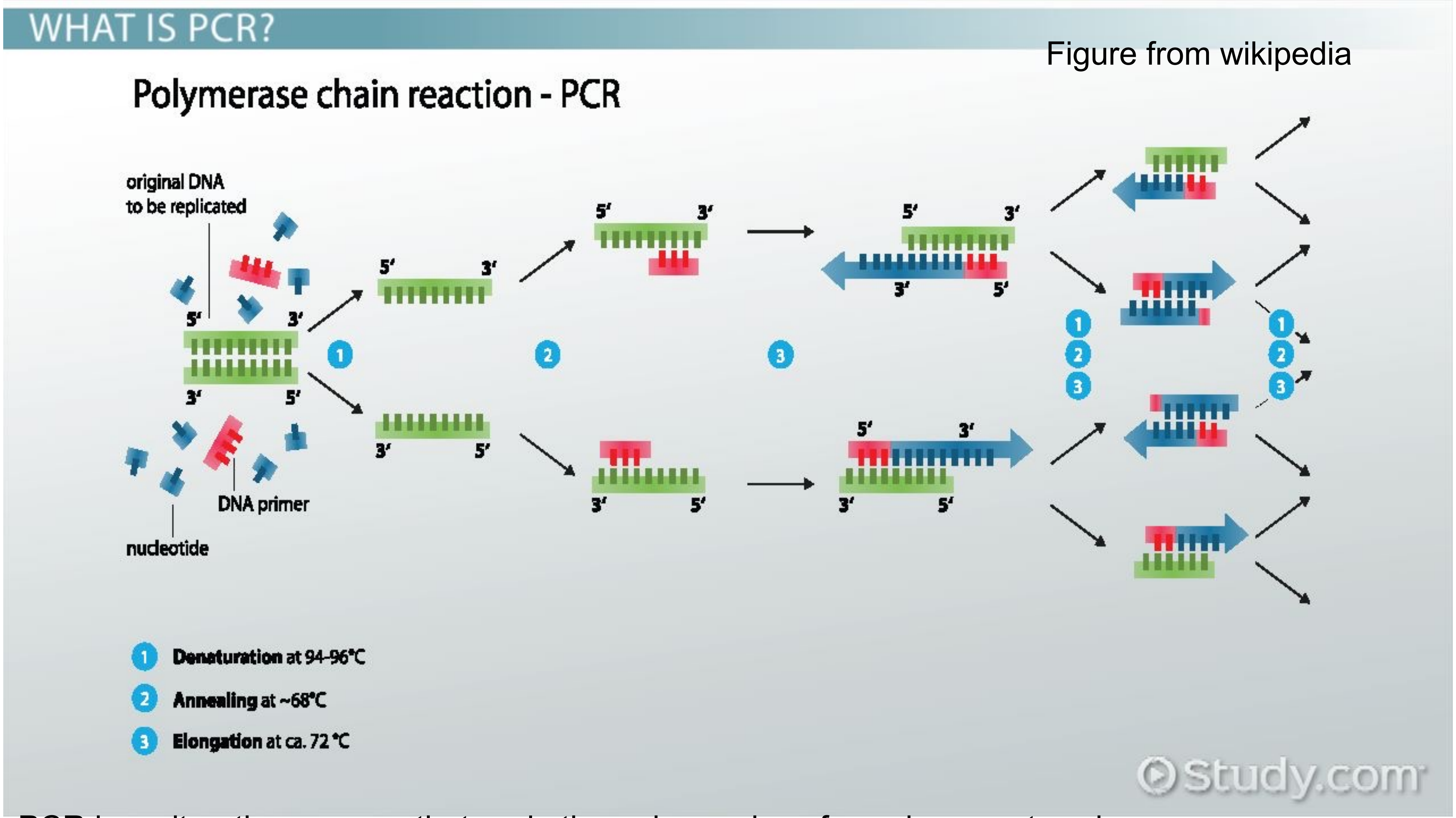
What we want are routes that start with LA and end with NY. To accomplish this we can use a technique called Polymerase Chain Reaction (PCR), which allows you to produce many copies of a specific sequence of DNA.

After many iterations of PCR, the DNA you're working on is amplified exponentially.

So to selectively amplify the itineraries that start and stop with our cities of interest, we use primers that are complimentary to LA and NY.

What we end up with after PCR is a test tube full of double stranded DNA of various lengths, encoding itineraries that start with LA and end with NY.

Figure from wikipedia



WHAT IS PCR?

Polymerase chain reaction - PCR

1 **Denaturation** at 94-96°C

2 **Annealing** at ~68°C

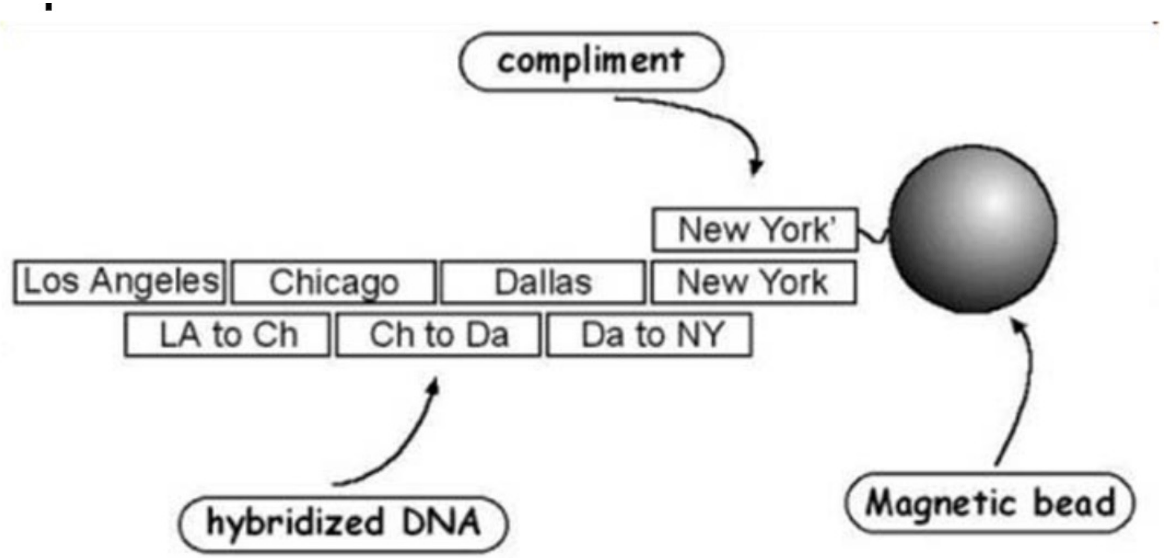
3 **Elongation** at ca. 72 °C

PCR is an iterative process that cycle through a series of copying events using an enzyme called polymerase. Polymerase will copy a section of single stranded DNA starting at the position of a primer, a short piece of DNA complimentary to one end of a section of the DNA that you're interested in.

By selecting primers that flank the section of DNA you want to amplify, the polymerase preferentially amplifies the DNA between these primers, doubling the amount of DNA containing this sequence.

Step 5: Itineraries Selection: have a Complete Set of Cities

DNA containing a specific sequence can be purified from a sample of mixed DNA by a technique called affinity purification, as shown below. This is accomplished by attaching the compliment of the sequence in question to a substrate like a magnetic bead. The beads are then mixed with the DNA. DNA, which contains the sequence you're after then hybridizes with the complement sequence on the beads. These beads can then be retrieved and the DNA isolated.



Select itineraries that have a complete set of cities. Sequentially affinity-purify n times, using a different city complement for each run. We are left with itineraries that start in LA, visit each city once, and end in NY.

Adleman’s approach pros & cons

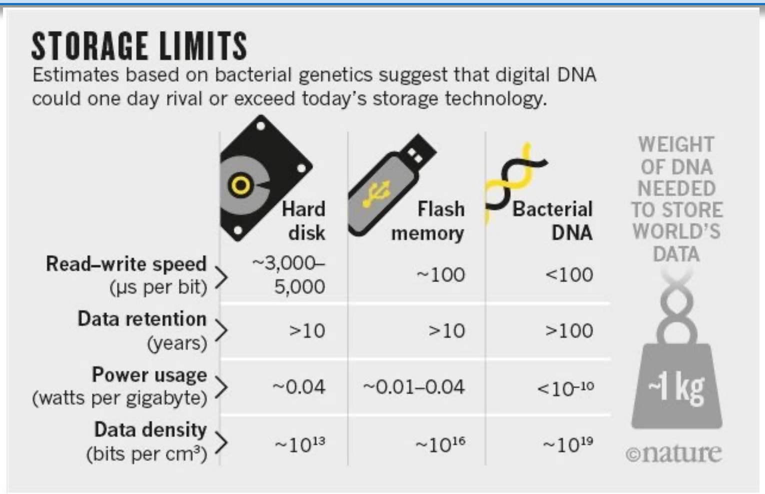
1 gram of DNA can hold about 1x10^{14} MB of data. A test tube of DNA can contain trillions of strands. 5 grams of DNA contain 10 ^{21} bases (Zetta Bytes) Each operation on a test tube of DNA is carried out on all strands in the tube in parallel (Speed: 500-5000 base pairs a second); Adleman estimated 2 x 10^{19} operations per joule.

Adleman's experiment solved a seven city problem, but there are two major shortcomings preventing a large scaling up of his computation.

The complexity of the Hamiltonian problem simply doesn’t disappear when applying a different method of solution - it still increases exponentially. Adleman’s process to solve the Hamiltonian problem for 200 cities would require an amount of DNA that weighed more than the Earth.

Random access in large-scale DNA data storage

DNA is not only BIG data:
It is also a way to store information and computing. More at the end!



STORAGE LIMITS

Estimates based on bacterial genetics suggest that digital DNA could one day rival or exceed today's storage technology.

	Hard disk	Flash memory	Bacterial DNA
Read–write speed (μs per bit)	~3,000–5,000	~100	<100
Data retention (years)	>10	>10	>100
Power usage (watts per gigabyte)	~0.04	~0.01–0.04	$<10^{-10}$
Data density (bits per cm³)	$\sim 10^{13}$	$\sim 10^{16}$	$\sim 10^{19}$

WEIGHT OF DNA NEEDED TO STORE WORLD'S DATA

©nature

The data longevity and information density of current DNA data storage systems already surpass those of traditional storage systems, but the cost and the read and write speeds do not.

Storing one megabyte of data in DNA with existing technology costs hundreds of dollars, compared with less than $0.0001 per year using tape, the standard for archival data storage.

The price of DNA storage will undoubtedly drop substantially as the costs of DNA synthesis and sequencing fall.

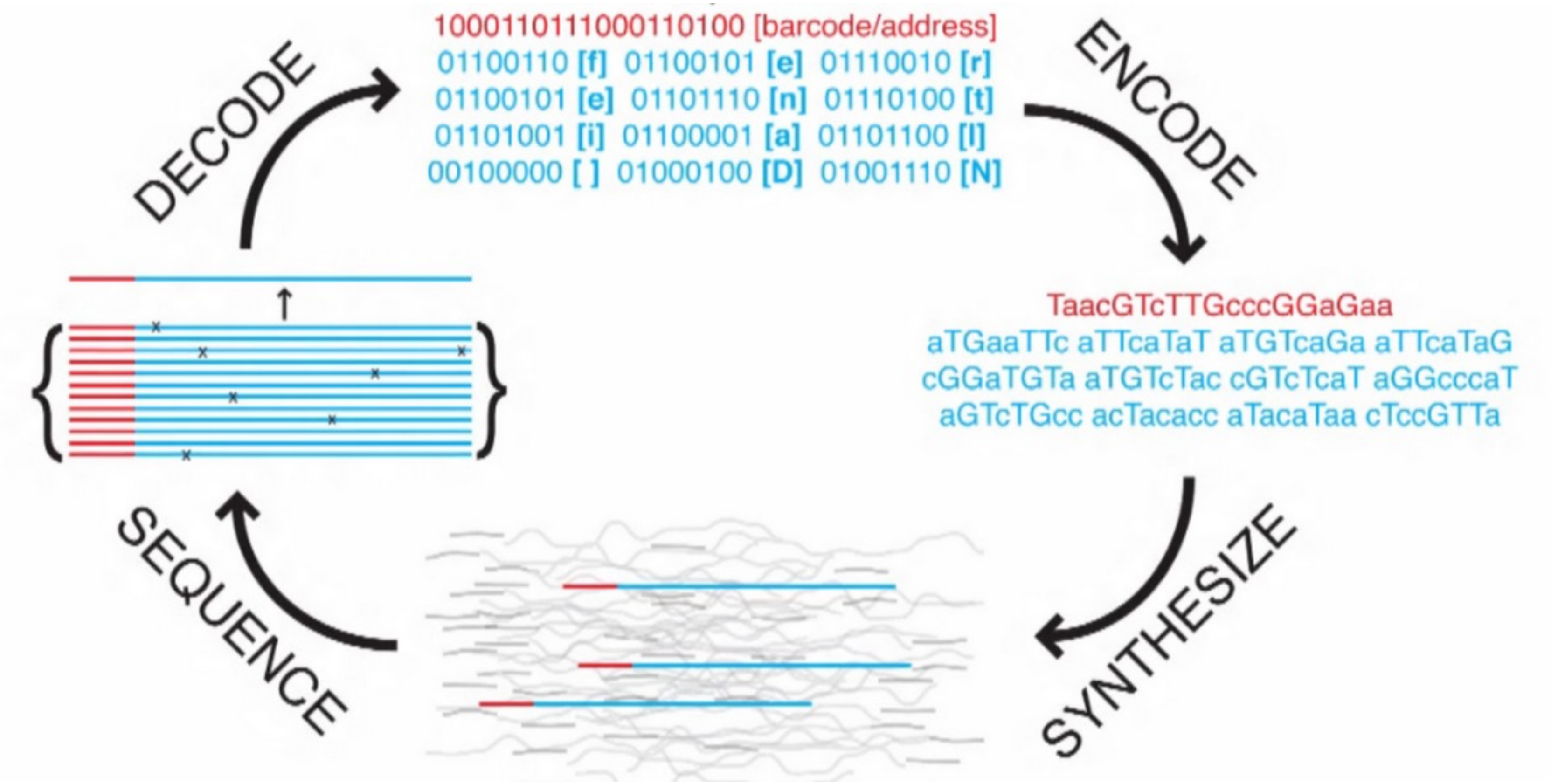
The more pressing challenge is that DNA synthesis and sequencing are inherently slow.

DNA synthesis and sequencing DNA can be extensively parallelized, their slow speeds limit the amount of data that can be written and read in a given time interval. The bottleneck for both cost and speed is synthesis.

A fully automated DNA drive would include synthesis and sequencing technology, components to store and handle the DNA, as well as a supply of chemicals.

Random access in large-scale DNA data storage

DNA strands that store 96 bits are synthesized, with each of the bases (TGAC) representing a binary value (T and G = 1, A and C = 0).
To read the data stored in DNA, you simply sequence it — just as if you were sequencing the human genome — and convert each of the TGAC bases back into binary. To aid with sequencing, each strand of DNA has a 19-bit address block at the start (the red bits in the image below) — so a whole vat of DNA can be sequenced out of order, and then sorted into usable data using the addresses.



1000110111000110100 [barcode/address]
01100110 [f] 01100101 [e] 01110010 [r]
01100101 [e] 01101110 [n] 01110100 [t]
01101001 [i] 01100001 [a] 01101100 [l]
00100000 [ ] 01000100 [D] 01001110 [N]

TaacGTcTTGcccGGaGaa
aTGaaTTc aTTcaTaT aTGTcaGa aTTcaTaG
cGGaTGTa aTGTcTac cGTcTcaT aGGcccaT
aGTcTGcc acTacacc aTacaTaa cTccGTTa

Random access in large-scale DNA data storage

Synthetic DNA is durable and can encode digital data with high density, making it an attractive medium for data storage.

However, recovering stored data on a large-scale currently requires all the DNA in a pool to be sequenced, even if only a subset of the information needs to be extracted.

Here, they encode and store 35 distinct files (over 200 MB of data), in more than 13 million DNA oligonucleotides, and show that they can recover each file individually and with no errors, using a random access approach.

Organik et al design and validate a large library of primers that enable individual recovery of all files stored within the DNA. These advances demonstrate a viable, large-scale system for DNA data storage and retrieval.

Random access in large-scale DNA data storage

Organick et al. stored and retrieved more than 200 megabytes of data.

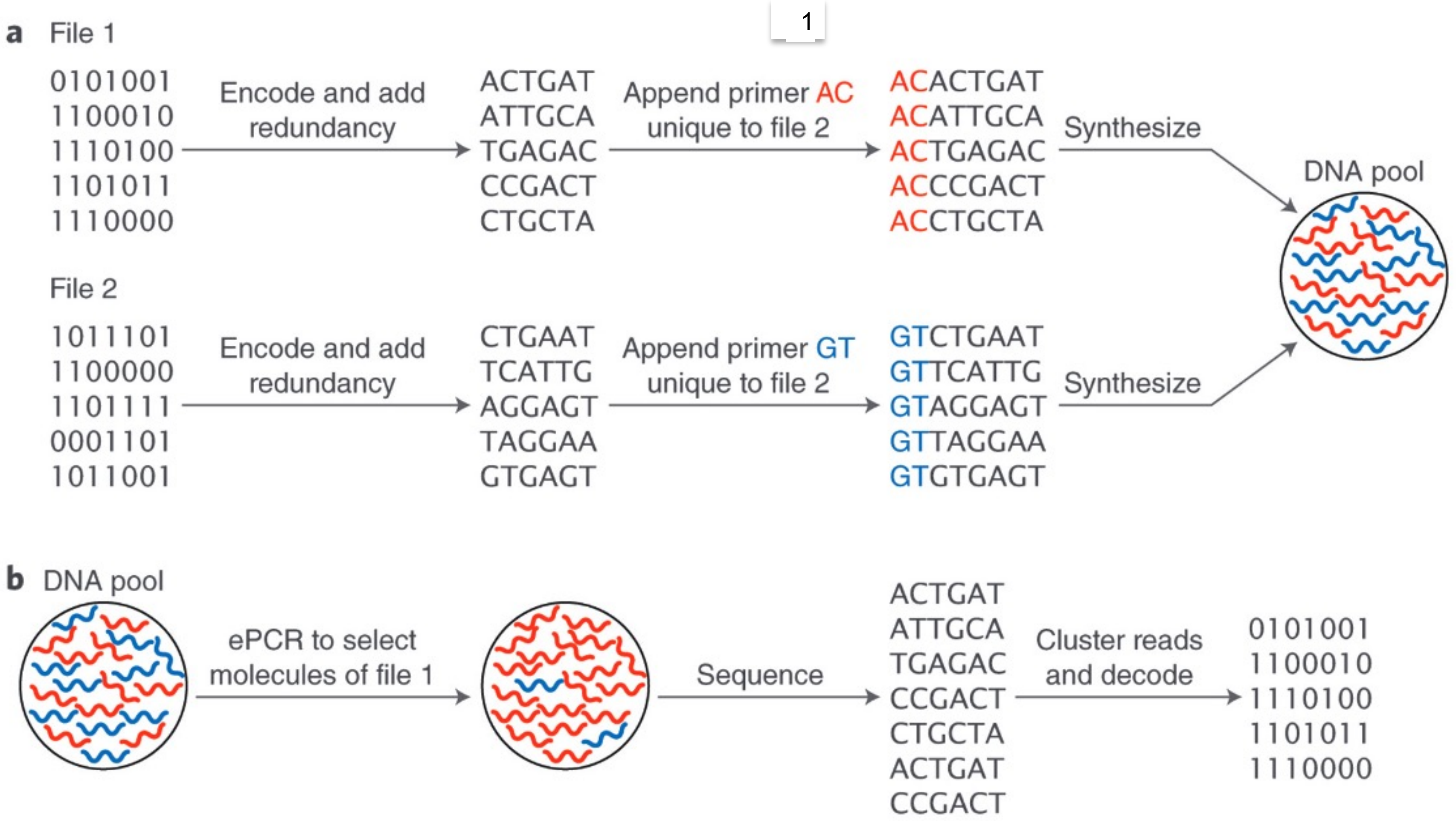
Specifically, they attach distinct primers to each set of DNA molecules carrying information about a file. This allows them to retrieve a given file by selectively amplifying and sequencing only the molecules with the primer marking the desired file.

To test their scheme, they designed a primer library that allowed them to uniquely tag data stored in DNA. They encoded 35 digital files into 13,448,372 DNA sequences, each 150-nucleotides long. Redundant information using error detection codes is also included to increase robustness to missing sequences and errors.

To improve recovery of the information, Organick et al. develop a clustering and consensus algorithm that aligns and filters reads before error correction.

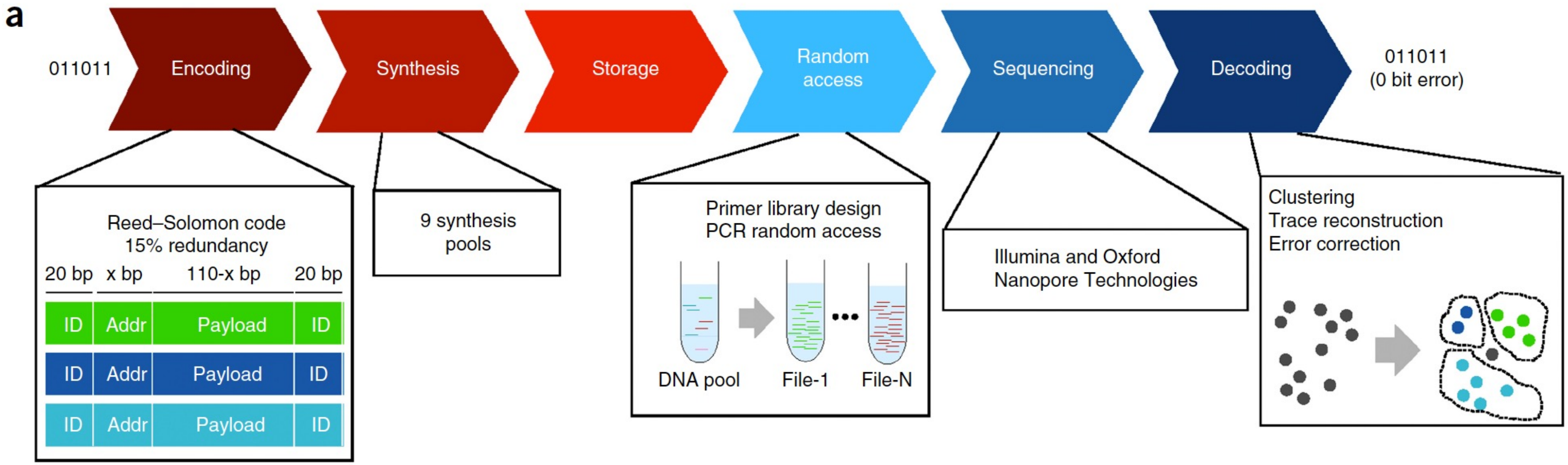
This algorithm also takes into account reads that differ from the correct length.

Random access in large-scale DNA data storage



The principle of DNA information storage in Organick et al. (a) Two files are stored by encoding each file in a set of different DNA sequences. Redundant information is added to enable error recovery at retrieval, and a distinct primer is appended to each set of sequences corresponding to a file. The resulting strings are synthesized and stored as a pool of different DNA molecules. (b) A specific file is retrieved by amplifying the molecules corresponding to the file by ePCR, sequencing the PCR products, and algorithmically reconstructing the data from the reads.

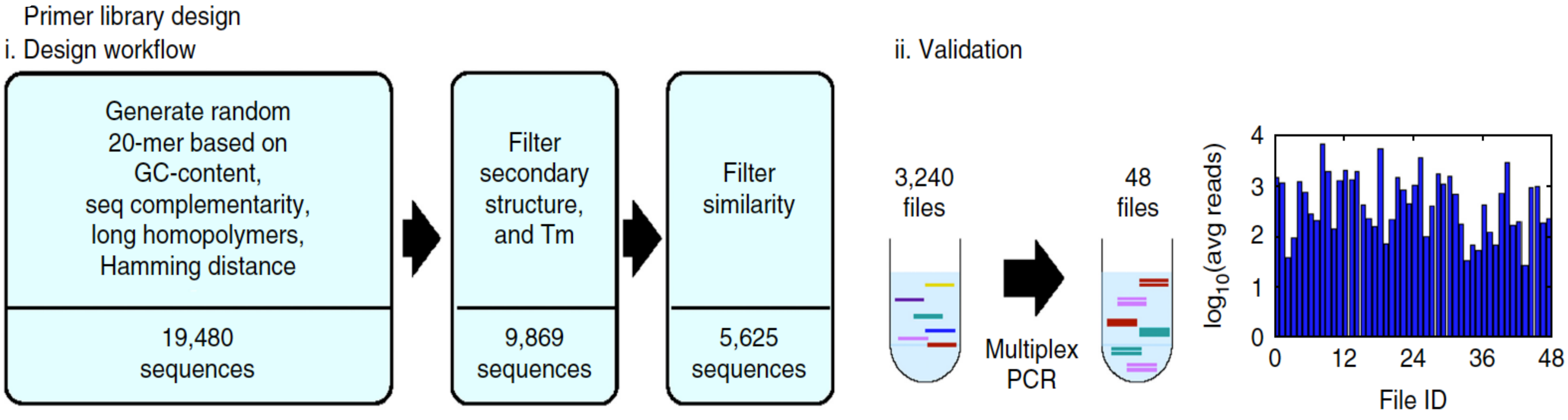
Random access in large-scale DNA data storage



This work describes large-scale random access, low redundancy, and robust encoding and decoding of information stored in DNA, as well as a notable increase in the volume of data stored (200 MB, the largest synthetic DNA pool available to date).Overview of the DNA data storage workflow and stored data.

(a) The encoding process maps digital files into a large set of 150-nucleotide DNA sequences, including Reed–Solomon code redundancy to overcome errors in synthesis and sequencing. The resulting collection of sequences is synthesized. The random access process starts with amplifying a subset of the sequences corresponding to one of the files using PCR. The amplified pools are sequenced. Finally, sequencing reads are decoded using clustering, consensus and error correction algorithms.

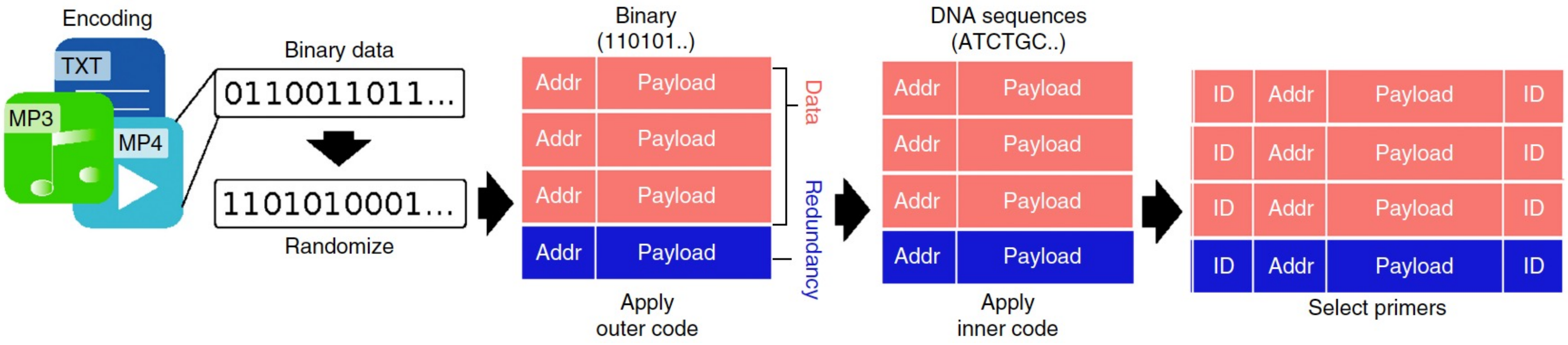
Random access in large-scale DNA data storage



Design of random access primers and coding algorithm.

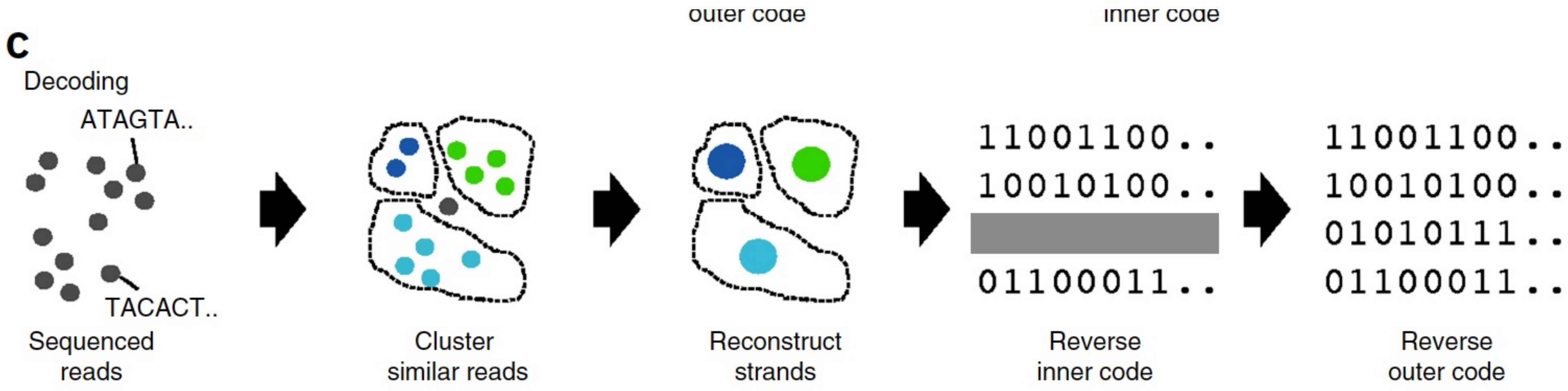
(i) They designed a primer library. The primer sequence set is then filtered that has low similarity between the sequences. (a, ii) The resulting set of candidate primers is then validated experimentally by synthesizing a pool of about 100,000 strands containing sets of size 1 to 200 DNA sequences each, surrounded by one of the candidate primer pairs, and then randomly selecting 48 of those pairs for amplification. The product is sequenced, and sequences with each of the 48 primer pairs appear among sequencing reads, albeit at different relative proportions when normalized to the number of sequences in each set.

Random access in large-scale DNA data storage



The encoding process starts by randomizing data to reduce chances of secondary structures, primer–payload non-specific binding, and improved properties during decoding. It then breaks the data into fixed-size payloads, adds addressing information (Addr), and applies outer coding, which adds redundant sequences using a Reed–Solomon code to increase robustness to missing sequences and errors. The level of redundancy is determined by expected errors in sequencing and synthesis, as well as DNA degradation. Next, it applies inner coding, which ultimately converts the bits to DNA sequences. The resulting set of sequences is surrounded by a primer pair chosen from the library based on (low) level of overlap with payloads.

Random access in large-scale DNA data storage

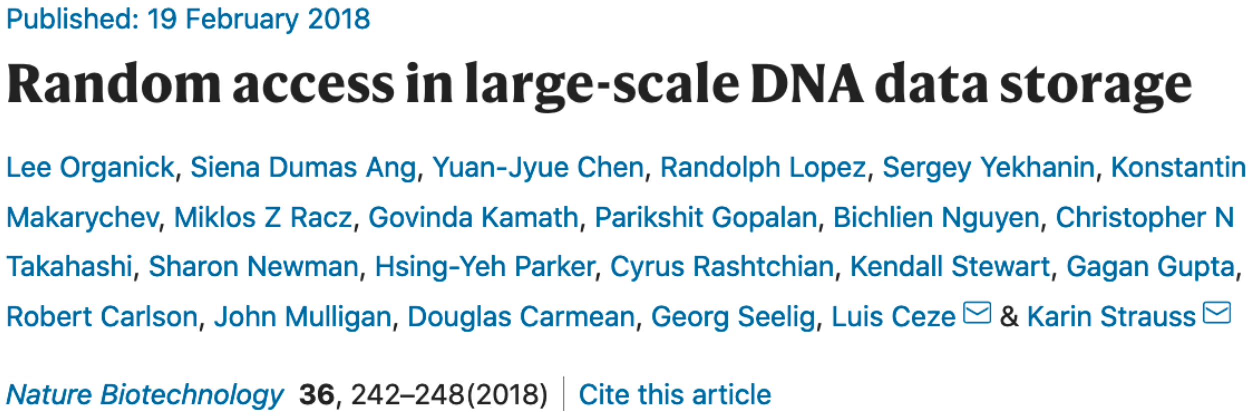


The decoding process starts by clustering reads based on similarity, and finding a consensus between the sequences in each cluster to reconstruct the original sequences, which are then decoded back to digital data.

Reference for this section

Reference: Adleman, L. M. (1994). “*Molecular computation of solutions to combinatorial problems*”. Science 266 (5187): 1021-1024. doi:10.1126/science.7973651. PMID 7973651

L. Adelman, *Scientific American*, pp. 54-61 (Aug 1998);



Published: 19 February 2018

Random access in large-scale DNA data storage

Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, Christopher N Takahashi, Sharon Newman, Hsing-Yeh Parker, Cyrus Rashtchian, Kendall Stewart, Gagan Gupta, Robert Carlson, John Mulligan, Douglas Carmean, Georg Seelig, Luis Ceze ✉ & Karin Strauss ✉

Nature Biotechnology **36**, 242–248(2018) | Cite this article

https://www.nature.com/articles/nbt.4079

Section 9

Simulation of biological reactions (also epidemics, social dynamics etc)

- Algorithm: Doob-Gillespie

Simulation of DNA and protein reactions

Problem statement: if we start with N types of molecules that can interact through one of M reactions at a given time, what will be the population levels of species after a given period of time?

One approach is to use ODE (obtaining a deterministic solution); another is to use an exact Stochastic Simulation that allows to: avoid averaging assumptions; it has a probabilistic formulation of the type:

– When does next reaction occur?

– Which reaction occurs next?

Advantages: continuous time, discrete population changes;

captures effects of noise; simple implementation; small memory requirements.

Disadvantages: CPU intensive; typically must simulate many runs; must use good random number generator

Doob-Gillespie algorithm to simulate reactions

- In a common chemical reaction system, two particles collide to form one or more products (see figure at the bottom).
- Biochemical reaction systems with a low to moderate number of molecules are often simulated (in well-stirred conditions) with methods that produce statistically exact sample paths such as the Doob-Gillespie algorithm
- The Doob-Gillespie algorithm uses two random numbers per step. The first is used to find when the next reaction occurs and the second is used to determine which reaction occurs at that time.
- It was developed by Joseph L. Doob and others (about 1945), used for chemical reactions by Dan Gillespie in 1976. The figures below show the set of reactions that involve 3 species; the system is updated after the interval τ.

R_1: $A + B \rightarrow C$

R_2: $C \,|\!\rightarrow A + B$

R_3: $A \rightarrow B$

R_4: $C \rightarrow$

System State:

	t	t+τ
A	1	2
B	1	2
C	8	7

How to simulate reactions

1. The idea of the Doob-Gillespie algorithm is that one first determines when something happens next.
2. Suppose the current time is t. Within a time t + τ a reaction could happen; we draw an exponentially distributed random number scaled by the sum of all process rates.
3. Then, the Doob-Gillespie algorithm determines what happens next. This is done by drawing a process randomly from all possible processes according to their respective probabilities (propensity functions).
4. When we have determined which process happens, we can update the variables (the so-called state of the system). Then we iterate this process as long as we want.
5. In practice the propensity function can be thought as a stochastic reaction rate; more formally in chemistry it describes the probability while reaction rate describes the changing rate. Propensity functions are defined based on population of species while the reaction rates are defined based on the concentration of species.

How to simulate DNA and protein reactions

A **propensity function** $\mathbf{a}_i$ is associated to each reaction step. These probabilites are related to the kinetics constants.

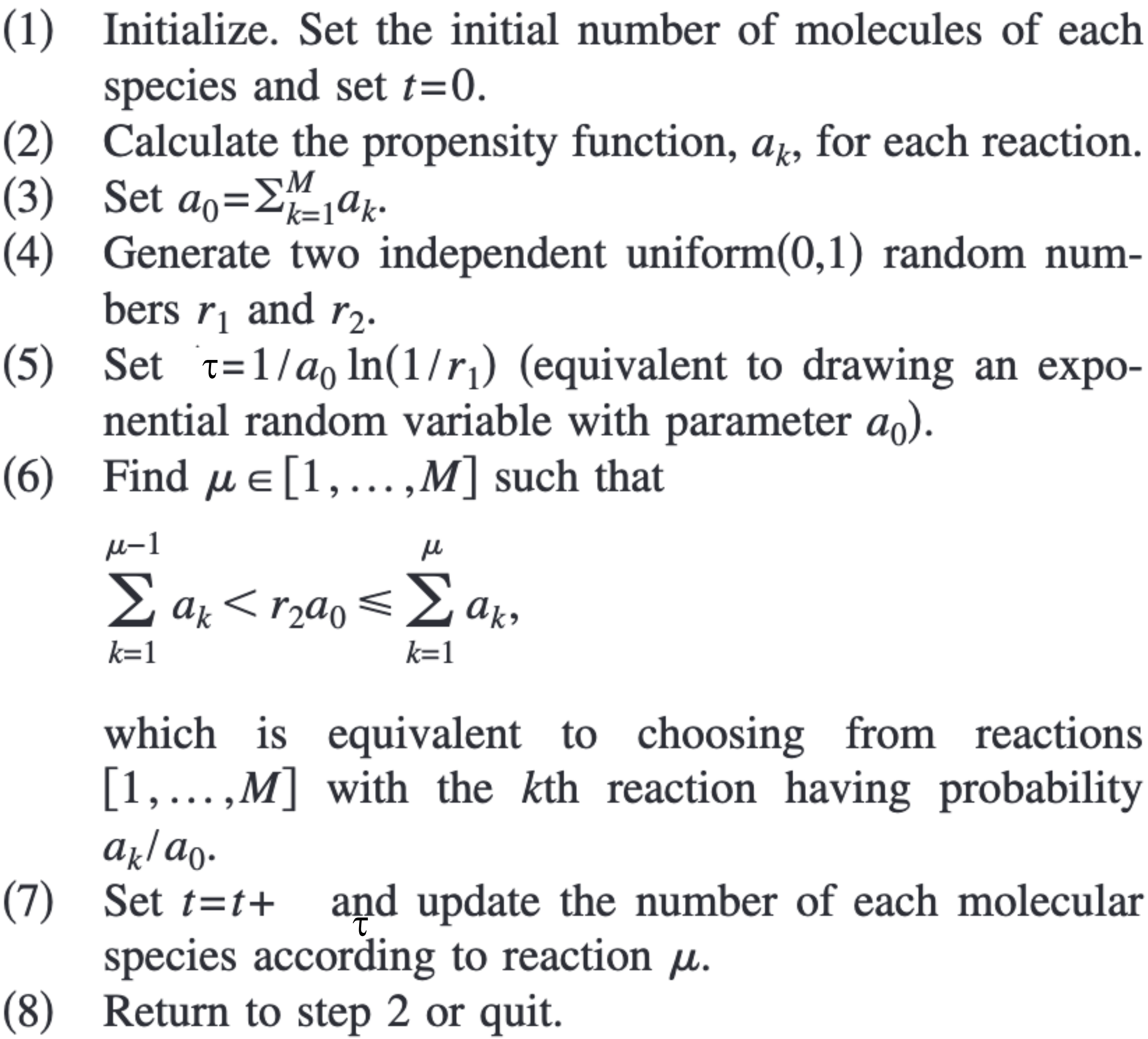
Initial number of molecules of each species are specified.

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

Generate r_1 and r_2 and calculate the reaction that occurs as well as the time till this reaction occurs.

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

Dobb-Gillespie Algorithm



(1) Initialize. Set the initial number of molecules of each species and set $t=0$.

(2) Calculate the propensity function, a_k, for each reaction.

(3) Set $a_0 = \Sigma_{k=1}^{M} a_k$.

(4) Generate two independent uniform(0,1) random numbers r_1 and r_2.

(5) Set $\tau = 1/a_0 \ln(1/r_1)$ (equivalent to drawing an exponential random variable with parameter a_0).

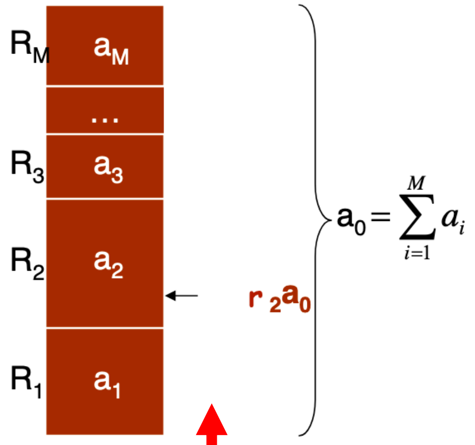
(6) Find $\mu \in [1,\ldots,M]$ such that

$$\sum_{k=1}^{\mu-1} a_k < r_2 a_0 \leqslant \sum_{k=1}^{\mu} a_k,$$

which is equivalent to choosing from reactions $[1,\ldots,M]$ with the kth reaction having probability a_k/a_0.

(7) Set $t=t+\tau$ and update the number of each molecular species according to reaction μ.

(8) Return to step 2 or quit.



Details on step 6

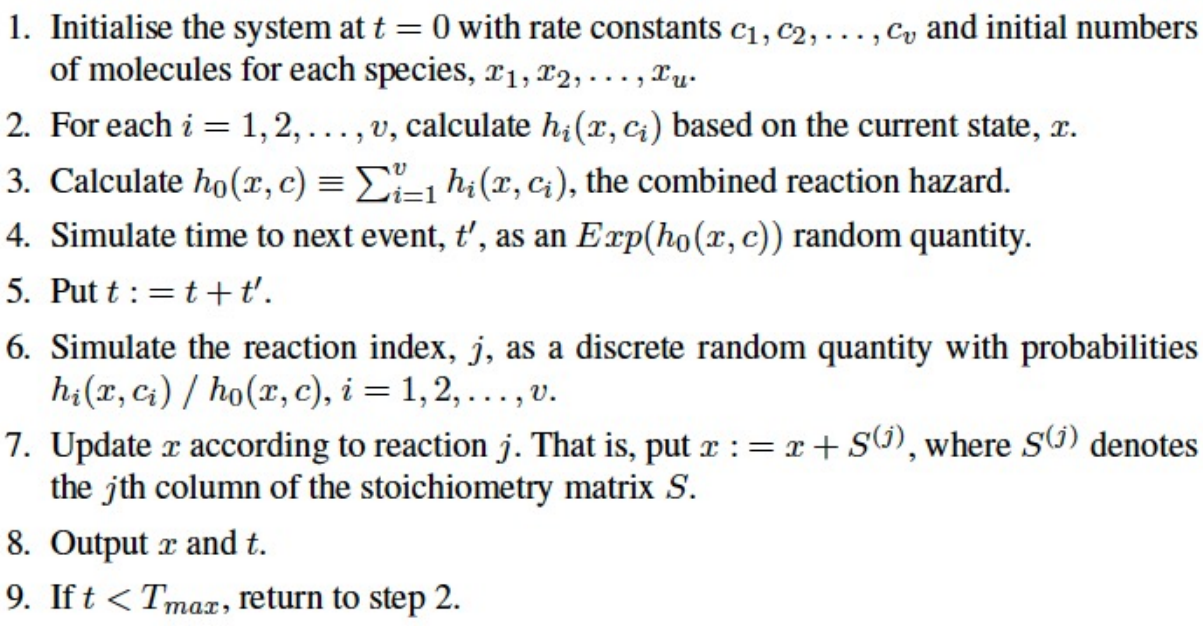
μ is the integer for which $\sum_{i=1}^{\mu-1} a_i < \quad \mathbf{r}_2 a_o < \sum_{i=1}^{\mu} a_i$

Examples

In a given reaction system with v reactions, we know that the hazard for a type i reaction is $h_i(x, c_i)$, so the hazard for a reaction of some type occurring is

$$h_0(x, c) \equiv \sum_{i=1}^{v} h_i(x, c_i).$$

It is clear that the time to the next reaction is $Exp(h_0(x, c))$, and also that this reaction will be a random type, picked with probabilities proportional to the $h_i(x, c_i)$, independent of the time to the next event. That is, the reaction type will be i with probability $h_i(x, c_i)/h_0(x, c)$. Using the time to the next event and the event type, the state of the system can be updated, and simulation can continue.



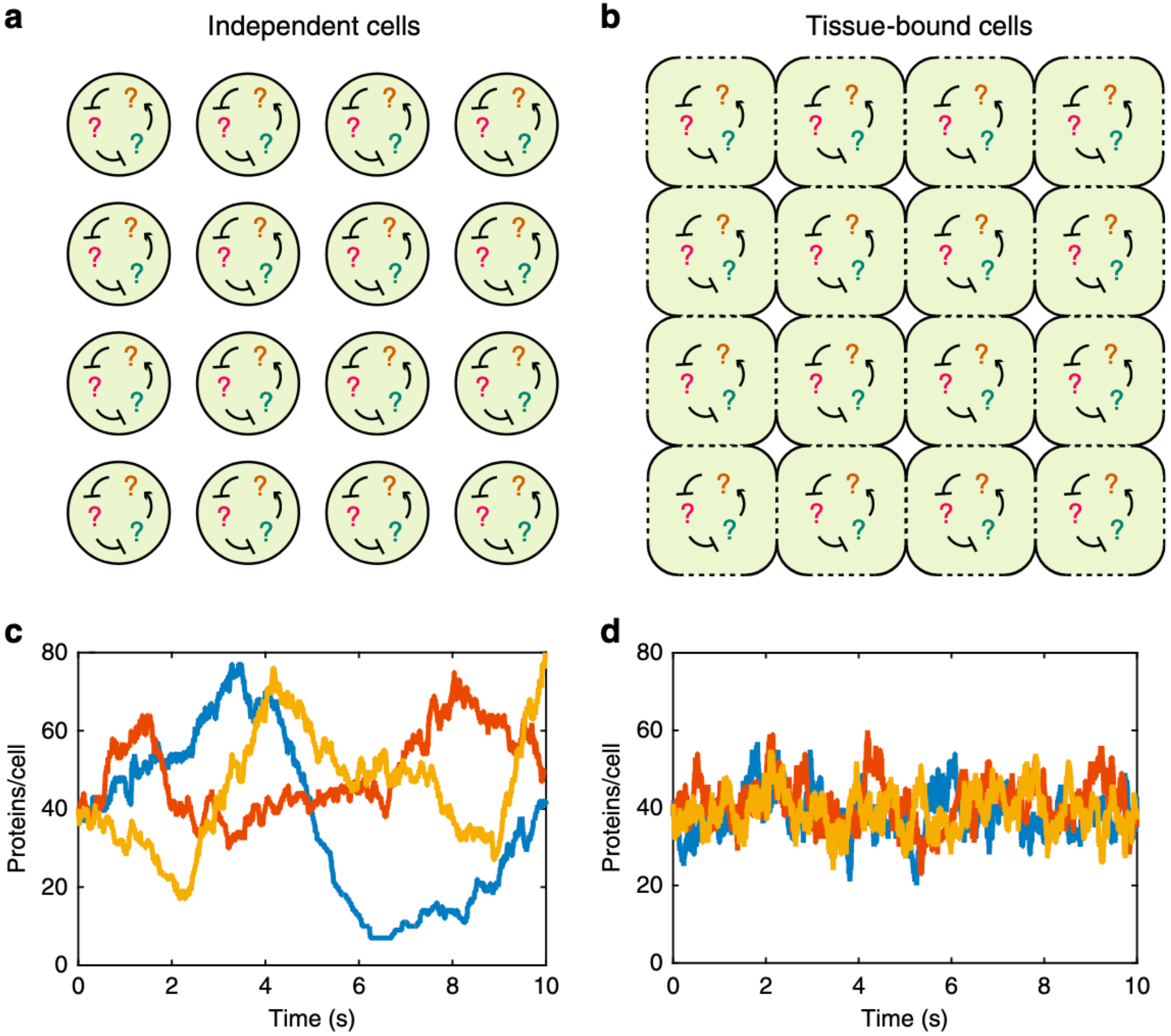
1. Initialise the system at $t = 0$ with rate constants $c_1, c_2, \ldots, c_v$ and initial numbers of molecules for each species, $x_1, x_2, \ldots, x_u$.
2. For each $i = 1, 2, \ldots, v$, calculate $h_i(x, c_i)$ based on the current state, x.
3. Calculate $h_0(x, c) \equiv \sum_{i=1}^{v} h_i(x, c_i)$, the combined reaction hazard.
4. Simulate time to next event, t', as an $Exp(h_0(x, c))$ random quantity.
5. Put $t := t + t'$.
6. Simulate the reaction index, j, as a discrete random quantity with probabilities $h_i(x, c_i) \,/\, h_0(x, c)$, $i = 1, 2, \ldots, v$.
7. Update x according to reaction j. That is, put $x := x + S^{(j)}$, where $S^{(j)}$ denotes the jth column of the stoichiometry matrix S.
8. Output x and t.
9. If $t < T_{max}$, return to step 2.

Complexity

- **Memory (N + 2M + 1)**
- N species populations
- (Compute a_i values for each of M reactions, compute $a_{0;}$ compute random numbers)
- **Total time scales with number of reactions that occur**
- Operation per reaction: generate two random numbers, μ, τ, calculate a_0 and a_i values.

Example of Doob-Gillespie application

Differences between a population of isolated cells and a tissue of cells. a) A population of isolated cells: each cell contains an identical genetic network (three species, two inhibiting and one activating functions). b) A tissue of cells: each cell contains an identical genetic network and some molecules can be transported between neighbouring cells (dotted lines).c) Typical single-cell protein trajectories of system (1) in isolated cells.d) Typical single-cell protein trajectories of system (1) in a tissue of connected cells: noise is clearly reduced compared to c.



Reference for this section

Gillespie D.T., (1976) A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *J. Comp. Phys*., 22: 403-434.

Stephen Smith& Ramon Grima 2018 Single-cell variability in multicellular organisms. Nature Communications

.

GillesPy2

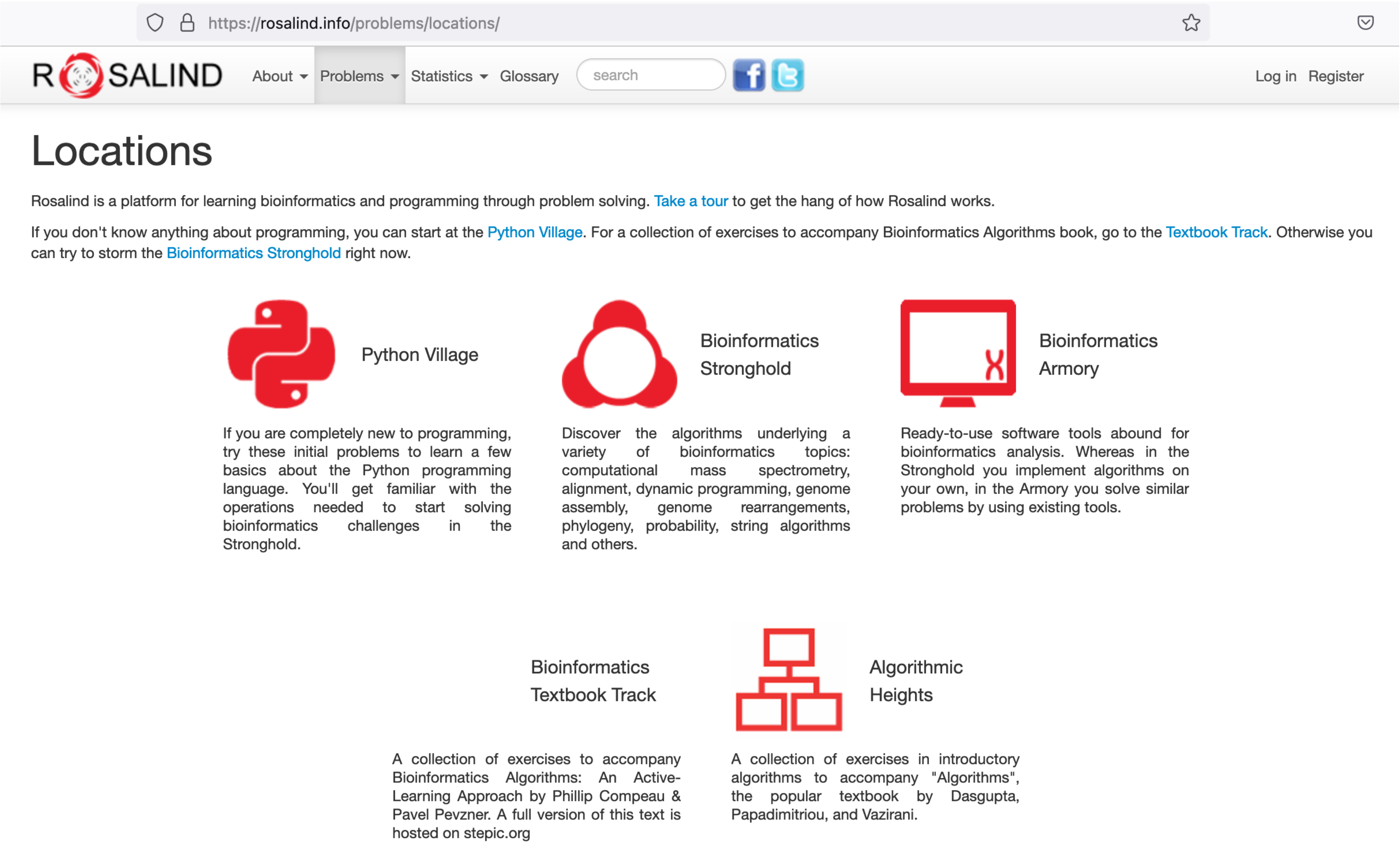
GillesPy2 is a Python 3 package for stochastic simulation of biochemical systems.

Library in python

Exam questions

Guidelines

- Algorithm (method, problem)
  - Name
  - Type of algorithm
  - Brief description (what it does?), input, output
  - Motivation (the problem it is trying to solve and why is it important?)
  - Assumptions
  - Main steps
  - Time and space complexity
  - Speed-up solutions, if applicable
  - When comparing: caveats and advantages (and when it is appropriate to use)
- Tip: make sure you know how to demonstrate with a small example
- Software, technique
  - Name
  - Brief description (what it does?)
  - Motivation (the problem it is trying to solve and why is it important?)
  - Assumptions
  - Input
  - Main steps
  - Output
  - When comparing: caveats and advantages (and when it is appropriate to use)
- Examples
  - Simple as possible
- Terms
  - Give a concise and complete definition



Locations

Rosalind is a platform for learning bioinformatics and programming through problem solving. Take a tour to get the hang of how Rosalind works.

If you don't know anything about programming, you can start at the Python Village. For a collection of exercises to accompany Bioinformatics Algorithms book, go to the Textbook Track. Otherwise you can try to storm the Bioinformatics Stronghold right now.

Python Village

If you are completely new to programming, try these initial problems to learn a few basics about the Python programming language. You'll get familiar with the operations needed to start solving bioinformatics challenges in the Stronghold.

Bioinformatics Stronghold

Discover the algorithms underlying a variety of bioinformatics topics: computational mass spectrometry, alignment, dynamic programming, genome assembly, genome rearrangements, phylogeny, probability, string algorithms and others.

Bioinformatics Armory

Ready-to-use software tools abound for bioinformatics analysis. Whereas in the Stronghold you implement algorithms on your own, in the Armory you solve similar problems by using existing tools.

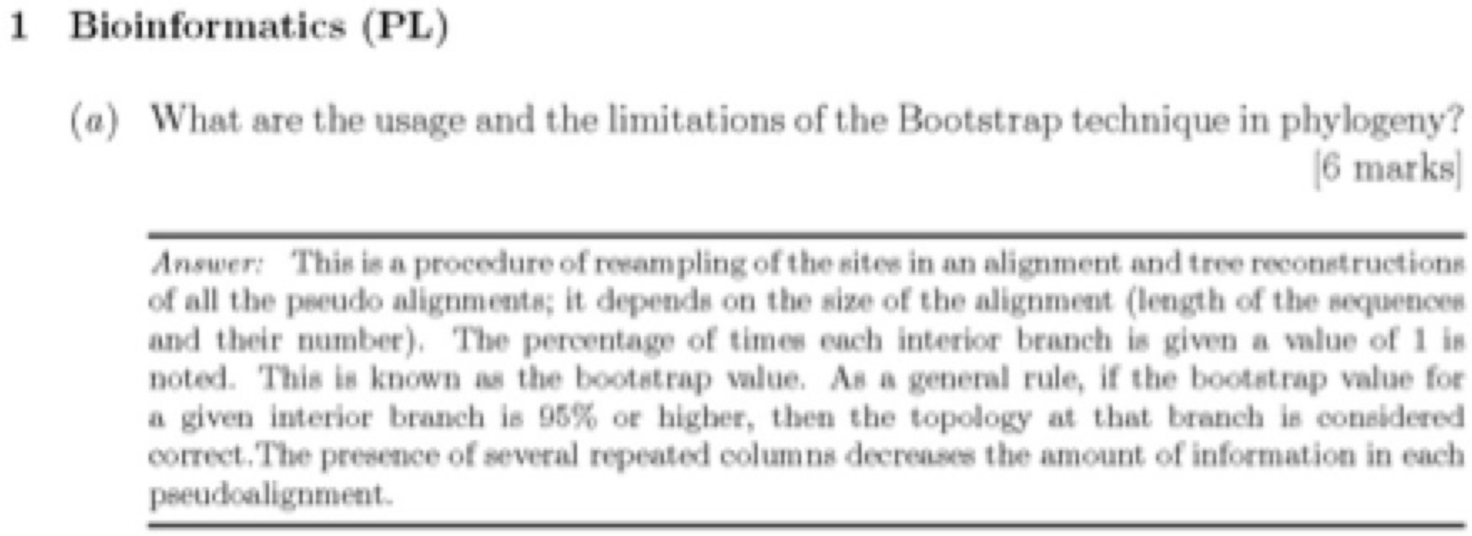
Bioinformatics Textbook Track

A collection of exercises to accompany Bioinformatics Algorithms: An Active-Learning Approach by Phillip Compeau & Pavel Pevzner. A full version of this text is hosted on stepic.org

Algorithmic Heights

A collection of exercises in introductory algorithms to accompany "Algorithms", the popular textbook by Dasgupta, Papadimitriou, and Vazirani.

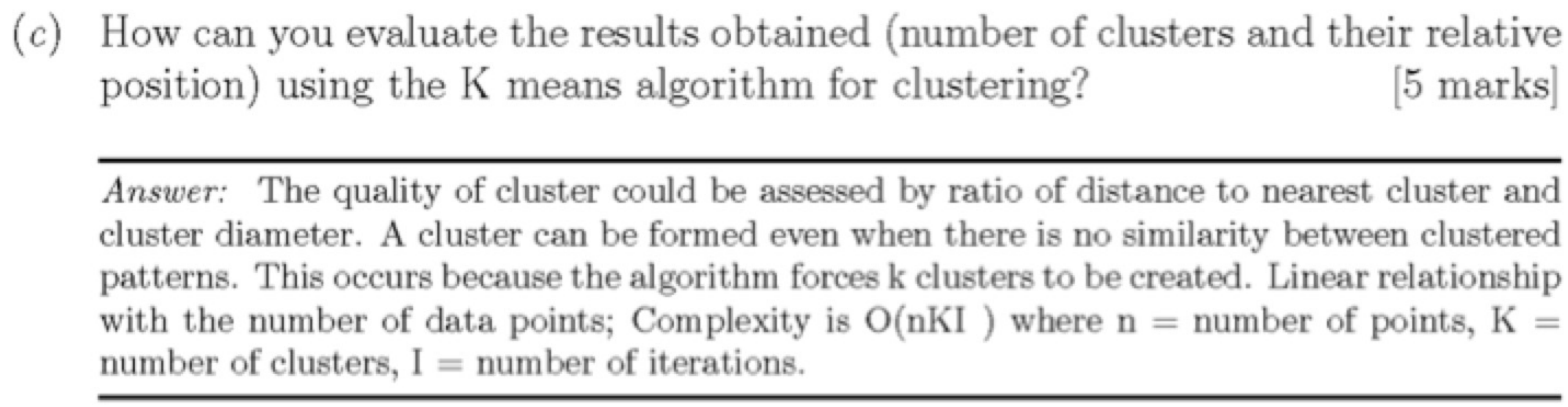
Exam questions



1 Bioinformatics (PL)

(*a*) What are the usage and the limitations of the Bootstrap technique in phylogeny? [6 marks]

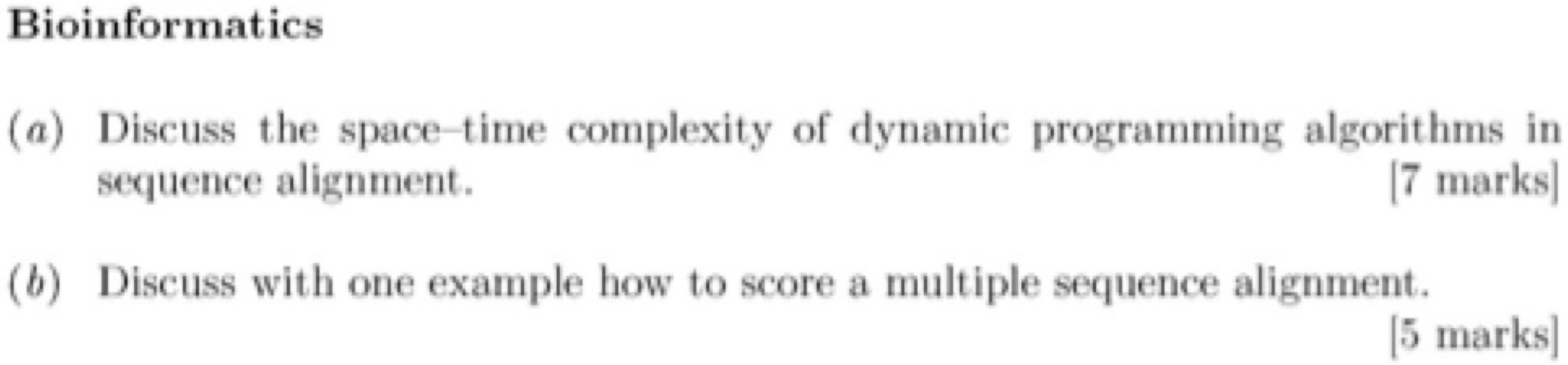
Answer: This is a procedure of resampling of the sites in an alignment and tree reconstructions of all the pseudo alignments; it depends on the size of the alignment (length of the sequences and their number). The percentage of times each interior branch is given a value of 1 is noted. This is known as the bootstrap value. As a general rule, if the bootstrap value for a given interior branch is 95% or higher, then the topology at that branch is considered correct.The presence of several repeated columns decreases the amount of information in each pseudoalignment.



(*c*) How can you evaluate the results obtained (number of clusters and their relative position) using the K means algorithm for clustering? [5 marks]

Answer: The quality of cluster could be assessed by ratio of 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; Complexity is O(nKI) where n = number of points, K = number of clusters, I = number of iterations.

Exam questions



Bioinformatics

(*a*) Discuss the space–time complexity of dynamic programming algorithms in sequence alignment. [7 marks]

(*b*) Discuss with one example how to score a multiple sequence alignment. [5 marks]

Exam questions

1. Give the alignment matrix of the sequences `AATCGCGCGGT' and `ATGCGCCGT' assuming the following costs: Cost(a,a)=0; Cost(a,b)=3 when a ≠ b, Cost(a,-)=Cost(-,a)=2.
2. How would you set the function Cost in order to compute the longest subsequence common to x and y?
3. Describe the differences between the algorithms for global and local alignments
4. Which of the following reasons would lead you to use the Smith-Waterman local alignment algorithm instead of the Needleman-Wunsch global alignment algorithm?

Select all appropriate answers.

(a) Computer memory is too limited to compute the optimal global alignment.

(b) One wants to identify common protein domains in the two sequences.

(c) The sequences have very different lengths.

(d) Smith-Waterman is faster than Needleman-Wunsch on long sequences.

5. Describe the notion of a parsimonious phylogeny for a finite set of sequences and the hypothesis assumed on them

3 Bioinformatics (PL)

Given the two DNA sequences: GCACTT and CCCAAT

(*a*) Compute the alignment (using the edit graph) and the final score with the following rules: match score = +1, mismatch = −1, gap penalty = −1. [4 marks]

(*b*) Discuss how the alignment score and the quality of the result depend on the match score, mismatch, and gap penalty. [6 marks]

(*c*) Generate four, short DNA sequences (a,b,c,d) such that their relations as a tree are approximately the following: ((a,b),(c,d)). [5 marks]

(*d*) How is the score matrix used in phylogenetic tree building techniques? [5 marks]

COMPUTER SCIENCE TRIPOS Part II – 2013 – Paper 9

1 Bioinformatics (PL)

(*a*) What are the usage and the limitations of the Bootstrap technique in phylogeny? [6 marks]

(*b*) We often use Hidden Markov Models (HMM) to predict a pattern (for instance the exons). How can you compute the number of True Positives, True Negatives, False Positives and False Negatives and use them to evaluate your HMM? [6 marks]

(*c*) How can you evaluate the results obtained (number of clusters and their relative position) using the K means algorithm for clustering? [5 marks]

HMM

(*b*) We often use Hidden Markov Models (HMM) to predict a pattern (for instance the exons). How can you compute the number of True Positives, True Negatives, False Positives and False Negatives and use them to evaluate your HMM?

[6 marks]

Answer:

(*i*) be predicted to occur: Predicted Positive (PP)

(*ii*) be predicted not to occur: Predicted Negative (PN)

(*iii*) actually occur: Actual Positive (AP)

(*iv*) actually not occur: Actual Negative (AN)

(*v*) True Positive $TP = PP \bigcap AP$

(*vi*) True Negative $TN = PN \bigcap AN$

(*vii*) False Negative $FN = PN \bigcap AP$

(*viii*) False Positive $FP = PP \bigcap AN$

(*ix*) Sensitivity: probability of correctly predicting a positive example Sn = TP/(TP + FN)

(*x*) Specificity: probability of correctly predicting a negative example Sp = TN/(TN + FP) or

(*xi*) probability that positive prediction is correct Sp = TP/(TP + FP)