Bioinformatics

Michaelmas 2022

Computer Science Tripos Part II

Pietro Lio
Ph.D. Engineering, Ph.D. Genetics
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

Course in 12 lectures; largely based on P. Compeau and P. Pevzner: Bioinformatics algorithms;

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
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.

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.
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
ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACCATTGTCATCTGACTCCTGA
GGAAGATCTGCGCTTACTGCCCTGTTGGGCAAGGTGAACGTGGATGAAGTTGGTGTGAGGCCTGGGC
AGGCTGCTGGTGTCTACCCCTTTGGACCCAGAGTTCTTCCTTGGAGTCCTTTTGGGATCTGTCCACTCCTGATG
CTGTTATGGGCAACCTAAGGTGAAGGCTCATGGCAAGAAAGTGCTCCTGGTGCTTTTAGTGTAGGGCCTGGA
TCACCTGGACAAACCTCAAGGGCAACCTTTGCCACACTGAAGCTGAGCTGACACTGCTGACACCTGGAT
CCTGAGAACTTTCAAGGACAGGTGGCTGGTCTGTGTGCTGGCATTCACTTTGAGGCAAGGAATTCA
CCCACCAGTGCGGAGCTGCCCCTATGAAAGGTGCTGGCTGCTGCTGGCCTGCTAGGGCCTGGCCACAAAGTATCA
CTAAGCTCTGCTTTCTTGGTGTCACTTCTTCTATTAAGGTTCCTTTTTGACCTTCCCTTGAGACATCCTACTAAACT
GGGGGATATTATGAAGGGCAACCTTGAGCATCCTGGATTTCTGCTCAATAAAAACATTTATTTTCATTGC

>gi|4504349|ref|NP_000509.1| beta globin [Homo sapiens]
MVHLTPFEEKSATVXLWGKVNVDEVGGEALGRLLVYPWTQRFSESFGDLSTPDAVMGNPKVKAHGKKVGL
AFSDGLAHLDNLKGTFLATLSELHCDKLHVDPNGRLLGNVLCVLAPHTFGKEFTPVPQAAYKVVAGVAN
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
ACATTTGCTTCTGACACACTGTGTTCACTAGCAACCTCAACAGACACCATGGTGATCTGACTCCTGAG
GCTGAAGTCTGCTGCCGTTTACTGCCTGTGGGCAAGGTGAACGTGGAAGTTGAGGTGAGGAGGCCTGGGC
AGGCTGCTGGTGTCTACCCCTTGGACCCAGAGGTTTCTTTGAGTCTCTTTGGGATCTTGCCACTCCTGATG
CTGTTATGGCACAACCTAAGGTGAAGGCTCATGGCAAGAAAAGTGCTCGGTGCTGCTTTTAGTGTAGGCGCTG
TCACCTGGACAAACCTCAAGGCACCTTGGCCACACTGAGTGAGCTGACACTTGACAAAGCTGCAGCTGGAT
CCTGAGAACTTCAGGCTCCTGGCAACGTGCTGGTCTGTGTGCTGGCCCATCACTTTGGCCAAAGAAATCCA
CCCCACCACTGCAAGCTGCCCTATCACAGAAAGTGGTGGCTGGTGTGGCTAATGCCCTGGCCCACAAGTATCA
CTAAGCTCGCTTTCTGGCTCCAATTTCTATATTAAAGGTTCTTTGCTTTTGGCCCTAAGTCAATCCTAAAAC
GGGGATATTATGAAAGGCCTTGAGCATCTGGATTCTGCCCATAAAGAAAAACATTTATTTTCATTG

>gi|4504349|ref|NP_000509.1| beta globin [Homo sapiens]
MVHLTPVEKSAVTALWGKVNDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDADMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTATLSELHCDKLHVDPENFRLLGNVLCVLAAHFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH
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.
Proteins: from sequence to 3d structure

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

5'-CCTGAGCCAACCTATTGATGAA-3'
3'-GGACTCGGTTGATAACTACTT-5'

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

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

Garage genomics
DNA is big data


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

doi:10.1371/journal.pbio.1002195.t001
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 $3.7 \times 10^{13}$ cells in the human body (and hence $1.7 \times 10^{15}$ chromosomes or strands), your entire DNA would stretch about $7.4 \times 10^{10}$ 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: $5.3 \times 10^{31}$ $\pm 3.6 \times 10^{31}$ 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.
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.

**Simple regulation**

**Complex regulation**
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).
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.
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.
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).
Logic gates: The Cell as an information processing device

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)
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.
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.
Scales of electronic and bio devices

proteins inside a bacterium

λ = 0.25 micron in Pentium II

1 micron

Bacterium

Human chromosome

(a) NAND gate layout geometry.
CS – Bio parallelism

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

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

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

# Cells versus Computers

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

**Challenges:** quantify information in a cell; building computers inspired by cells information flow
DNA, RNA and proteins can:

- Organise themselves to self assemble different types of devices (mechanisms such as 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
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

<table>
<thead>
<tr>
<th>‘Omics’</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genome</td>
<td>The full complement of genetic information both coding and non-coding in an organism</td>
</tr>
<tr>
<td>Proteome</td>
<td>The complete set of proteins expressed by the genome in an organism</td>
</tr>
<tr>
<td>Transcriptome</td>
<td>The population of mRNA transcripts in the cell, weighted by their expression levels as transcripts copy number</td>
</tr>
<tr>
<td>Metabolome</td>
<td>The quantitative complement of all the small molecules present in a cell in a specific physiological state</td>
</tr>
<tr>
<td>Interactome</td>
<td>Product of interactions between all macromolecules in a cell</td>
</tr>
<tr>
<td>Phenome</td>
<td>Qualitative identification of the form and function derived from genes, but lacking a quantitative, integrative definition</td>
</tr>
<tr>
<td>Glycome</td>
<td>The population of carbohydrate molecules in the cell</td>
</tr>
<tr>
<td>Translatome</td>
<td>The population of mRNA transcripts in the cell, weighted by their expression levels as protein products</td>
</tr>
<tr>
<td>Regulome</td>
<td>Genome-wide regulatory network of the cell</td>
</tr>
<tr>
<td>Operome</td>
<td>The characterization of proteins with unknown biological function</td>
</tr>
<tr>
<td>Synthetome</td>
<td>The population of the synthetic gene products</td>
</tr>
<tr>
<td>Hypothome</td>
<td>Interactome of hypothetical proteins</td>
</tr>
</tbody>
</table>
Single cell/ single tissue repositories

The figure shows the differences between single cell and classic microarray analysis.
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)
A nice free book: cell biology by the numbers
http://book.bionumbers.org/
Others:
http://tandy.cs.illinois.edu/Hunter_MolecularBiology.pdf
Biology and Computers: A lesson in what is possible
https://www.ncbi.nlm.nih.gov/home/tutorials/ and ebi.ac.uk
If you want to code you may have a look at these libraries:

- **BioJava** – [www.biojava.org](http://www.biojava.org)
- **BioPerl** – [www.bioperl.org](http://www.bioperl.org)
- **BioPython** – [www.biopython.org](http://www.biopython.org)
- **BioCorba** – [www.biocorba.org](http://www.biocorba.org)
- **BioRuby** – [www.bioruby.org](http://www.bioruby.org)
- **BioHaskell** – [www.biohaskell.org](http://www.biohaskell.org)
If you want data you may have a look at these repositories:

- 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 $x = x_1, x_2, \ldots, x_M$, $y = y_1, y_2, \ldots, y_N$, 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.

AGGCTATCACCTGACCTCCAGGCCGATGCC
TAGCTATCACGACCACCAGGCTGATTTGCCGAC

AGGCTATCACCTGACCTCCAGGCCGATGCC
TAGCTATCACGACCACCAGGCTGATTTGCCGAC
What Is the Sequence Alignment?

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

Left: mapping human chromosomes onto mouse chromosome reveal many similar regions (low resolution). Right: a higher resolution, each region reveals many local rearrangements.
A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

**Longest Common Subsequence**

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 \( w \):

\[
\begin{align*}
v & : \quad A \ T \ G \ T \ T \ A \ T \quad m = 7 \\
w & : \quad A \ T \ C \ G \ T \ A \ C \quad n = 7
\end{align*}
\]

Alignment: \( 2 \times k \) matrix (\( k > m, n \) )

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>--</th>
<th>G</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>A</td>
<td>T</td>
<td>C</td>
<td>G</td>
<td>T</td>
<td>--</td>
<td>A</td>
<td>--</td>
<td>C</td>
</tr>
</tbody>
</table>

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

and a sequence of positions in $w$: $1 \leq j_1 < j_2 < \ldots < j_t \leq n$

such that $i_t$-th letter of $v$ equals to $j_t$-th letter of $w$ and $t$ is maximal.
Every common subsequence is a path in 2-D grid
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 $w$.

$V = \text{ATATATAT}$
$W = \text{TATATATA}$

Hamming distance: $d(v, w) = 8$
Computing Hamming distance is a trivial task.

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

$V = \text{-ATATATAT}$
$W = \text{TATATATA-}$

Edit distance: $d(v, w) = 2$
Computing edit distance is a non-trivial task.
Two similar alignments; the score is 5 for both the alignment paths.
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
Let \( v_i = \) prefix of \( v \) of length \( i \): \( v_1 \ldots v_i \)
and \( w_j = \) prefix of \( w \) of length \( j \): \( w_1 \ldots w_j \)
The length of LCS(\( v_i, w_j \)) is computed by:

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

Every Path in the Grid Corresponds to an Alignment

V = AT - GT
W= A T C G -

0 1 2 3 4

49
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 $t \times t$ 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 $O(\log n)$. 

LCS pseudocode

```plaintext
LCS(v, w)
1 for i ← 0 to n
2 \quad s_{t,0} ← 0
3 for j ← 1 to m
4 \quad s_{0,j} ← 0
5 for i ← 1 to n
6 \quad for j ← 1 to m
7 \quad s_{t,j} ← \max \left\{ \begin{align*}
8 \quad & s_{t-1,j} \\
9 \quad & s_{t,j-1} + 1, \quad \text{if } v_i = w_j \\
10 \quad & s_{t-1,j-1} + 1, \quad \text{if } v_i = w_j
11 \end{align*} \right. \\
12 \quad b_{t,j} \left\{ \begin{align*}
13 \quad & \uparrow \quad \text{if } s_{t,j} = s_{t-1,j} \\
14 \quad & \leftarrow \quad \text{if } s_{t,j} = s_{t,j-1} \\
15 \quad & \downarrow \quad \text{if } s_{t,j} = s_{t-1,j-1} + 1
16 \end{align*} \right.
17 return (s_{n,m}, b)
```

```plaintext
PRINTLCS(b, v, i, j)
1 if i = 0 or j = 0
2 \quad return \\
3 if b_{i,j} = "\uparrow"
4 \quad PRINTLCS(b, v, i-1, j-1)
5 \quad print v_i \\
6 else
7 \quad if b_{i,j} = "\leftarrow"
8 \quad PRINTLCS(b, v, i-1, j)
9 \quad else
10 \quad PRINTLCS(b, v, i, j-1)
```
$s_{i, j} = \max$

$S_{i-1, j} - \sigma$

$S_{i, j-1} - \sigma$

$S_{i-1, j-1} + 1$, if $v_i = w_j$

$S_{i-1, j-1} - \mu$, if $v_i \neq w_j$

General Alignment Graph
Towards an algorithm to align biological sequences

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.

Notice three possible cases:

1. $x_i$ aligns to $y_j$
   
   \[
   x_1 \ldots x_{i-1} \ x_i \\
   y_1 \ldots y_{j-1} \ y_j
   \]

   \[
   F(i,j) = F(i-1, j-1) + m, \text{ if } x_i = y_j
   \]

   \[
   F(i,j) = F(i-1, j-1) - s, \text{ if not}
   \]

2. $x_i$ aligns to a gap

   \[
   x_1 \ldots x_{i-1} \ x_i \\
   y_1 \ldots y_j
   \]

   \[
   F(i,j) = F(i-1, j) - d
   \]

3. $y_j$ aligns to a gap

   \[
   x_1 \ldots x_i
   \]

   \[
   y_1 \ldots y_{j-1} \ y_j
   \]

   \[
   F(i,j) = F(i, j-1) - d
   \]
Alignment

- How do we know which case is correct?

**Inductive assumption:**

<table>
<thead>
<tr>
<th>F[i-1, j-1]</th>
<th>F[i, j-1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>F[i-1, j]</td>
<td>F[i, j]</td>
</tr>
</tbody>
</table>

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

Then,

\[
F(i, j) = \max \left\{ F(i-1, j-1) + s(x_i, y_j), \ F(i-1, j) - d, \ F(i, j-1) - d \right\}
\]

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

```
tccCAGTTATGTCA\text{Gg}gacacgcagcatgcagagac
|           |           |           |           |           |           |           |           |
aattgc\text{g}gccggtcgt\text{g}t\text{t}tttcagCAGTTATGTCA\text{Gatc}
```
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
   For each $i = 1, \ldots, M$
   For each $j = 1, \ldots, N$
   \[
   F(i, j) = \max \begin{cases} 
   F(i-1, j) - d & \text{[case 1]} \\
   F(i, j-1) - d & \text{[case 2]} \\
   F(i-1, j-1) + s(x_i, y_j) & \text{[case 3]} 
   \end{cases}
   \]
   \[
   \text{Ptr}(i, j) = \begin{cases} 
   \text{UP}, & \text{if [case 1]} \\
   \text{LEFT}, & \text{if [case 2]} \\
   \text{DIAG}, & \text{if [case 3]} 
   \end{cases}
   \]

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

---

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:

\[
\begin{align*}
\text{CTATCAC} & \text{CTGACCT} \text{CCA} \text{GCG} \text{ATGCCCCTTCCGGC} \\
\text{GCGAGTTCAT} & \text{CTATCAC} -- \text{GACCGC} -- \text{GGTCG} \\
\end{align*}
\]

Changes:

1. **Initialization**
   For all \( i, j \),
   \[
   F(i, 0) = 0 \\
   F(0, j) = 0
   \]

2. **Termination**
   \[
   F_{OPT} = \max \left\{ \max_i F(i, N), \max_j F(M, j) \right\}
   \]
Local Alignment = Global Alignment in a subrectangle
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

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

e.g. \( x = \text{aaaacccccgggg} \)
\( y = \text{ccccgggaaccaacc} \)

**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...
   \( F_{\text{OPT}} = \max_{i,j} F(i, j) \)
2. If we want **all** local alignments scoring > \( t \)
   For all \( i, j \) find \( F(i, j) > t \), and trace back

David Waterman
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
We previously assigned a fixed penalty $\sigma$ 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: GA--CAG GA--CAG

(GA-C-AG GA--CAG) (higher score)
#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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>−</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>+1</td>
<td>−μ</td>
<td>−μ</td>
<td>−μ</td>
<td>−σ</td>
</tr>
<tr>
<td>C</td>
<td>−μ</td>
<td>+1</td>
<td>−μ</td>
<td>−μ</td>
<td>−σ</td>
</tr>
<tr>
<td>G</td>
<td>−μ</td>
<td>−μ</td>
<td>+1</td>
<td>−μ</td>
<td>−σ</td>
</tr>
<tr>
<td>T</td>
<td>−μ</td>
<td>−μ</td>
<td>−μ</td>
<td>+1</td>
<td>−σ</td>
</tr>
<tr>
<td>−</td>
<td>−σ</td>
<td>−σ</td>
<td>−σ</td>
<td>−σ</td>
<td>−σ</td>
</tr>
</tbody>
</table>

Scoring matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>−</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>+1</td>
<td>−3</td>
<td>−5</td>
<td>−1</td>
<td>−3</td>
</tr>
<tr>
<td>C</td>
<td>−4</td>
<td>+1</td>
<td>−3</td>
<td>−2</td>
<td>−3</td>
</tr>
<tr>
<td>G</td>
<td>−9</td>
<td>−7</td>
<td>+1</td>
<td>−1</td>
<td>−3</td>
</tr>
<tr>
<td>T</td>
<td>−3</td>
<td>−5</td>
<td>−8</td>
<td>+1</td>
<td>−4</td>
</tr>
<tr>
<td>−</td>
<td>−4</td>
<td>−2</td>
<td>−2</td>
<td>−1</td>
<td>−1</td>
</tr>
</tbody>
</table>

Even more general scoring matrix
example: Y (Tyr) often mutates into F (score +7) but rarely mutates into P (score -5)

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

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>S</td>
<td>T</td>
<td>P</td>
<td>A</td>
<td>G</td>
<td>N</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-2</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>-3</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>-3</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>-4</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Q</td>
<td>-5</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>H</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>R</td>
<td>-4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-2</td>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>-5</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
</tr>
<tr>
<td>I</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>L</td>
<td>-6</td>
<td>-3</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>-3</td>
<td>-4</td>
<td>-3</td>
</tr>
<tr>
<td>V</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>F</td>
<td>-4</td>
<td>3</td>
<td>-5</td>
<td>-4</td>
<td>-5</td>
<td>-2</td>
<td>-4</td>
<td>3</td>
<td>-5</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>3</td>
<td>-5</td>
<td>-3</td>
<td>-3</td>
<td>2</td>
<td>4</td>
<td>-4</td>
<td>4</td>
</tr>
<tr>
<td>W</td>
<td>-8</td>
<td>2</td>
<td>-5</td>
<td>-6</td>
<td>-6</td>
<td>-7</td>
<td>-4</td>
<td>-7</td>
<td>-7</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>S</td>
<td>T</td>
<td>P</td>
<td>A</td>
<td>G</td>
<td>N</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>

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.
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.

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


$\sigma$ - the gap opening penalty
$\epsilon$ - the gap extension penalty
$\sigma > \epsilon$, start a gap is penalized more than extending it.

$\text{lower}_{i,j} = \max \{ \text{lower}_{i-1,j} - \epsilon, \text{middle}_{i-1,j} - \sigma \}$

$\text{middle}_{i,j} = \max \{ \text{middle}_{i-1,j-1} + \text{score}(v_i, w_j), \text{upper}_{i-1,j} - \epsilon, \text{upper}_{i,j-1} - \sigma \}$

$\text{upper}_{i,j} = \max \{ \text{upper}_{i-1,j} - \epsilon, \text{middle}_{i,j-1} - \sigma, 0 \}$

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

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:**

\[
F(i, j) = \max \left\{ F(i-1, j-1) + s(x_i, y_j), \quad \begin{array}{ll}
\max_{k=0 \ldots i-1} F(k, j) - \gamma(i-k) \\
\max_{k=0 \ldots j-1} F(i, k) - \gamma(j-k)
\end{array} \right\}
\]

**Termination:** same

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

**Space:** $O(NM)$
A compromise: affine gaps

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

gap | gap
---|---
gap
open | 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.

\[ F(i, j) : \text{score of alignment } x_1 \ldots x_i \text{ to } y_1 \ldots y_j \text{ if } x_i \text{ aligns to } y_j \]

\[ G(i, j) : \text{score if } x_i, \text{ or } y_j, \text{ aligns to a gap} \]

**Initialization:**

\[ F(i, 0) = d + (i - 1) \times e; \quad F(0, j) = d + (j - 1) \times e \]

**Iteration:**

\[ F(i, j) = \max \left\{ \begin{array}{l} F(i - 1, j - 1) + s(x_i, y_j) \\ G(i - 1, j - 1) + s(x_i, y_j) \end{array} \right. \]

\[ G(i, j) = \max \left\{ \begin{array}{l} F(i - 1, j) - d \\ F(i, j - 1) - d \end{array} \right. \]

**Termination:**

same
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:** \( \# \text{ gaps}(x, y) < k(N) \)  

\[
\begin{align*}
x_i & \quad \text{implies} \quad |i - j| < k(N) \\
y_j
\end{align*}
\]

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

Note that for diagonals, \( i-j = \text{constant} \).
Banded Dynamic Programming

**Initialization:**

\[ F(i,0), \ F(0,j) \text{ undefined for } i, j > k \]

**Iteration:**

For \( i = 1 \ldots M \)

For \( j = \max(1, i - k) \ldots \min(N, i+k) \)

\[ F(i, j) = \max \left\{ F(i - 1, j - 1) + s(x_i, y_j), F(i, j - 1) - d, \text{ if } j > i - k(N) \right\} \]

\[ F(i - 1, j) - d, \text{ if } j < i + k(N) \]

**Termination:**

same

Easy to extend to the affine gap case
**Example global alignment**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **match=2**, **mismatch=-1**, **gap=-1**

**Example**
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
</tbody>
</table>

**ACGCTG**

**match=2**

**mismatch=-1**

**gap=-1**
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

match=2, mismatch=-1, gap=-1

---

CATGT
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**match=2**

**mismatch=-1**

**gap=-1**
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>match=2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>mismatch=-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gap=-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>-1</th>
<th>-2</th>
<th>-3</th>
<th>-4</th>
<th>-5</th>
<th>-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>match=2</td>
<td>mismatch=-1</td>
<td>gap=-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>C</td>
<td>G</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>5</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ACG
-C-
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

match=2, mismatch=-1, gap=-1

ACGC
---C
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>C 1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>A 2</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T 3</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G 4</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T 5</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

match=2
mismatch=-1
gap=-1

ACG
-CA
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>T</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>-4</td>
<td>-1</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>-5</td>
<td>-2</td>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

- **match=2**
- **mismatch=-1**
- **gap=-1**
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>T</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>-4</td>
<td>-1</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>-5</td>
<td>-2</td>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

match=2, mismatch=-1, gap=-1
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

match=2
mismatch=-1
gap=-1

81
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

match=2  
mismatch=-1  
gap=-1  

\[\text{ACGCTG} - C - \text{ATGT}\]
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>-1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Alignment:**

ACGCTG

---

**Scores:**

- **Match:** 2
- **Mismatch:** -1
- **Gap:** -1

**Alignment:** ACGCTG

**Match:**

- A to C: Match (2)
- C to G: Match (2)
- G to C: Match (2)
- C to T: Mismatch (-1)
- T to G: Mismatch (-1)

**Gap:**

- A to C: Gap (-1)
- C to G: Gap (-1)
- G to C: Gap (-1)
- C to T: Gap (-1)
- T to G: Gap (-1)

**Total Score:** 83
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T 3</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G 4</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T 5</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

match=2, mismatch=-1, gap=-1

-ACGCTG
CATG-T-
### Local Alignment Example

match = 1  
mismatch = -1  
gap = -1

\[ y = TAATA \]
\[ x = TACTAA \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example**
Local Alignment Example

\[ y = \text{TAAATA} \]
\[ x = \text{TACTAA} \]

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Local Alignment Example

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A 2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>A 3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T 4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>A 5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

\(y = \text{TAATA-}\)

\(x = \text{TACTAA}\)
Local Alignment Example

\[
\begin{array}{cccccccc}
\text{y} = & \text{TAA} & \text{TAA} & \text{TAA} & \text{TAA} & \text{TAA} & \text{TAA} & \text{TAA} \\
\text{x} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
T & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
A & 2 & 0 & 0 & 2 & 0 & 0 & 2 & 1 \\
A & 3 & 0 & 0 & 1 & 1 & 0 & 1 & 3 \\
T & 4 & 0 & 0 & 0 & 0 & 2 & 0 & 1 \\
A & 5 & 0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\end{array}
\]
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)

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

\[
\begin{array}{cccc}
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
0 & m/2 & m \\
\end{array}
\]

store \textit{prefix}(i) column
Computing Suffix(i)

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

![Diagram showing suffix(i) computation]

store $\text{suffix}(i)$ 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 $length(i)$
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.
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

\[
\text{length}(i)
\]

as the length of the longest path from \((0,0)\) to \((n,m)\) that passes through vertex \((i, m/2)\).
Define \((m, m/2)\) as the vertex where the longest path crosses the middle column.

\[
\text{length}(\text{mid}) = \text{optimal length} = \max_{0 \leq i \leq n} \text{length}(i)
\]
Middle Column of the Alignment

middle column
(midle=#columns/2)
Middle Node of the Alignment

(a node where an optimal alignment path crosses the middle column)
AlignmentPath(source, sink)
find MiddleNode
Divide and Conquer Approach to Sequence Alignment

**AlignmentPath**(source, sink)

find *MiddleNode*

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

The only problem left is how to find this middle node 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
Can We Find the Middle Node without Constructing the Longest Path?

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

4-path that visits the node (4,middle)
In the middle column
Can We Find The Lengths of All $i$-paths?

$\text{length}(i)$: length of an $i$-path:

$\text{length}(0)=2$
$\text{length}(4)=4$
Can We Find The Lengths of All $i$-paths?
Can We Find The Lengths of $i$-paths?

$\text{length}(i) = \text{fromSource}(i) + \text{toSink}(i)$

$\text{length}(i)$: length of an $i$-path
Computing *FromSource* and *toSink*

(fromSource)(i)  

(toSink)(i)
How Much Time Did It Take to Find the Middle Node?

fromSource(i)

toSink(i)
How much time would it take to conquer 2 subproblems?

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

\[ \text{area/4 + area/4 = area/2} \]
Laughable Progress: $O(nm + nm/2)$ Time to Find **THREE** Nodes!

How much time would it take to conquer 4 subproblems?

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

\[
\frac{\text{area}}{8} + \frac{\text{area}}{8} + \frac{\text{area}}{8} + \frac{\text{area}}{8} = \frac{\text{area}}{4}
\]
• $O(nm+nm/2+nm/4)$ Time to Find NEARLY ALL Nodes!

How much time would it take to conquer ALL subproblems?

area + area/2 + area/4 + area/8 + area/16 + .... + < 2 · area
The Middle Edge

Middle Edge: an edge in an optimal alignment path starting at the middle node
Middle Edge in Linear Space Problem. Find a middle edge in the alignment graph in linear space.

- **Input:** Two strings and matrix \( \text{score} \).
- **Output:** A middle edge in the alignment graph of these strings (as defined by the matrix \( \text{score} \)).
Recursive `LinearSpaceAlignment`

```plaintext
LinearSpaceAlignment(top, bottom, left, right)
    if left = right
        return alignment formed by bottom-top edges “↓”
    middle ← ⌊(left+right)/2⌋
    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: \( \text{area} + \text{area}/2 + \text{area}/4 + \text{area}/8 + \text{area}/16 + \ldots \)
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 $n \times n$ grid into blocks of size $t \times t$
- We are comparing two sequences, each of size $n$, and each sequence is sectioned off into chunks, each of length $t$
- Sequence $u = u_1...u_n$ becomes
  $$|u_1...u_t| |u_{t+1}...u_{2t}| ... |u_{n-t+1}...u_n|$$
  and sequence $v = v_1...v_n$ becomes
  $$|v_1...v_t| |v_{t+1}...v_{2t}| ... |v_{n-t+1}...v_n|$$
Partitioning Alignment Grid into Blocks

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

**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 $n \times n$ edit graph partitioned into $t \times t$ subgrids

**Output**: The block alignment of $u$ and $v$ with the maximum score (longest block path through the edit graph)
Let $s_{i,j}$ denote the optimal block alignment score between the first $i$ blocks of $u$ and first $j$ blocks of $v$

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

- $\sigma_{\text{block}}$ is the penalty for inserting or deleting an entire block
- $\beta_{i,j}$ is score of pair of blocks in row $i$ and column $j$.

- To solve: compute alignment score $\beta_{i,j}$ for each pair of blocks $|u_{(i-1)*t+1 \ldots (i\cdot t)}|$ and $|v_{(j-1)*t+1 \ldots (j\cdot t)}|$
- How many blocks are there per sequence? $(n/t)$ blocks of size $t$
- How many pairs of blocks for aligning the two sequences? $(n/t) \times (n/t)$
- For each block pair, solve a mini-alignment problem of size $t \times t$
• Indices $i,j$ range from 0 to $n/t$

• Running time of algorithm is

$$O(\left\lfloor \frac{n}{t} \right\rfloor \times \left\lfloor \frac{n}{t} \right\rfloor) = O(\frac{n^2}{t^2})$$

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 \((n/t) \times (n/t)\) 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 \( t = (\log n)/4 \). Then \( 4^t \times 4^t = n \)

Each sequence has \( t \) nucleotides

\[
\begin{align*}
\text{Lookup table “Score”} \\
\begin{array}{cccc}
\text{AAAAAA} & \text{AAAAAC} & \text{AAAAAG} & \text{AAAAAT} \\
\text{AAAAAC} & & & \\
\text{AAAAAG} & & & \\
\text{AAAAAT} & & & \\
\text{AAAAACA} & & & \\
\vdots & & & \\
\end{array}
\end{align*}
\]

Each sequence has \( t \) nucleotides

\[
s_{i,j} = \max \left\{ s_{i-1,j} - \sigma_{\text{block}}, s_{i,j-1} - \sigma_{\text{block}}, s_{i-1,j-1} - \text{Score}(i^{th} \text{ block of } v, j^{th} \text{ block of } u) \right\}
\]

The new lookup table Score is indexed by a pair of \( t \)-nucleotide strings.
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 $(n/t)\ast(n/t)$ accesses to the lookup table;
Each access takes $O(\log n)$ time. Overall running time:
$O( \left\lceil n^2/t^2 \right\rceil \ast \log n )$; Since $t = \log n$, substitute in: $O( \left\lceil n^2/(\log n)^2 \right\rceil \ast \log n ) \geq O( n^2/\log n )$. 

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

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 \( \frac{n}{t} \) blocks.

Block alignment has \( \left( \frac{n}{t} \right) \times \left( \frac{n}{t} \right) = \frac{n^2}{t^2} \) points of interest.

LCS alignment has \( O\left( \frac{n^2}{t} \right) \) 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 $t \times t$ mini square, compute alignment scores in the last row and column of the minisquare.

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

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

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

we know these scores

we can calculate these scores

$t \times t$ block
Four Russians Speedup

- Build a lookup table for all possible values of the four variables:
  1. all possible scores for the first row $s_{*,j}$
  2. all possible scores for the first column $s_{*,j}$
  3. substring of sequence $u$ in this block (4^t possibilities)
  4. substring of sequence $v$ in this block (4^t possibilities)

- For each quadruple we store the value of the score for the last row and last column.

- This will be a huge table, but we can eliminate alignments scores that don’t make sense: 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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>original encoding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>binary encoding</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Reducing Lookup Table Size

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

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

\[ x = \text{AACT} \]
\[ y = \text{CACT} \]
Central Dogma and Genetic Code

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.

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

The three levels of organization of RNA structure: a) sequence (1D); 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 $j-i+1$. 

```
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.
```
Example: GGGAAAUCC

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

**Initialisation** γ(i, i - 1) = 0 & γ(i, i) = 0

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

γ(i, j) =

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

Where \(\delta(i, j) = 1\) if \(x_i\) and \(x_j\) are a complementary base pair, and \(\delta(i, j) = 0\), otherwise.
Nussinov Folding Algorithm:
After scores for subsequences of length 2

\[
\gamma(i, j) = \max \begin{cases} 
\gamma(i+1, j) \\
\gamma(i, j-1) \\
\gamma(i+1, j-1) + \delta(i, j) \\
\max_{i<k<j}[\gamma(i, k) + \gamma(k+1, j)] 
\end{cases}
\]
Nussinov Folding Algorithm:
After scores for subsequences of length 3

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

\[
\max_{i < k < j} [\gamma(i, k) + \gamma(k + 1, j)]
\]
Nussinov Folding Algorithm
After scores for subsequences of length 4

\[
\gamma(i, j) = \max \left\{ \gamma(i + 1, j), \gamma(i, j - 1), \gamma(i + 1, j - 1) + \delta(i, j) \right\}
\]

Two optimal substructures for same subsequence
Nussinov Folding Algorithm
After scores for subsequences of length 5

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

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>U</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
139
\]
Nussinov Folding Algorithm

After scores for subsequences of length 6

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

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & & \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & & \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & & \\
0 & 0 & 0 & 0 & 0 & 0 & & & \\
0 & 0 & 0 & 0 & 0 & 0 & & & \\
0 & 0 & 0 & 0 & 0 & 0 & & & \\
\end{array}
\]
Nussinov Folding Algorithm
After scores for subsequences of length 7

\[
\gamma(i, j) = \max \left\{ \begin{array}{l}
\gamma(i + 1, j) \\
\gamma(i, j - 1) \\
\gamma(i + 1, j - 1) + \delta(i, j) \end{array} \right. \\
\max_{i \leq k < j} [\gamma(i, k) + \gamma(k + 1, j)]
\]
Nussinov Folding Algorithm
After scores for subsequences of length 8

\[ \gamma(i, j) = \max\left\{ \begin{array}{l} \gamma(i + 1, j) \\ \gamma(i, j - 1) \\ \gamma(i + 1, j - 1) + \delta(i, j) \\ \max_{i<k<j}[\gamma(i, k) + \gamma(k + 1, j)] \end{array} \right\} \]
Nussinov Folding Algorithm
After scores for subsequences of length 9

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

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>U</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[143\]
# Nussinov Folding Algorithm

## Traceback

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>U</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

```
A    A
A    U
G    C
G    C
```

Diagram: A chain of nucleotides with connections showing the folding pattern.
Algorithm: Nussinov RNA folding, fill stage

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

Recursion: starting with all subsequences of length 2, to length \( L \):
\[ \gamma(i, j) = \max \left\{ \gamma(i + 1, j), \gamma(i, j - 1), \gamma(i + 1, j - 1) + \delta(i, j), \max_{i < k < j} [\gamma(i, k) + \gamma(k + 1, j)] \right\}. \]

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

Blue: addition of unpaired base 3 or 7

Green: addition of paired bases 1,7

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

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

Final structure
RECAP

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

1. $i,j$ pair
2. $i$ unpaired
3. $j$ unpaired
4. Bifurcation

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

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 GGAAAUAUCC after initialization, after the recursive fill, and after an optimal structure with three base pairs has been traced back.
Initialisation \( \gamma(i, i-1) = 0 \& \gamma(i, i) = 0 \)

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

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.
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.
Reference for this section

- Chapter 5 Vol 1 (pag 226-291)
- 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

Ancestral Node or ROOT of the Tree

Branches or Lineages

Internal Nodes (fossil)

Terminal Nodes (Living species)

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

species tree by Darwin

((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:

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

No:

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

\[ D_{i,j} = \text{number of differing symbols between } i\text{-th and } j\text{-th rows of a “multiple alignment”}. \]

<table>
<thead>
<tr>
<th>SPECIES</th>
<th>ALIGNMENT</th>
<th>DISTANCE MATRIX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Chimp</td>
</tr>
<tr>
<td>Chimp</td>
<td>ACGTAGGCCT</td>
<td>0</td>
</tr>
<tr>
<td>Human</td>
<td>ATGTAAGACT</td>
<td>3</td>
</tr>
<tr>
<td>Seal</td>
<td>TCGAGAGCAC</td>
<td>6</td>
</tr>
<tr>
<td>Whale</td>
<td>TCGAAAGCAT</td>
<td>4</td>
</tr>
</tbody>
</table>
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”.

<table>
<thead>
<tr>
<th>Species</th>
<th>Alignment</th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>ACGTA GG CCT</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>ATGTA AG ACT</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>TCGAG AG GC AC</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>TCG AA AG GC AT</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Constructing a Distance Matrix

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

<table>
<thead>
<tr>
<th>SPECIES</th>
<th>ALIGNMENT</th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>ACGTAGGCCT</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>ATGTAAGACT</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>TCGAGAGCAC</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>TCGAAAGGCAT</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

How else could we form a distance matrix?
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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Fitting a Tree to a Matrix
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.

\[
\begin{array}{cccc}
  i & j & k & l \\
  i & 0 & 3 & 4 & 3 \\
  j & 3 & 0 & 4 & 5 \\
  k & 4 & 4 & 0 & 2 \\
  l & 3 & 5 & 2 & 0 \\
\end{array}
\]
No Tree Fits a Matrix

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

\[
\begin{array}{cccc}
  & i & j & k & l \\
  i & 0 & 3 & 4 & 3 \\
  j & 3 & 0 & 4 & 5 \\
  k & 4 & 4 & 0 & 2 \\
  l & 3 & 5 & 2 & 0 \\
\end{array}
\]

Additive matrix: distance matrix such that there exists an unrooted tree fitting it.
## More Than One Tree Fits a Matrix

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

More than one tree fits a matrix.
More Than One Tree Fits a Matrix

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

![Phylogenetic Tree]

-Chimp
   - Human
       - Seal
       - Whale

-Chimp
   - Human
       - Seal
       - Whale
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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

How do we compute the unknown distances?
Toward a Recursive Algorithm

\[ d_{k,m} = \left( (d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m}) \right) / 2 \]
Toward a Recursive Algorithm

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

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

\[ d_{k,m} = \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2} \]

\[ \therefore d_{i,m} = \frac{D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j})}{2} \]

\[ d_{i,m} = \frac{(D_{i,k} + D_{i,j} - D_{j,k})}{2} \]
An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

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

\[d_{k,m} = \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2}\]

\[d_{i,m} = D_{i,k} - \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2}\]

\[d_{i,m} = \frac{(D_{i,k} + D_{i,j} - D_{j,k})}{2}\]
## An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
d_{k,m} = \frac{(d_{i,k} + d_{j,k} - d_{i,j})}{2}
\]
\[
d_{k,m} = \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2}
\]
\[
d_{i,m} = D_{i,k} - \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2}
\]
\[
d_{i,m} = \frac{(D_{i,k} + D_{i,j} - D_{j,k})}{2}
\]
An Idea for Distance-Based Phylogeny

\[ d_{k,m} = \frac{(d_{i,k} + d_{j,k} - d_{i,j})}{2} \]
\[ d_{k,m} = \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2} \]
\[ d_{i,m} = D_{i,k} - \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2} \]
\[ d_{Seal,m} = \frac{(D_{Seal,Chimp} + D_{Seal,Whale} - D_{Whale,Chimp})}{2} \]
An Idea for Distance-Based Phylogeny

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

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

The diagram illustrates the relationships between Chimp, Human, Seal, and Whale based on the distances provided in the table. The distances are represented by the lines connecting the species, with the numbers indicating the distance between each pair.
An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$m$</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$m$</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

![Diagram showing the phylogenetic relationships between Chimp, Human, Seal, and Whale.](image)
An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>m</td>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

![Phylogenetic tree diagram](image)
An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>m</td>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

```
Chimp   | Human | m   |
--------|-------|-----|
Chimp   | 0     | 3   | 4   |
Human   | 3     | 0   | 5   |
m       | 4     | 5   | 0   |
```
An Idea for Distance-Based Phylogeny

\[ d_{k,m} = \frac{(d_{i,k} + d_{j,k} - d_{i,j})}{2} \]
\[ d_{k,m} = \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2} \]
\[ d_{i,m} = D_{i,k} - \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2} \]
\[ d_{\text{Chimp},a} = \frac{(D_{\text{Chimp},m} + D_{\text{Chimp,Human}} - D_{\text{Human},m})}{2} \]
An Idea for Distance-Based Phylogeny

\[
\begin{array}{|c|c|c|}
\hline
 & \text{Chimp} & \text{Human} \\ \hline
\text{Chimp} & 0 & 3 \\ \text{Human} & 3 & 0 \\ m & 4 & 5 \\ \hline
\end{array}
\]

\[
d_{k,m} = \frac{d_{i,k} + d_{j,k} - d_{i,j}}{2}
\]

\[
d_{k,m} = \frac{D_{i,k} + D_{j,k} - D_{i,j}}{2}
\]

\[
d_{i,m} = D_{i,k} - \frac{D_{i,k} + D_{j,k} - D_{i,j}}{2}
\]

\[
d_{\text{Chimp,}a} = 1
\]
# An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>M</td>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

![Diagram](image_url)
An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>m</td>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

![Distance-based phylogeny diagram]
An Idea for Distance-Based Phylogeny

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
**Exercise Break:** Apply this recursive approach to the distance matrix below.

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>13</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>j</td>
<td>13</td>
<td>0</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>k</td>
<td>21</td>
<td>12</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>l</td>
<td>22</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>
What Was Wrong With Our Algorithm?

\[
\begin{array}{cccc}
 i & j & k & l \\
 i & 0 & 13 & 21 & 22 \\
 j & 13 & 0 & 12 & 13 \\
 k & 21 & 12 & 0 & 13 \\
 l & 22 & 13 & 13 & 0 \\
\end{array}
\]

![Graph Diagram]
What Was Wrong With Our Algorithm?

minimum element is $D_{j,k}$
What Was Wrong With Our Algorithm?

The minimum element is $D_{j,k}$.

$j$ and $k$ are not neighbors!
Rather than trying to find neighbors, let’s instead try to compute the length of limbs, the edges attached to leaves.
\[d_{k,m} = \frac{[(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})]}{2}\]
\[d_{k,m} = \frac{(d_{i,k} + d_{j,k} - d_{i,j})}{2}\]
\[d_{k,m} = \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2}\]
\[d_{i,m} = D_{i,k} - \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2}\]
\[d_{i,m} = \frac{(D_{i,k} + D_{i,j} - D_{j,k})}{2}\]
From Neighbors to Limbs

Assumes that \( i \) and \( j \) are neighbors...
Computing Limb Lengths

Limb Length Theorem: \( \text{LimbLength}(i) \) 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, \( \text{LimbLength}(j) \).

**Code Challenge:** Solve the Limb Length Problem.
Computing Limb Lengths

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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
(D_{\text{chimp, human}} + D_{\text{chimp, seal}} - D_{\text{human, seal}}) / 2 = (3 + 6 - 7) / 2 = 1
\]
Computing Limb Lengths

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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\left( D_{\text{chimp}, \text{human}} + D_{\text{chimp}, \text{seal}} - D_{\text{human}, \text{seal}} \right) / 2 = (3 + 6 - 7) / 2 = 1
\[
\left( D_{\text{chimp}, \text{human}} + D_{\text{chimp}, \text{whale}} - D_{\text{human}, \text{whale}} \right) / 2 = (3 + 4 - 5) / 2 = 1
\]
Computing Limb Lengths

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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

\[
(D_{\text{whale, chimp}} + D_{\text{chimp, seal}} - D_{\text{whale, seal}}) / 2 = (6 + 4 - 2) / 2 = 4
\]
Computing Limb Lengths

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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
AdditivePhylogeny In Action

$$D$$

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>13</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>j</td>
<td>13</td>
<td>0</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>k</td>
<td>21</td>
<td>12</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>l</td>
<td>22</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>

$$\text{TREE}(D)$$
AdditivePhylogeny In Action

1. Pick an arbitrary leaf $j$. 

$$D$$

<table>
<thead>
<tr>
<th></th>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
<th>$l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0</td>
<td>13</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>$j$</td>
<td>13</td>
<td>0</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>$k$</td>
<td>21</td>
<td>12</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>$l$</td>
<td>22</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>
## Additive Phylogeny In Action

<table>
<thead>
<tr>
<th></th>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
<th>$l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0</td>
<td>13</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>$j$</td>
<td>13</td>
<td>0</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>$k$</td>
<td>21</td>
<td>12</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>$l$</td>
<td>22</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>

$LimbLength(j) = 2$

2. Compute its limb length, $LimbLength(j)$. 
3. Subtract $\text{LimbLength}(j)$ from each row and column to produce $D^{\text{bald}}$ in which $j$ is a $\text{bald limb}$ (length 0).
Additive Phylogeny In Action

4. Remove the $j$-th row and column of the matrix to form the $(n - 1) \times (n - 1)$ matrix $D_{\text{trim}}$. 

\[
\begin{array}{cccc}
i & j & k & l \\
i & 0 & 11 & 21 & 22 \\
j & 11 & 0 & 10 & 11 \\
k & 21 & 10 & 0 & 13 \\
l & 22 & 11 & 13 & 0 \\
\end{array}
\]
5. Construct $Tree(D^{trim})$. 

Additive Phylogeny In Action

$D^{trim}$

<table>
<thead>
<tr>
<th></th>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
<th>$l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0</td>
<td>11</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>$j$</td>
<td>11</td>
<td>0</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$k$</td>
<td>21</td>
<td>10</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>$l$</td>
<td>22</td>
<td>11</td>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>
6. Identify the point in $Tree(D_{\text{trim}})$ where leaf $j$ should be attached.
AdditivePhylogeny In Action

\[
D = \begin{pmatrix}
i & j & k & l \\
i & 0 & 13 & 21 & 22 \\
j & 13 & 0 & 12 & 13 \\
k & 21 & 12 & 0 & 13 \\
l & 22 & 13 & 13 & 0 \\
\end{pmatrix}
\]

\[\text{LimbLength}(j) = 2\]

7. Attach \(j\) by an edge of length \(\text{LimbLength}(j)\) in order to form \(\text{Tree}(D)\).
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 $(n - 1) \times (n - 1)$ 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, $\text{LimbLength}(j)$.
3. Subtract $\text{LimbLength}(j)$ 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 $(n–1) \times (n–1)$ matrix $D^\text{trim}$.
5. Construct $\text{Tree}(D^\text{trim})$.
6. **Identify the point in $\text{Tree}(D^\text{trim})$ where leaf $j$ should be attached.**
7. Attach $j$ by an edge of length $\text{LimbLength}(j)$ in order to form $\text{Tree}(D)$. 
# Attaching a Limb

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>11</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>j</td>
<td>11</td>
<td>0</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>k</td>
<td>21</td>
<td>10</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>l</td>
<td>22</td>
<td>11</td>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\text{TREE}(D_{\text{trim}})
\]

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

The Limb Length Theorem states that the length of the limb of $j$ is equal to the minimum value of $(D_{bald}^{i,j} + D_{bald}^{j,k} - D_{bald}^{i,k})/2$ over all leaves $i$ and $k$. This is mathematically represented as:

$$(D_{bald}^{i,j} + D_{bald}^{j,k} - D_{bald}^{i,k})/2 = 0$$
Attaching a Limb

\[
\begin{array}{c|cccc}
 & i & j & k & l \\
\hline
i & 0 & 11 & 21 & 22 \\
\hline
j & 11 & 0 & 10 & 11 \\
\hline
k & 21 & 10 & 0 & 13 \\
\hline
l & 22 & 11 & 13 & 0 \\
\end{array}
\]

\[
D_{bald}^{i,j} + D_{bald}^{j,k} - D_{bald}^{i,k} = 0
\]

\[
D_{bald}^{i,j} + D_{bald}^{j,k} = D_{bald}^{i,k}
\]
# Attaching a Limb

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

$$D_{i,j}^{\text{bald}} + D_{j,k}^{\text{bald}} = D_{i,k}^{\text{bald}}$$
**AdditivePhylogeny**

**AdditivePhylogeny**(\(D\)):
1. Pick an arbitrary leaf \(j\).
2. Compute its limb length, \(\text{LimbLength}(j)\).
3. Subtract \(\text{LimbLength}(j)\) 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 \((n - 1) \times (n - 1)\) matrix \(D^{\text{trim}}\).
5. Construct \(\text{Tree}(D^{\text{trim}})\).
6. Identify the point in \(\text{Tree}(D^{\text{trim}})\) where leaf \(j\) should be attached.
7. Attach \(j\) by an edge of length \(\text{LimbLength}(j)\) in order to form \(\text{Tree}(D)\).

**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.
Sum of Squared Errors

\[ \text{Discrepancy}(T, D) = \sum_{1 \leq i < j \leq n} (d_{i,j}(T) - D_{i,j})^2 \]
\[ = 1^2 + 1^2 = 2 \]
Sum of Squared Errors

**Exercise Break:** Assign lengths to edges in $T$ in order to minimise $\text{Discrepancy}(T, D)$.

$$
\begin{array}{cccc}
\text{i} & \text{j} & \text{k} & \text{l} \\
	ext{i} & 0 & 3 & 4 & 3 \\
\text{j} & 3 & 0 & 4 & 5 \\
\text{k} & 4 & 4 & 0 & 2 \\
\text{l} & 3 & 5 & 2 & 0 \\
\end{array}

\begin{array}{cccc}
\text{i} & \text{j} & \text{k} & \text{l} \\
\text{i} & 0 & ? & ? & ? \\
\text{j} & ? & 0 & ? & ? \\
\text{k} & ? & ? & 0 & ? \\
\text{l} & ? & ? & ? & 0 \\
\end{array}
$$
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 $n \times n$ distance matrix $D$.
- **Output:** A weighted tree $T$ with $n$ leaves minimizing $\text{Discrepancy}(T, D)$ 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).

![Diagram of Ultrametric Trees](image-url)
UPGMA: A Clustering Heuristic

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

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>k</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
2. Find the two closest clusters $C_1$ and $C_2$ according to the average distance

$$D_{\text{avg}}(C_1, C_2) = \sum_{i \in C_1, j \in C_2} D_{i,j} / |C_1| \cdot |C_2|$$

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

\begin{tabular}{cccc}
  $i$ & $j$ & $k$ & $l$ \\
  $i$ & 0 & 3 & 4 & 3 \\
  $j$ & 3 & 0 & 4 & 5 \\
  $k$ & 4 & 4 & 0 & 2 \\
  $l$ & 3 & 5 & 2 & 0 \\
\end{tabular}
### UPGMA: A Clustering Heuristic

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

<table>
<thead>
<tr>
<th></th>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
<th>$l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$j$</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$k$</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$l$</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

$\{k, l\}$

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

![Matrix and tree diagram]

<table>
<thead>
<tr>
<th></th>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
<th>$l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$j$</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$k$</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$l$</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
5. Update the distance matrix by computing the average distance between each pair of clusters.

**UPGMA: A Clustering Heuristic**

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>{k, l}</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>j</td>
<td>3</td>
<td>0</td>
<td>4.5</td>
</tr>
<tr>
<td>{k, l}</td>
<td>3.5</td>
<td>4.5</td>
<td>0</td>
</tr>
</tbody>
</table>

![Distance Matrix and Cluster Diagram]
6. Iterate until a single cluster contains all species.

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>{k, l}</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>j</td>
<td>3</td>
<td>0</td>
<td>4.5</td>
</tr>
<tr>
<td>{k, l}</td>
<td>3.5</td>
<td>4.5</td>
<td>0</td>
</tr>
</tbody>
</table>
UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.

\[
\begin{array}{c|cc}
\{i, j\} & \{k, l\} \\
\{i, j\} & 0 & 4 \\
\{k, l\} & 4 & 0 \\
\end{array}
\]

\[
\begin{array}{c}
\{i, j\} \\
\{k, l\} \\
\end{array}
\]

\[
\begin{array}{c}
1.5 \\
1.5 \\
1 \\
1 \\
0 \\
0 \\
0 \\
0 \\
\end{array}
\]
UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.
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

$$D_{\text{avg}}(C_1, C_2) = \sum_{i \in C_1, j \in C_2} D_{i,j} / |C_1| \cdot |C_2|$$

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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>k</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
UPGMA Doesn’t “Fit” a Tree to a Matrix

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>k</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

![Diagram](image)
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 $n \times n$ distance matrix $D$, its neighbor-joining matrix is the matrix $D^*$ defined as

$$D_{i,j}^* = (n - 2) \cdot D_{i,j} - \text{TotalDistance}_D(i) - \text{TotalDistance}_D(j)$$

where $\text{TotalDistance}_D(i)$ is the sum of distances from $i$ to all other leaves.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>TotalDistance$_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0</td>
<td>13</td>
<td>21</td>
<td>22</td>
<td>56</td>
</tr>
<tr>
<td>$j$</td>
<td>13</td>
<td>0</td>
<td>12</td>
<td>13</td>
<td>38</td>
</tr>
<tr>
<td>$k$</td>
<td>21</td>
<td>12</td>
<td>0</td>
<td>13</td>
<td>46</td>
</tr>
<tr>
<td>$l$</td>
<td>22</td>
<td>13</td>
<td>13</td>
<td>0</td>
<td>48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
<th>$l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0</td>
<td>-68</td>
<td>-60</td>
<td>-60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$j$</td>
<td>-68</td>
<td>0</td>
<td>-60</td>
<td>-60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k$</td>
<td>-60</td>
<td>-60</td>
<td>0</td>
<td>-68</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$l$</td>
<td>-60</td>
<td>-60</td>
<td>-68</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Neighbor-Joining Theorem: If $D$ is additive, then the smallest element of $D^*$ corresponds to neighboring leaves in $\text{Tree}(D)$.
# Neighbor-Joining in Action

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

```
<table>
<thead>
<tr>
<th></th>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
<th>$l$</th>
<th>TotalDistance$_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0</td>
<td>-68</td>
<td>-60</td>
<td>-60</td>
<td>56</td>
</tr>
<tr>
<td>$j$</td>
<td>-68</td>
<td>0</td>
<td>-60</td>
<td>-60</td>
<td>38</td>
</tr>
<tr>
<td>$k$</td>
<td>-60</td>
<td>-60</td>
<td>0</td>
<td>-68</td>
<td>46</td>
</tr>
<tr>
<td>$l$</td>
<td>-60</td>
<td>-60</td>
<td>-68</td>
<td>0</td>
<td>48</td>
</tr>
</tbody>
</table>
```
## Neighbor-Joining in Action

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
<th>TotalDistance$_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>-68</td>
<td>-60</td>
<td>-60</td>
<td>56</td>
</tr>
<tr>
<td>j</td>
<td>-68</td>
<td>0</td>
<td>-60</td>
<td>-60</td>
<td>38</td>
</tr>
<tr>
<td>k</td>
<td>-60</td>
<td>-60</td>
<td>0</td>
<td>-68</td>
<td>46</td>
</tr>
<tr>
<td>l</td>
<td>-60</td>
<td>-60</td>
<td>-68</td>
<td>0</td>
<td>48</td>
</tr>
</tbody>
</table>

2. Find a minimum element $D^*_{i,j}$ of $D^*$. 
### Neighbor-Joining in Action

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
<th>TotalDistance&lt;sub&gt;D&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>-68</td>
<td>-60</td>
<td>-60</td>
<td>56</td>
</tr>
<tr>
<td>D*</td>
<td>j</td>
<td>-68</td>
<td>0</td>
<td>-60</td>
<td>38</td>
</tr>
<tr>
<td>k</td>
<td>-60</td>
<td>-60</td>
<td>0</td>
<td>-68</td>
<td>46</td>
</tr>
<tr>
<td>l</td>
<td>-60</td>
<td>-60</td>
<td>-68</td>
<td>0</td>
<td>48</td>
</tr>
</tbody>
</table>

2. Find a minimum element $D^*_{i,j}$ of $D^*$. 
3. Compute $\Delta_{i,j} = \frac{(\text{TotalDistance}_{D}(i) - \text{TotalDistance}_{D}(j))}{(n - 2)}$. 

\[ \Delta_{i,j} = \frac{56 - 38}{4 - 2} = 9 \]
## Neighbor-Joining in Action

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
<th>TotalDistance&lt;sub&gt;D&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>13</td>
<td>21</td>
<td>22</td>
<td>56</td>
</tr>
<tr>
<td>j</td>
<td>13</td>
<td>0</td>
<td>12</td>
<td>13</td>
<td>38</td>
</tr>
<tr>
<td>k</td>
<td>21</td>
<td>12</td>
<td>0</td>
<td>13</td>
<td>46</td>
</tr>
<tr>
<td>l</td>
<td>22</td>
<td>13</td>
<td>13</td>
<td>0</td>
<td>48</td>
</tr>
</tbody>
</table>

\[
\Delta_{i,j} = \frac{(56 - 38)}{(4 - 2)} = 9
\]

\[
\text{LimbLength}(i) = \frac{1}{2}(13 + 9) = 11
\]

\[
\text{LimbLength}(i) = \frac{1}{2}(13 - 9) = 2
\]

4. Set \(\text{LimbLength}(i)\) equal to \(\frac{1}{2}(D_{i,j} + \Delta_{i,j})\) and \(\text{LimbLength}(j)\) equal to \(\frac{1}{2}(D_{i,j} - \Delta_{i,j})\).
Neighbor-Joining in Action

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>TotalDistance$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>0</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$k$</td>
<td>10</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>$l$</td>
<td>11</td>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>
Flashback: Computation of $d_{k,m}$

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

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

\[ d_{k,m} = \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2} \]
6. Apply **NeighborJoining** to $D'$ to obtain $Tree(D')$. 

$$
\begin{array}{ccc}
  m & k & l \\
  m & 0 & 10 & 11 \\
  k & 10 & 0 & 13 \\
  l & 11 & 13 & 0 \\
\end{array}
$$
7. Reattach limbs of \( i \) and \( j \) to obtain \( \text{Tree}(D) \).

\[
\begin{array}{ccc}
  m & k & l \\
  m & 0 & 10 & 11 \\
  k & 10 & 0 & 13 \\
  l & 11 & 13 & 0 \\
\end{array}
\]

\[
\text{LimbLength}(i) = \frac{1}{2}(13 + 9) = 11 \\
\text{LimbLength}(i) = \frac{1}{2}(13 - 9) = 2
\]
### Neighbor-Joining in Action

#### Step 7
Reattach limbs of $i$ and $j$ to obtain $\text{Tree}(D)$.

<table>
<thead>
<tr>
<th></th>
<th>$m$</th>
<th>$k$</th>
<th>$l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>0</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$k$</td>
<td>10</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>$l$</td>
<td>11</td>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>

![Tree](image)

$\text{Tree}(D)$
Neighbor-Joining

NeighborJoining(D):
1. Construct neighbor-joining matrix $D^*$ from $D$.
2. Find a minimum element $D^*_{i,j}$ of $D^*$.
3. Compute $\Delta_{i,j} = (\text{TotalDistance}_D(i) - \text{TotalDistance}_D(j)) / (n - 2)$.
4. Set $\text{LimbLength}(i)$ equal to $\frac{1}{2}(D_{i,j} + \Delta_{i,j})$ and $\text{LimbLength}(j)$ 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 $\text{Tree}(D')$.
7. Reattach limbs of $i$ and $j$ to obtain $\text{Tree}(D)$.

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 \texttt{NeighborJoining}. 
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?

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>k</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

![UPGMA tree](image)
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...

<table>
<thead>
<tr>
<th>Species</th>
<th>Alignment</th>
<th>Distance Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>ACGTAGGCCT</td>
<td>0  3  6  4</td>
</tr>
<tr>
<td>Human</td>
<td>ATGTAAGACT</td>
<td>3  0  7  5</td>
</tr>
<tr>
<td>Seal</td>
<td>TCGAGAGCAC</td>
<td>6  7  0  2</td>
</tr>
<tr>
<td>Whale</td>
<td>TCGAAAGCAT</td>
<td>4  5  2  0</td>
</tr>
</tbody>
</table>
Example and RECAP (note different notation)

<table>
<thead>
<tr>
<th>Distance matrix</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>10</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>8</td>
<td>11</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>U₁</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>U₂</th>
<th>U₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>U₂</td>
<td>3</td>
</tr>
<tr>
<td>U₃</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>U₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
</tr>
</tbody>
</table>

Step 1
S calculations

\[
S_a = \frac{(5+4+7+6+6+8)}{6} = 7.5
\]

\[
S_b = \frac{(5+7+10+9+11+4)}{6} = 10.5
\]

\[
S_c = \frac{(4+7+7+6+6+8)}{6} = 8
\]

\[
S_d = \frac{(7+10+7+5+9+4)}{6} = 9.5
\]

\[
S_e = \frac{(6+9+6+5+8)}{6} = 8.5
\]

\[
S_f = \frac{(8+11+8+9+8)}{6} = 11
\]

Step 2
Calculate pair with smallest \( M_{ij} \) where \( M_{ij} = D_{ij} - S_i - S_j \).

Smallest is \( M_{AB} = 5 - 7.5 - 10.5 = -13 \)

Smallest is \( M_{DE} = 5 - 9.5 - 8.5 = -13 \)

Choose one of these (AB here).

Step 3
Create a node (U) that joins pair with lowest \( M_{ij} \) such that \( S_{ij} = D_{ij} / 2 + (S_i - S_j)/2 \).

\[
S_{U_{12}} = D_{A2} / 2 + (S_A - S_2)/2 = 1
\]

\[
S_{U_{13}} = D_{A3} / 2 + (S_A - S_3)/2 = 4
\]

Step 4
Join i and j according to S above and make all other taxa in form of a star. Branches in black are of unknown length. Branches in red are of known length.

Step 5
Calculate new distance matrix of all other taxa to U with \( D_{U1} = D_{i1} + D_{k1} - D_{ij} \), where i and j are those selected from above.

http://evolution-textbook.org/content/free/tables/Ch_27/T11_EVOW_Ch27.pdf
QuickTree: building huge Neighbor-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: kih@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.
Here we do not use a distance matrix and we value each column of the alignment; each column could output a tree.
Parsimony score: sum of Hamming distances along each edge.
Toward a Computational Problem

**Parsimony score:** sum of Hamming distances along each edge.

Parsimony Score: 8

- **Chimp:** ACGTAAGCCT
- **Human:** ATGTAAGACT
- **Seal:** TCGAGAGCAC
- **Whale:** TCGAAAGCAT
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 minimizes 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 $k$. 
Exercise Break: Prove the following recurrence relation:

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

For symbols $i$ and $j$, define

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

\[ s_k(v) = \min_{\text{all symbols } i} \{s_i(Daughter(v)) + \delta_{i,k}\} + \min_{\text{all symbols } i} \{s_i(Son(v)) + \delta_{i,k}\} \]
A Dynamic Programming Algorithm

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

Parsimony Score: 8
Finding the Most Parsimonious Tree

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.
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 ($W, X, Y, Z$).
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**.
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

(a) Sample

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GAGG</td>
<td>GAGG</td>
<td>ACC</td>
<td>CCT</td>
<td>CTA</td>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>GCGT</td>
<td>GGGG</td>
<td>AAG</td>
<td>CCC</td>
<td>GAG</td>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CAGG</td>
<td>AGAG</td>
<td>AAG</td>
<td>AAG</td>
<td>AGA</td>
<td>AGA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CAAA</td>
<td>AGAG</td>
<td>AGA</td>
<td>AGA</td>
<td>AGA</td>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GCCG</td>
<td>CAGG</td>
<td>AAG</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Inferred tree

(b) Subhypothesis 1

(b) Subhypothesis 2

Pseudosample 1

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GCGG</td>
<td>GAGG</td>
<td>AAG</td>
<td>ACC</td>
<td>CTT</td>
<td>CAA</td>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>GCGG</td>
<td>CCGG</td>
<td>AGG</td>
<td>GAG</td>
<td>GAG</td>
<td>GAG</td>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CCCC</td>
<td>CAAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CAAA</td>
<td>AGAG</td>
<td>CAA</td>
<td>CAA</td>
<td>CAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GCCG</td>
<td>CAGG</td>
<td>AAA</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bootstrapped trees

Pseudosample 2

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CCCC</td>
<td>CAAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GCCG</td>
<td>CAGG</td>
<td>AAA</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pseudosample n

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GGGG</td>
<td>GAGG</td>
<td>AAG</td>
<td>CCC</td>
<td>CCA</td>
<td>CAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>GGGG</td>
<td>CCGG</td>
<td>AGG</td>
<td>GAG</td>
<td>GAG</td>
<td>GAG</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CCCC</td>
<td>CAAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CAAA</td>
<td>AGAG</td>
<td>CAA</td>
<td>CAA</td>
<td>CAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GCCG</td>
<td>CAGG</td>
<td>AAA</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td>ATT</td>
<td></td>
</tr>
</tbody>
</table>

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 alignment problem: Generalising Pairwise to Multiple Alignment

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

```
AT  -  GCG  -
A   -  CGT  -  A
ATC  A  C  -  A
```

• Our scoring function should score alignments with conserved columns higher.
Alignments = Paths in 3-D

- Alignment of ATGC, AATC, and ATGC

#symbols up to a given position

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>--</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>A</td>
<td>T</td>
<td>--</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>
Alignments = Paths in 3-D

- Alignment of ATGC, AATC, and ATGC

\[(0,0,0) \rightarrow (1,1,0) \rightarrow (1,2,1) \rightarrow (2,3,2) \rightarrow (3,3,3) \rightarrow (4,4,4)\]
2-D Alignment Cell versus 3-D Alignment Cell
Multiple Alignment: Dynamic Programming

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

• \( \delta(x, y, z) \) 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:

\[ AC - GC G G - C \]
\[ AC - G C - G A G \]
\[ G C C G C - G A G \]

\[ A C G C G G - C \quad A C - G C G G - C \quad A C - G C G A G \]
\[ A C G C - G A C \quad G C C G C - G A G \quad G C C G C G A G \]
Idea: Construct Multiple from Pairwise Alignments

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

AAAATTTT----
----TTTTGGGG

AAAAATTTT----
TTTTGGGG----

GGGGAAAA----
----GGGGAAAA
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)
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

Distance matrix

Painwise alignments

Guide tree

Software: muscle, MAFFT
Challenges

Very little information

Identification of paralogous, orthologous, xenologous events
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

<table>
<thead>
<tr>
<th>Genes</th>
<th>eij = expression level of gene i at checkpoint j</th>
</tr>
</thead>
<tbody>
<tr>
<td>YLR361C</td>
<td>0.14 0.03 -0.06 0.07 -0.01 -0.06 -0.01</td>
</tr>
<tr>
<td>YMR290C</td>
<td>0.12 -0.23 -0.24 -1.16 -1.40 -2.67 -3.00</td>
</tr>
<tr>
<td>YNR065C</td>
<td>-0.10 -0.14 -0.03 -0.06 -0.07 -0.14 -0.04</td>
</tr>
<tr>
<td>YGR043C</td>
<td>-0.43 -0.73 -0.06 -0.11 -0.16 3.47 2.64</td>
</tr>
<tr>
<td>YLR258W</td>
<td>0.11 0.43 0.45 1.89 2.00 3.32 2.56</td>
</tr>
<tr>
<td>YPL012W</td>
<td>0.09 -0.28 -0.15 -1.18 -1.59 -2.96 -3.08</td>
</tr>
<tr>
<td>YNL141W</td>
<td>-0.16 -0.04 -0.07 -1.26 -1.20 -2.82 -3.13</td>
</tr>
<tr>
<td>YJL028W</td>
<td>-0.28 -0.23 -0.19 -0.19 -0.32 -0.18 -0.18</td>
</tr>
<tr>
<td>YKL026C</td>
<td>-0.19 -0.15 0.03 0.27 0.54 3.64 2.74</td>
</tr>
<tr>
<td>YPR055W</td>
<td>0.15 0.15 0.17 0.09 0.07 0.09 0.07</td>
</tr>
</tbody>
</table>

### Gene expression vector

- **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
Biologists need algorithms to find similar behaving genes

### Genes as Points in Multidimensional Space

A gene expression vector can be thought of as a point in an $m$-dimensional space, where each dimension represents a different gene's expression level. The expression levels of different genes are represented by the coordinates of these points. The data for 10 genes is shown below:

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

#### Good Clustering Principle

To identify groups of genes with similar expression, we can think of an expression vector of length $m$ as a point in $m$-dimensional space. Genes with similar expression profiles will correspond to nearby points. Ideally, clusters should satisfy the following common-sense principle, which is illustrated in Figure 1.5:

**Good Clustering Principle:**

Every pair of points from the same cluster should be closer to each other than any pair of points from different clusters.

We have therefore embedded gene expression analysis within the algorithmic problem of partitioning a collection of $n$ points in $m$-dimensional space into $k$ clusters, which might not always be possible.
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 $< \Delta$
- distance between elements in different clusters $> \Delta$
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:

\[ d(\text{DataPoint}, \text{Centers}) = \min_{\text{all points } x \text{ from Centres}} d(\text{DataPoint}, x) \]
Distance from \textit{Data} to \textit{Centers}

\[
\text{MaxDistance}(\text{Data, Centers}) = \max \text{ all points}\ \text{DataPoint}\ \text{from}\ \text{Data} \quad d(\text{DataPoint, Centers})
\]
**k-Center Clustering Problem.** Given a set of points *Data*, find *k* centers minimising $\text{MaxDistance}(\text{Data}, \text{Centers})$.

- **Input:** A set of points *Data* and an integer *k*.
- **Output:** A set of *k* points *Centers* that minimises $\text{MaxDistance}(\text{Data Points}, \text{Centers})$ over all possible choices of *Centers*.
**k-Center Clustering Problem.** Given a set of points \( \text{Data} \), find \( k \) centers minimising \( \text{MaxDistance}(\text{Data, Centers}) \).

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

\[ \text{Centers} \leftarrow \text{the set consisting of a single } DataPoint \text{ from } Data \]

\[ \text{while } \text{Centers} \text{ have fewer than } k \text{ points} \]

\[ \text{DataPoint} \leftarrow \text{a point in } Data \text{ maximising } d(DataPoint, \text{Centers}) \]

\[ \text{among all data points} \]

\[ \text{add } DataPoint \text{ to } \text{Centers} \]
What Is Wrong with \textbf{FarthestFirstTraversal}?

\textbf{FarthestFirstTraversal} selects \textit{Centers} that minimise $\text{MaxDistance}(\text{Data}, \text{Centers})$.

But biologists are interested in \textbf{typical} rather than \textbf{maximum} deviations, since maximum deviations may represent \textbf{outliers} (experimental errors).
Modifying the Objective Function

The **maximal distance** between \textit{Data} and \textit{Centers}:

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

The **squared error distortion** between \textit{Data} and \textit{Centers}:

\[
\text{Distortion}(\text{Data}, \text{Centers}) = \sum_{\text{DataPoint from Data}} \frac{d(\text{DataPoint}, \text{Centers})^2}{n}
\]

A **single** data point contributes to \textit{MaxDistance}

All data points contribute to \textit{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 \( \text{MaxDistance}(\text{DataPoints, Centers}) \) 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 \( \text{Distortion}(\text{Data, Centers}) \) over all choices of *Centers*.

\[ NP\text{-Hard for } k > 1 \]
**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

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

Centers

again!

new centers ⇐ clusters’ centers of gravity

new centers
The Lloyd Algorithm in Action

Clusters

Centers

again!

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).
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 $r_{\text{blue}}$ and $r_{\text{red}}$ ($r_{\text{blue}} + r_{\text{red}} = 1$).
Flipping One Biased Coin

• We flip a loaded coin with an unknown bias $\theta$ (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

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

This expression is maximised at $\theta = \frac{i}{n}$ (most likely bias)
Flipping Two Biased Coins

Data

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>H T T T H T T H T H</td>
<td>0.4</td>
</tr>
<tr>
<td>H H H H H T H H H H H</td>
<td>0.9</td>
</tr>
<tr>
<td>H T H H N H H H T H H</td>
<td>0.8</td>
</tr>
<tr>
<td>H T T T T T T H H T T</td>
<td>0.3</td>
</tr>
<tr>
<td>T H H H T H N H H T H</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Goal: estimate the probabilities $\theta_A$ and $\theta_B$
If We Knew Which Coin Was Used in Each Sequence...

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHHHTTTHTTH</td>
<td>0.4 1</td>
</tr>
<tr>
<td>HHHHTHHHHH</td>
<td>0.9 0</td>
</tr>
<tr>
<td>HTHHHHHHTHH</td>
<td>0.8 0</td>
</tr>
<tr>
<td>HHHHTTHHTTT</td>
<td>0.3 1</td>
</tr>
<tr>
<td>TTHHTHHHTTH</td>
<td>0.7 0</td>
</tr>
</tbody>
</table>

**Goal:** estimate \( Parameters = (\theta_A, \theta_B) \) when \( HiddenVector \) is given
If We Knew Which Coin Was Used in Each Sequence...

\[
\theta_A = \text{fraction of heads generated in all flips with coin } A = \frac{4+3}{10+10} = \frac{0.4+0.3}{2} = 0.35
\]

\[
\theta_B = \text{fraction of heads generated in all flips with coin } B = \frac{9+8+7}{10+10+10} = \frac{0.9+0.8+0.7}{1+1+1} = 0.80
\]

\[
\begin{array}{ccc}
\text{Data} & \text{Hidden Vector} \\
\text{HHHHHTHHHHH} & 0.9 & 0 \\
\text{HHTHHHHHTHH} & 0.8 & 0 \\
\text{HTTTTTTHTT} & 0.3 & 1 \\
\text{THTHHTHHHTH} & 0.7 & 0 \\
\end{array}
\]
**Parameters as a Dot-Product**

\[ Data \times HiddenVector \quad Parameters = (\theta_A, \theta_B) \]

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHHHTTHH</td>
<td>0.4 * 1</td>
<td>(0.35, 0.80)</td>
</tr>
<tr>
<td>HHHHHTHHH</td>
<td>0.9 * 0</td>
<td></td>
</tr>
<tr>
<td>HTHHHHHTHH</td>
<td>0.8 * 0</td>
<td></td>
</tr>
<tr>
<td>HHHHHTTHHTT</td>
<td>0.3 * 1</td>
<td></td>
</tr>
<tr>
<td>THHHTHHHTTH</td>
<td>0.7 * 0</td>
<td></td>
</tr>
</tbody>
</table>

\[ \theta_A = \text{fraction of heads generated in all flips with coin } A = \frac{4+3}{10+10} = \frac{0.4+0.3}{2} = 0.35 \]

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

\[ \sum_{\text{all data points}} Data_i \times HiddenVector_i / \sum_{\text{all data points}} HiddenVector_i = 0.35 \]

\[ Data \times HiddenVector \quad (1,1,\ldots,1) \times HiddenVector = 0.35 \]

1 refers to a vector (1,1, … ,1) consisting of all 1's
**Parameters** as a Dot-Product

\[
\text{Data} \times \text{HiddenVector} = (\theta_A, \theta_B)
\]

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHHHHTHHHHH</td>
<td>0.9 * 0</td>
<td>(0.35, 0.80)</td>
</tr>
<tr>
<td>HTHHHHHTHHH</td>
<td>0.8 * 0</td>
<td></td>
</tr>
<tr>
<td>HHHHTTHHTTH</td>
<td>0.4 * 1</td>
<td></td>
</tr>
<tr>
<td>HHHHTTHHTHT</td>
<td>0.3 * 1</td>
<td></td>
</tr>
<tr>
<td>HHHHHTHHHTH</td>
<td>0.7 * 0</td>
<td></td>
</tr>
</tbody>
</table>

\[\theta_B = \text{fraction of heads generated in all flips with coin } B\]
\[= \frac{9+8+7}{10+10+10} = \frac{0.9+0.8+0.7}{1+1+1} = 0.80\]
\[= \frac{0.5*0+0.9*1+0.8*1+0.4*0+0.7*1}{0+1+1+0+1} = 0.80\]

\[\sum_{all \ points \ i} \text{Data}_i \times (1-\text{HiddenVector}_i) / \sum_{all \ points \ i} (1-\text{HiddenVector}_i) =\]

\[\text{Data} \times (1-\text{HiddenVector}) \quad / \quad 1 \times (1 - \text{HiddenVector})\]
### Parameters as a Dot-Product

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTTTHTTHTH</td>
<td>0.4 * 1</td>
<td>$(\theta_A, \theta_B)$</td>
</tr>
<tr>
<td>HHHHTHTHHHHH</td>
<td>0.9 * 0</td>
<td></td>
</tr>
<tr>
<td>HTHHHHHHTHH</td>
<td>0.8 * 0</td>
<td>$(0.35, 0.80)$</td>
</tr>
<tr>
<td>HTTTTTHTHTT</td>
<td>0.3 * 1</td>
<td></td>
</tr>
<tr>
<td>TTTTTHTTTTT</td>
<td>0.7 * 0</td>
<td></td>
</tr>
</tbody>
</table>

$\theta_A = \text{fraction of heads generated in all flips with coin } A$

$= (0.4+0.3)/2=0.35$

$= Data \times HiddenVector / 1 \times HiddenVector$

$\theta_B = \text{fraction of heads generated in all flips with coin } B$

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

$= Data \times (1-HiddenVector) / 1 \times (1 - HiddenVector)$
**Data, HiddenVector, Parameters**

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters = (θ_A, θ_B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>1</td>
<td>(0.35, 0.80)</td>
</tr>
<tr>
<td>0.9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Data, **HiddenVector, Parameters**

\[
\text{Parameters} = (\theta_A, \theta_B)
\]

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>?</td>
<td>(0.35, 0.80)</td>
</tr>
<tr>
<td>0.9</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>
From *Data & Parameters to HiddenVector*

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters=$(\theta_A, \theta_B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>?</td>
<td>$(0.35, 0.80)$</td>
</tr>
<tr>
<td>0.9</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>

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

\[
\Pr(1^{\text{st}} \text{ sequence}|\theta_A) = \theta_A^4 (1-\theta_A)^6 = 0.35^4 \cdot 0.65^6 \approx 0.00113 > \\
\Pr(1^{\text{st}} \text{ sequence}|\theta_B) = \theta_B^4 (1-\theta_B)^6 = 0.80^4 \cdot 0.20^6 \approx 0.00003
\]
From Data & Parameters to HiddenVector

Data  HiddenVector  Parameters=(\( \theta_A, \theta_B \))

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>0.9</td>
<td>?</td>
</tr>
<tr>
<td>0.8</td>
<td>?</td>
</tr>
<tr>
<td>0.3</td>
<td>?</td>
</tr>
<tr>
<td>0.7</td>
<td>?</td>
</tr>
</tbody>
</table>

From Data & Parameters to HiddenVector

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

\[
\Pr(1^{st} \text{ sequence} | \theta_A) = \theta_A^4 (1-\theta_A)^6 = 0.35^4 \cdot 0.65^6 \approx 0.00113 \\
\Pr(1^{st} \text{ sequence} | \theta_B) = \theta_B^4 (1-\theta_B)^6 = 0.80^4 \cdot 0.20^6 \approx 0.00003
\]
From *Data & Parameters* to *HiddenVector*

\[
\begin{array}{ccc}
\text{Data} & \text{HiddenVector} & \text{Parameters}=(\theta_A, \theta_B) \\
0.4 & 1 & \\
0.9 & ? & (0.35, 0.80) \\
0.8 & ? & \\
0.3 & ? & \\
0.7 & ? & \\
\end{array}
\]

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

\[
\begin{align*}
\Pr(2\text{nd sequence}|\theta_A) &= \theta_A^9(1-\theta_A)^1 = 0.35^9 \cdot 0.65^1 \approx 0.00005 < \\
\Pr(2\text{nd sequence}|\theta_B) &= \theta_B^9(1-\theta_B)^1 = 0.80^9 \cdot 0.20^1 \approx 0.02684
\end{align*}
\]
From *Data & Parameters to HiddenVector*

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters=$\theta_A, \theta_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>1</td>
<td>$(0.35, 0.80)$</td>
</tr>
<tr>
<td>0.9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>

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

\[
\Pr(2\text{nd sequence}|\theta_A) = \theta_A^9(1-\theta_A)^1 = 0.35^9 \cdot 0.65^1 \approx 0.00005 < \\
\Pr(2\text{nd sequence}|\theta_B) = \theta_B^9(1-\theta_B)^1 = 0.80^9 \cdot 0.20^1 \approx 0.02684
\]
HiddenVector Reconstructed!

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters = (θ_A, θ_B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>1</td>
<td>(0.35, 0.80)</td>
</tr>
<tr>
<td>0.9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
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 $Data = (Data_1, \ldots, Data_n)$

$Parameters$: $Centers = (Center_1, \ldots, Center_k)$

$HiddenVector$: 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” $\approx 0.5$ for a midpoint.
From *Data & Parameters to HiddenVector*

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters = ((\theta_A, \theta_B))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>?</td>
<td>((0.60, 0.82))</td>
</tr>
<tr>
<td>0.9</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>

Which coin is more likely to have generated the first sequence (with 4 H)?

\[
\Pr(1^{st}\text{ sequence}|\theta_A) = \theta_A^5(1-\theta_A)^5 = 0.60^4 \cdot 0.40^6 \approx 0.000531 > \\
\Pr(1^{st}\text{ sequence}|\theta_B) = \theta_B^5(1-\theta_B)^5 = 0.82^4 \cdot 0.18^6 \approx 0.000015
\]
### Memory Flash:
From *Data & Parameters to Hidden Vector*

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters = ((\theta_A, \theta_B))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>1</td>
<td>((0.60, 0.82))</td>
</tr>
<tr>
<td>0.9</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>

Which coin is more likely to have generated the first sequence (with 4 H)?

\[
\Pr(1^{st} \text{ sequence}|\theta_A) = \theta_A^5(1-\theta_A)^5 = 0.60^4 \cdot 0.40^6 \approx 0.000531 > \\
\Pr(1^{st} \text{ sequence}|\theta_B) = \theta_B^5(1-\theta_B)^5 = 0.82^4 \cdot 0.18^6 \approx 0.000015
\]
From *Data & Parameters to HiddenMatrix*

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenMatrix</th>
<th>Parameters = ((\theta_A, \theta_B))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.97</td>
<td>0.03</td>
</tr>
<tr>
<td>0.9</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>?</td>
<td>((0.60, 0.82))</td>
</tr>
<tr>
<td>0.3</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>

What are the **responsibilities** of coins for this sequence?

\[
\text{Pr}(\text{1st sequence}|\theta_A) \approx 0.000531 > \text{Pr}(\text{1st sequence}|\theta_B) \approx 0.000015
\]

\[
0.000531 / (0.000531 + 0.000015) \approx 0.97 \\
0.000015 / (0.000531 + 0.000015) \approx 0.03
\]
From *Data & Parameters to HiddenMatrix*

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenMatrix</th>
<th>Parameters = (θ_A, θ_B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.97 0.03</td>
<td>(0.60, 0.82)</td>
</tr>
<tr>
<td>0.9</td>
<td>0.12 0.88</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>

What are the responsibilities of coins for the 2\textsuperscript{nd} sequence?

\[
\text{Pr}(2\textsuperscript{nd} \text{sequence} | \theta_A) \approx 0.0040 < \text{Pr}(2\textsuperscript{nd} \text{sequence} | \theta_B ) \approx 0.0302
\]

\[
0.0040 / (0.0040 + 0.0302) = 0.12 \\
0.0342 / (0.0040 + 0.0342) = 0.88
\]
HiddenMatrix Reconstructed!

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenMatrix</th>
<th>Parameters = ((\theta_A, \theta_B))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.97 0.03</td>
<td>(0.60, 0.82)</td>
</tr>
<tr>
<td>0.9</td>
<td>0.12 0.88</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>0.29 0.71</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.99 0.01</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.55 0.45</td>
<td></td>
</tr>
</tbody>
</table>
Expectation Maximization Algorithm

Data

HiddenMatrix

Parameters
E-step

Data

HiddenMatrix

Parameters
M-step

Data

HiddenVector

Parameters’
Memory Flash: Dot Product

Data  HiddenVector  Parameters=$=(\theta_A, \theta_B)$

<table>
<thead>
<tr>
<th></th>
<th>Data</th>
<th>*</th>
<th>HiddenVector</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHTTTHTTTTH</td>
<td>0.4</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>HHHHTTHHHHH</td>
<td>0.9</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>HTHHHHHTTHH</td>
<td>0.8</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>HHTTTTHHTTT</td>
<td>0.3</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>TTHHHHTTHHT</td>
<td>0.7</td>
<td>*</td>
<td>0</td>
</tr>
</tbody>
</table>

$\theta_A = \text{Data} \ast \text{HiddenVector} / 1 \ast \text{HiddenVector}$

$\theta_B = \text{Data} \ast (1-\text{HiddenVector}) / 1 \ast (1-\text{HiddenVector})$
From Data & HiddenMatrix to **Parameters**

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters = ( θ_A, θ_B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTHTTTHTH</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>HHHHTHHTTHH</td>
<td>0.9</td>
<td>0</td>
</tr>
<tr>
<td>HTHHHHHTTHH</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>HTTTTHHHTT</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>THHHHTHHTH</td>
<td>0.7</td>
<td>0</td>
</tr>
</tbody>
</table>

θ_A = Data * HiddenVector / 1 * HiddenVector

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

HiddenVector = (1 0 0 1 0)

What is HiddenMatrix corresponding to this HiddenVector?
### From *Data* & *HiddenMatrix* to *Parameters*

$$\text{Data} \quad \text{HiddenVector} \quad \text{Parameters} = (\theta_A, \theta_B)$$

<table>
<thead>
<tr>
<th><strong>Data</strong></th>
<th><strong>HiddenVector</strong></th>
<th><strong>Parameters</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTTHTTH</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>HHHHTTHH</td>
<td>0.9</td>
<td>0</td>
</tr>
<tr>
<td>HTHHHHHH</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>HHTTTTHH</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>TTHHTHHH</td>
<td>0.7</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \theta_A = \text{Data} \times \text{HiddenVector} / 1 \times \text{HiddenVector} \]

\[ \theta_A = \text{Data} \times 1^{\text{st}} \text{ row of HiddenMatrix} / 1 \times 1^{\text{st}} \text{ row of HiddenMatrix} \]

\[ \theta_B = \text{Data} \times (1-\text{HiddenVector}) / 1 \times (1-\text{HiddenVector}) \]

\[ \theta_B = \text{Data} \times 2^{\text{nd}} \text{ row of HiddenMatrix} / 1 \times 2^{\text{nd}} \text{ row of HiddenMatrix} \]

**HiddenVector** =  

\[
\begin{pmatrix}
1 & 0 & 0 & 1 & 0
\end{pmatrix}
\]

**Hidden Matrix** =  

\[
\begin{pmatrix}
1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1
\end{pmatrix} = 1 - \text{HiddenVector}
\]
From Data & HiddenMatrix to Parameters

Data | HiddenMatrix | Parameters = ($\theta_A, \theta_B$)

| HiddenVector | .4  | .97 | .03 |
| HHHHHTHHHHH | .9  | .12 | .88 |
| HTHHHHHTHHH | .8  | .29 | .71 |
| HHTTTTTHHTTTT | .3  | .99 | .01 |
| THHHTHHHTTHH | .7  | .55 | .45 |

$\theta_A = \text{Data} \times \text{HiddenVector} / 1 \times \text{HiddenVector}$

$\theta_A = \text{Data} \times \text{1st row of HiddenMatrix} / 1 \times \text{1st row of HiddenMatrix}$

$\theta_B = \text{Data} \times (1 - \text{HiddenVector}) / 1 \times (1 - \text{HiddenVector})$

$\theta_B = \text{Data} \times \text{2nd row of HiddenMatrix} / 1 \times \text{2nd row of HiddenMatrix}$

HiddenVector = 
\[
\begin{pmatrix}
1 & 0 & 0 & 1 & 0
\end{pmatrix}
\]

Hidden Matrix = 
\[
\begin{pmatrix}
.97 & .03 & .29 & .99 & .55 \\
.03 & .97 & .71 & .01 & .45
\end{pmatrix}
\]
From *HiddenVector* to *HiddenMatrix*

**Data:** data points $Data = \{Data_1, \ldots, Data_n\}$

**Parameters:** $Centers = \{Center_1, \ldots, Center_k\}$

**HiddenVector:** assignments of data points to centers

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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 \( j \)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
From *HiddenVector* to *HiddenMatrix*

**Data:** data points $\text{Data} = \{\text{Data}_1, \ldots, \text{Data}_n\}$

**Parameters:** $\text{Centers} = \{\text{Center}_1, \ldots, \text{Center}_k\}$

$\text{HiddenMatrix}_{i,j}$: responsibility of center $i$ for data point $j$
Responsibilities and the Law of Gravitation

The responsibility of star $i$ for a planet $j$ is proportional to the pull (Newtonian law of gravitation):

$$Force_{i,j} = \frac{1}{\text{distance}(\text{Data}_j, \text{Center}_i)^2}$$

<table>
<thead>
<tr>
<th>stars</th>
<th>planets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.70</td>
<td>0.15</td>
</tr>
<tr>
<td>0.20</td>
<td>0.80</td>
</tr>
<tr>
<td>0.10</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Hidden Matrix $ij$:

$$HiddenMatrix_{ij} = \frac{Force_{i,j}}{\sum_{\text{all centers } j} Force_{i,j}}$$
Responsibilities and Statistical Mechanics

![Diagram of data points and centers]

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 $\beta$ is a stiffness parameter.

<table>
<thead>
<tr>
<th>centers</th>
<th>0.70</th>
<th>0.15</th>
<th>0.73</th>
<th>0.40</th>
<th>0.15</th>
<th>0.80</th>
<th>0.05</th>
<th>0.05</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.20</td>
<td>0.80</td>
<td>0.17</td>
<td>0.20</td>
<td>0.80</td>
<td>0.10</td>
<td>0.05</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>0.05</td>
<td>0.10</td>
<td>0.40</td>
<td>0.05</td>
<td>0.10</td>
<td>0.90</td>
<td>0.75</td>
</tr>
</tbody>
</table>

$$HiddenMatrix_{ij} := \frac{Force_{i,j}}{\sum_{\text{all centers } j} Force_{i,j}}$$
How Does Stiffness Affect Clustering?

Hard $k$-means clustering

Soft $k$-means clustering (stiffness $\beta=1$)

Soft $k$-means clustering (stiffness $\beta=0.3$)
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.
From a Tree to a Partition into 4 Clusters

To capture stratification, the **hierarchical clustering** algorithm organises $n$ data points into a tree.
From a Tree to a Partition into 6 Clusters

To capture stratification, the **hierarchical clustering** algorithm first organises $n$ data points into a tree.
Hierarchical clustering starts from a transformation of \( n \times m \) expression matrix into \( n \times n \) similarity matrix or distance matrix.
Constructing the Tree

Identify the two closest clusters and merge them.
Constructing the Tree

Recompute the distance between two clusters as average distance between elements in the cluster.

\[
\begin{array}{cccccccccc}
g_1 & g_2 & g_3 & g_5 & g_4 & g_6 & g_7 & g_8 & g_9 & g_{10} \\
g_1 & 0.0 & 8.1 & 9.2 & 7.7 & 2.3 & 5.1 & 10.2 & 6.1 & 7.0 \\
g_2 & 8.1 & 0.0 & 12.0 & 0.9 & 9.5 & 10.1 & 12.8 & 2.0 & 1.0 \\
g_3, g_5 & 9.2 & 12.0 & 0.0 & 11.2 & 11.1 & 8.1 & 1.0 & 10.5 & 11.5 \\
g_4 & 7.7 & 0.9 & 11.2 & 0.0 & 9.2 & 9.5 & 12.0 & 1.6 & 1.1 \\
g_6 & 2.3 & 9.5 & 11.1 & 9.2 & 0.0 & 5.6 & 12.1 & 7.7 & 8.5 \\
g_7 & 5.1 & 10.1 & 8.1 & 9.5 & 5.6 & 0.0 & 9.1 & 8.3 & 9.3 \\
g_8 & 10.2 & 12.8 & 1.0 & 12.0 & 12.1 & 9.1 & 0.0 & 11.4 & 12.4 \\
g_9 & 6.1 & 2.0 & 10.5 & 1.6 & 7.7 & 8.3 & 11.4 & 0.0 & 1.1 \\
g_{10} & 7.0 & 1.0 & 11.5 & 1.1 & 8.5 & 9.3 & 12.4 & 1.1 & 0.0 \\
\end{array}
\]
Constructing the Tree

Identify the two closest clusters and merge them.
Constructing the Tree

Recompute the distance between two clusters (as average distance between elements in the cluster).
Constructing the Tree

Identify the two closest clusters and merge them.
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)
HierarchicalClustering \((D, n)\)

- Clusters \(\leftarrow n\) single-element clusters labeled 1 to \(n\)
- \(T\) \(\leftarrow\) 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 \(C_{\text{new}}\) with \(|C_i| + |C_j|\) elements
- add a new node labeled by cluster \(C_{\text{new}}\) to \(T\)
- connect node \(C_{\text{new}}\) 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 \(C_{\text{new}}\) by computing 
  \[D(C_{\text{new}}, C)\] for each cluster \(C\) in Clusters
- add \(C_{\text{new}}\) 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:

\[ D_{\text{avg}}(C_1, C_2) = \left( \sum \text{all points } i \text{ and } j \text{ in clusters } C_1 \text{ and } C_2, \text{ respectively } D_{i,j} \right) / (|C_1| \times |C_2|) \]

**Minimum distance** between elements of two clusters:

\[ D_{\text{min}}(C_1, C_2) = \min \text{ all points } i \text{ and } j \text{ in clusters } C_1 \text{ and } C_2, \text{ respectively } D_{i,j} \]
Clusters Constructed by Hierarchical Clustering

Cluster 1

Cluster 2

Cluster 3

Cluster 4

Cluster 5

Cluster 6

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
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
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 $\epsilon$, 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 \( \delta \) function \( \delta(u, v) \) is 1 if \( u = v \) and 0 otherwise and \( m = \frac{1}{2} \sum_{ij} A_{ij} \).

\( Q > 0 \) when the number of edges within groups > number of edges within groups in a randomly rewired graph.

Exact modularity optimisation is a problem that is computationally hard and so approximation algorithms are necessary when dealing with large networks.

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.
The gain in modularity $\Delta Q$ obtained by moving an isolated node $i$ into a community $C$ can easily be computed by

$$\Delta Q = \left[ \frac{\sum_{in}}{2m} + 2k_{i,in} - \left( \frac{\sum_{tot} + k_i}{2m} \right) \right] - \left[ \frac{\sum_{in}}{2m} - \left( \frac{\sum_{tot}}{2m} \right)^2 - \left( \frac{k_i}{2m} \right)^2 \right]$$

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

- $\Sigma_{in} \equiv \sum_{i,j \in C} A_{ij} \ldots$ sum of link weights between nodes in $C$
- $\Sigma_{tot} \equiv \sum_{i \in C} k_i \ldots$ sum of all link weights of nodes in $C$

$\Sigma_{in}$: $\Sigma_{in} = 10$

$\Sigma_{tot}$: $\Sigma_{tot} = 13$

\[ Q(C) = \frac{1}{2m} \sum_{i,j \in C} \left[ A_{ij} - \frac{k_i k_j}{2m} \right] = \frac{\Sigma_{i,j \in C} A_{ij}}{2m} - \frac{(\Sigma_{i \in C} k_i)(\Sigma_{j \in C} k_j)}{(2m)^2} \]

Links within the community $\frac{\Sigma_{in}}{2m}$ - Total links $\left(\frac{\Sigma_{tot}}{2m}\right)^2$

$Q(C)$ is large when most of the total links are within-community links
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.

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 $\Delta Q$.
    - 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 $\Delta Q = 0$
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

```
1: function LOUVAIN(Graph G, Partition P)
2:    do
3:        P ← MOVENODES(G, P)
4:        done ← |P| = |V(G)|
5:        if not done then
6:            G ← AGGREGATEGRAH(G, P)
7:            P ← SINGLETONPARTITION(G)
8:        end if
9:    while not done
10:     return flat*(P)
11: end function

12: function MOVENODES(Graph G, Partition P)
13:    do
14:        H_{old} = H(P)
15:        for v ∈ V(G) do
16:            C' ← arg max_{C ∈ P} \Delta H_P(v ↦ C)
17:            if \Delta H_P(v ↦ C') > 0 then
18:                v ↦ C'
19:            end if
20:        end for
21:    while H(P) > H_{old}
22:     return P
23: end function

24: function AGGREGATEGRAH(Graph G, Partition P)
25:    V ← P
26:    E ← \{(C, D) | (u, v) ∈ E(G), u ∈ C ∈ P, v ∈ D ∈ P\}
27:    return GRAPH(V, E)
28: end function

29: function SINGLETONPARTITION(Graph G)
30:     return \{\{v\} | v ∈ V(G)\}
31: end function
```

- Move nodes between communities
- Terminate when each community consists of only one node
- Create aggregate graph based on partition P
- Assign each node in aggregate graph to its own community
- Visit nodes (in random order)
- Determine best community for node v
- Perform only strictly positive node movements
- Move node v to community C’
- Continue until no more nodes can be moved
- Communities become nodes in aggregate graph
- E is a multiset
- Assign each node to its own community
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.
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.
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
Markov Clustering Algorithm:  https://micans.org/mcl/
https://towardsdatascience.com/markov-clustering-algorithm-577168dad475

Louvain/Leiden:

➢ Chapter 8 Vol 2 (pag 72-109)
Section 5

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*. 
Multiple (unsequenced) genome copies

Reads

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

so, what did the June 27, 2000 NY Times say?
The newspaper problem as an overlapping puzzle
The Newspaper Problem as an Overlapping Puzzle
Multiple Copies of a Genome (Millions of them)

CTGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC

Breaking the Genomes at Random Positions

CTGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
CTGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTACGATGCATTAGCAAGCTATCGGATCAGCTACCACATCGTAGC
Generating “Reads”

CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC
CTGATGATG GACTACGCT ACTACTGCTA GCTGTATTACG ATCAGCTACC ACACTCGTAGCT ACGATGCATTA GCAAGCTATC GGATCAGCTAC CACATCGTAGC
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCACA ATCGTAGCTACG ATGCATTAGCAA GCAAGCTATC GGATCAGCTAC CACATCGTAGC
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCACA ATCGTAGCTACG ATGCATTAGCAA GCAAGCTATC GGATCAGCTAC CACATCGTAGC

“Burning” Some Reads

CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC
CTGATGATG GACTACGCT ACTACTGCTA GCTGTATTACG ATCAGCTACC ACACTCGTAGCT ACGATGCATTA GCAAGCTATC GGATCAGCTAC CACATCGTAGC
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCACA ATCGTAGCTACG ATGCATTAGCAA GCAAGCTATC GGATCAGCTAC CACATCGTAGC
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCACA ATCGTAGCTACG ATGCATTAGCAA GCAAGCTATC GGATCAGCTAC CACATCGTAGC
CTGATGATGG ACTACGCTAC TACTGCTAGCT GTATTACGATC AGCTACCACA ATCGTAGCTACG ATGCATTAGCAA GCAAGCTATC GGATCAGCTAC CACATCGTAGC

424
No Idea What Position Every Read Comes From
What Is k-mer Composition?

\[
\text{Composition}_3(\text{TAAATGCCATGGGATGTT}) =
\]

- TAA
- AAT
- ATG
- TGCC
- GCC
- CCA
- CAT
- ATG
- TGG
- GGG
- GGA
- GAT
- ATG
- TGT
- GTT
k-mer Composition

Composition_3( TAATGCCCATGGGATGTT ) =
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)
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

TAA
AAT

OOPS!
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 $\rightarrow$ AAT
A Path Turns into a Graph

Yes. We simply need to connect k-mer₁ with k-mer₂ if suffix(k-mer₁)=prefix(k-mer₂).
E.g. TAA → AAT
A Path Turns into a Graph

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.

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
A Slightly Different Path

TAATGCCATGGGATGTT

3-mers as nodes

TAA → AAT → ATG → TGC → GCC → CCA → CAT → ATG → TGG → GGG → GGA → GAT → ATG → TGT → GTT

3-mers as edges

How do we label the starting and ending nodes of an edge?

prefix of TAA  TAA  suffix of TAA
Labeling Nodes in the New Path

 TAATGCCATGGGATGTT

3-mers as nodes

 TA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT

3-mers as edges and 2-mers as nodes

 TA AA AT TG GC CC CA AT TG GG GG GA AT TG GT TT
Labeling Nodes in the New Path

3-mers as edges and 2-mers as nodes
Gluing Identically Labeled Nodes
Gluing Identically Labeled Nodes
Gluing Identically Labeled Nodes
Gluing Identically Labeled Nodes

TAATGCCATGGGATGTT
Gluing Identically Labeled Nodes

TAATGCCATGGGATGTT
De Bruijn Graph of TAATGCCATGGGATGTT

Where is the Genome hiding in this graph?
It Was Always There!

An Eulerian path in a graph is a path that visits each edge exactly once.

TAATGCCATGGGATGTT
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?

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

TAATGCCCATGGGATGTT

[Diagram of a de Bruijn graph with nodes labeled and edges connecting them.]

451
What We Want: From Reads (k-mers) to Genome

TAATGCCATGGGATGTT
What We will Show: From Reads to de Bruijn Graph to Genome
Constructing de Bruijn Graph when Genome Is Known

TAATGCCATGGGATGTT
Constructing de Bruijn when Genome Is Unknown

Composition$_3$(TAATGCCATGGGATGT)
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
We Are Not Done with Gluing Yet
Gluing Identically Labeled Nodes
Gluing Identically Labeled Nodes
TAATGCCATGGGATGTT
The Same de Bruijn Graph:
DeBruin(Genome) = DeBruin(Genome Composition)
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
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
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!
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

Vertices are k-mers
Edges are pairwise alignments

Vertices are (k-1)-mers
Edges are k-mers

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.
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** (**BalancedGraph**)
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

TAATGCCATGGGATGT

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

Generate read-pairs: two reads from the ends of each fragment (separated by a fixed distance)
From $k$-mers to Paired $k$-mers

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(TAATGCGATGATGTT)?

Representing a paired 3-mer TAA GCC as a 2-line expression:

TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA
GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT
PairedComposition(TAACATGGGATGTT)

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 $\text{PairedComposition}(Text)$ is equal to the collection of paired $k$-mers.

How Would de Bruijn Assemble Paired $k$-mers?
Representing Genome TAATGCCATGGGATGTT as a Path

paired prefix of

paired suffix of
Labeling Nodes by Paired Prefixes and Suffixes
Glue nodes with identical labels

497
Glue nodes with identical labels

Paired de Bruijn Graph from the Genome
Constructing Paired de Bruijn Graph

paired prefix of

paired suffix of
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

Multiple genome reconstructions

TAATGCCCATGGGATGTT

TAATGCCCATGGGATGTT

TAATGGGATGCCCATGTT

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

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

atgccgtatggacacgact
atgccgtatg
gccgtatgga
gtatggacaa
gacaacgact

atgccgtatggacacgact
atgcc
tgcgg
gccgt
cgta
cgtat
gtatg
tatgg
atgga
tggac
ggaca
gacaa
acaac
caacg
aacga
acgac
cgact
2nd Unrealistic Assumption: Error-free Reads

Erroneous read (change of t into C)
De Bruijn Graph of ATGGCGTGCAATG… Constructed from Error-Free Reads

Errors in Reads Lead to Bubbles in the De Bruijn Graph
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

- Chapter 3 Vol 1 (pag 115-170)
Section 6

Genome assemble

- Suffix tree
- Algorithm: Burrow-Wheeler Transform
• **Reference genome:** database genome used for comparison (GRCh38).
  

• **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

Assemble the genome with overlapping reads

AGAATATCA  TGAGAATAT  GAGAATATC

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.

CTGAGGATGGACTACGCTACTACTACTGATAGCTGTTT
GAGGA   CCACG       TGA–A

Reference Reads
Why Not Use Alignment?

• **Fitting alignment**: align each read *Pattern* to the best substring of *Genome*.

• Has runtime $O(|Pattern| \times |Genome|)$ for each *Pattern*.

• Has runtime $O(|Patterns| \times |Genome|)$ 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
banana 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: $O(|Genome| \times |Pattern|)$
  – Multiple Patterns: $O(|Genome| \times |Patterns|)$
  – $|Patterns| = \text{combined length of Patterns}$
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.
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.
panamabanananas
• Runtime of Brute Force:
  – Total: $O(|Genome| \times |Patterns|)$

• Runtime of Trie Matching:
  – Trie Construction: $O(|Patterns|)$
  – Pattern Matching: $O(|Genome| \times |LongestPattern|)$
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!
The Suffix Trie and Pattern Matching

- For each Pattern, see if Pattern can be spelled out from the root downward in the suffix trie.
Memory Trouble Once Again

• Worst case: the suffix trie holds $O(|\text{Suffixes}|)$ nodes.

• For a Genome of length $n$, $|\text{Suffixes}| = n(n - 1)/2 = O(n^2)$
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.
• 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

```
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
AAAAAAATTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
```

Run-length encoding

```
10G11C7A15T5C1G
```

- Problem: Genomes don’t have lots of runs...
Converting Repeats to Runs

• ...but they do have lots of repeats!

\[ \text{Genome} \rightarrow \text{Convert repeats to runs} \rightarrow \text{Genome*} \rightarrow \text{Run-length encoding} \rightarrow \text{CompressedGenome*} \]
The Burrows – Wheeler Transform

Michael Burrows (left), David Wheeler (right) both at the Computer Laboratory
The Burrows Wheeler Transform

Three steps: 1) Given a string $T$ in input, we form a $N \times N$ 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).
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).
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\$. 

Final $T$
Burrows-Wheeler Transform (BWT)

Burrows-Wheeler Matrix (BWM)
Burrows-Wheeler Matrix

$acaacg$

aacg$aca$

acaacg$

acg$aca$

cacg$aca$

cg$aca$

ac$aca$

g$aca$ac
### Burrows-Wheeler Matrix

<table>
<thead>
<tr>
<th></th>
<th>$acaacg$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>aacg$ac</td>
</tr>
<tr>
<td>1</td>
<td>acaacg$</td>
</tr>
<tr>
<td>4</td>
<td>acg$aca</td>
</tr>
<tr>
<td>2</td>
<td>caacg$a</td>
</tr>
<tr>
<td>5</td>
<td>cg$acaa</td>
</tr>
<tr>
<td>6</td>
<td>g$acaac</td>
</tr>
</tbody>
</table>

See the suffix array?
Key observation

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.

```
a^1c^1a^2a^3c^2g^1$
```

“last first (LF) mapping”

```
1 $acaaacg^1$
2 $aacg^1ac$
3 $acg^1aca$
1 $caacg^1$
2 $cg^1aca$
3 $g^1aca$
```
Burrow Wheeler Transform

(a) $a c a a c g$
    $a a c g $ a c$
    $a c a a c g$

$a c a a c g$ $a c g $ a c a $
    $g c $ a a a c$

(c) $a a c$
    $a a c$
    $a a c$

(b) $g$
    $c g$
    $a c g$

$g $ a c a a c$
    $g a c a a c$
    $g a c a a c$

$g a c a a c$
    $g a c a a c$
    $g a c a a c$

$g a c a a c$ $g a c a a c$
    $g a c a a c$
    $g a c a a c$

$g a c a a c$
    $g a c a a c$
    $g a c a a c$
The Burrows-Wheeler Transform

Form all cyclic rotations of "panamabanananas$"

The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabananas$”
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabanananas$”

Sort the strings lexicographically ($ comes first)
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabanananas$”

Burrows-Wheeler Transform:
Last column = smnpbnnaaaaaaa$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$banana
- ana$ban
- ana$ban
- banana$
- na$banana
- na$ban
- na$ban
- na$ban

2-mers

- $b$ $a$
- a $a$
- na $na$
- ba $ba$
- $b$ $a$
- a $n$
- na $na$
- na $na$

Sort

- $b$
- a$
- a$
- a$
- a$
- an
- 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$bana$  na$  a$b
ana$bana$  nan  ana
ana$ba$  ban  ana
banana$  $ba  ban
na$banana  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

- We now know 4-mer composition of the circular string banana$
- Sorting gives us the first 4 columns of the matrix.
Reconstructing banana

We now know 5-mer composition of the circular string banana$

Sorting gives us the first 5 columns of the matrix.
Reconstructing banana

$banana \quad a$banana \quad $banan
a$banan \quad na$ban \quad a$bbna
ana$ban \quad nana$b \quad anaaba
anana$b \quad banana \quad anaaba$
banana$ \quad $banan \quad bannna
na$banana \quad ana$ba \quad na$ban
nana$ba \quad anana$ \quad 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

<table>
<thead>
<tr>
<th>$\text{banana}$</th>
<th>a$\text{bana}$</th>
<th>$\text{banan}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a$\text{banan}$</td>
<td>na$\text{ban}$</td>
<td>a$\text{bbna}$</td>
</tr>
<tr>
<td>ana$\text{ban}$</td>
<td>nana$\text{b}$</td>
<td>anaaba</td>
</tr>
<tr>
<td>anana$\text{b}$</td>
<td>banana</td>
<td>anaaaa$\text{a}$</td>
</tr>
<tr>
<td>banana$\text{b}$</td>
<td>6-mers</td>
<td>$\text{banan}$</td>
</tr>
<tr>
<td>na$\text{bana}$</td>
<td>ana$\text{ba}$</td>
<td>na$\text{ban}$</td>
</tr>
<tr>
<td>nana$\text{ba}$</td>
<td>anana$\text{a}$</td>
<td>nana$\text{b}$</td>
</tr>
</tbody>
</table>

- We now know 6-mer composition of the circular string banana$\text{a}$
- Sorting gives us the first 6 columns of the matrix.
Reconstructing banana

\[
\begin{array}{c}
\$ \text{banana} \\
 a \$ \text{banana} \\
 a \text{ana} \$ \text{ban} \\
 a \text{anana} \$ b \\
 b \text{anana} \$ \\
 n a \$ \text{bana} \\
 n a n a \$ b a
\end{array}
\]

• 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$banan$
anana$ban$
anana$b
banana$
nana$ba
na$bana
ana$ba

• Can we invert BWT with less space?
A Strange Observation

panamabananas
abananas$panama
amabananas$pan
anamabananas$pan
ananas$panamaban
anas$panamaban
ananas$panamaban
bananas$panama
mabananas$pana
namabananas$pan
nananas$panamabana
nas$panamaban
panamabananas$
s$panamabanana
A Strange Observation

$\text{panamabananas}$
$\text{abananas}$
$\text{amabananas}$
$\text{ananasa}$
$\text{amanabananas}$
$\text{panamananas}$
$\text{panamanananas}$
$\text{panamananas}$
$\text{ananamabananas}$
$\text{panamananas}$

$\text{s}$
$\text{p}$
$\text{a}$
$\text{n}$
$\text{m}$
$\text{a}$
$\text{b}$
Is It True in General?

These strings are sorted

$panamabananas$

1. abananas$panam$
2. amabananas$pan$
3. anamabananas$p$
4. ananas$panamab$
5. nas$panamaban$
6. as$panamaban$

bananas$panam$

mabananas$panam$

namabananas$p$
nanas$panamab$
nas$panamaban$
s$panamabanana$

Chop off a
Is It True in General?

These strings are sorted

$panamabanananas$
$panamabananas$pan
$mabananas$pan
$namabananas$pan
$nanas$panamab
$nas$panamaban
$s$panamabanan

Still sorted

Chop off a
Is It True in General?

These strings are sorted:

<table>
<thead>
<tr>
<th></th>
<th>banana$panama</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>apanamabananana</td>
</tr>
<tr>
<td>2</td>
<td>amabanananas$pan</td>
</tr>
<tr>
<td>3</td>
<td>anamabananas$p</td>
</tr>
<tr>
<td>4</td>
<td>ananas$panamab</td>
</tr>
<tr>
<td>5</td>
<td>anas$panamaban</td>
</tr>
<tr>
<td>6</td>
<td>a$panamabanan</td>
</tr>
</tbody>
</table>

Chop off a:

<table>
<thead>
<tr>
<th></th>
<th>banana$panama</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>amabanananas$pan</td>
</tr>
<tr>
<td>2</td>
<td>namabananas$pan</td>
</tr>
<tr>
<td>3</td>
<td>nanas$panamab</td>
</tr>
<tr>
<td>4</td>
<td>panamabananas$</td>
</tr>
<tr>
<td>5</td>
<td>panamabanan</td>
</tr>
<tr>
<td>6</td>
<td>panamabana</td>
</tr>
</tbody>
</table>

Still sorted.

Add a to end:

<table>
<thead>
<tr>
<th></th>
<th>banana$panama</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>panamabanananas$pan</td>
</tr>
<tr>
<td>2</td>
<td>namabanananas$pan</td>
</tr>
<tr>
<td>3</td>
<td>nanas$panamab</td>
</tr>
<tr>
<td>4</td>
<td>panamabananas$</td>
</tr>
<tr>
<td>5</td>
<td>panamabanan</td>
</tr>
<tr>
<td>6</td>
<td>panamabana</td>
</tr>
</tbody>
</table>

Still sorted.

Ordering doesn’t change!
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*.
More Efficient BWT Decompression

$s_1\text{panamabanana}s_1$
a_1\text{bananas}$\text{panam}_1$
a_2\text{mabanananas}$\text{pan}_1$
a_3\text{namabanananas}$p_1$
a_4\text{nanas}$\text{panamab}_1$
a_5\text{nas}$\text{panamaban}_2$
a_6\text{s}$\text{panamabanana}_3$
b_1\text{ananas}$\text{panama}_1$
m_1\text{abananas}$\text{pana}_2$
n_1\text{ambananas}$\text{pana}_3$
n_2\text{anana}$\text{panamaba}_4$
n_3\text{nas}$\text{panamaban}_a$
p_1\text{anamabananas}$s_1$
s_1\text{panamabanana}_6
More Efficient BWT Decompression

\[ \begin{align*}
    &\$ \text{panamabananas}_1 \\
    &a_1 \text{bananas} \$ \text{panam}_1 \\
    &a_2 \text{mabananas} \$ \text{pan}_1 \\
    &a_3 \text{namabananas} \$ p_1 \\
    &a_4 \text{nanas} \$ \text{panamab}_1 \\
    &a_5 \text{nas} \$ \text{panamaban}_2 \\
    &a_6 \text{s} \$ \text{panamabananas}_3 \\
    &b_1 \text{ananas} \$ \text{panama}_1 \\
    &m_1 \text{abanananas} \$ \text{pana}_2 \\
    &n_1 \text{amabananas} \$ p_3 \\
    &n_2 \text{ananas} \$ \text{panamaba}_4 \\
    &n_3 \text{as} \$ \text{panamaban}_5 \\
    &p_1 \text{anamabananas} \$ _1 \\
    &s_1 \$ \text{panamabanana}_6
\end{align*} \]
More Efficient BWT Decompression

- Memory: $2|Genome| = O(|Genome|)$. 

\[
\begin{align*}
&s_1\text{panamabananas}s_1 \\
a_1\text{bananas}$panam_1 \\
a_2\text{mabanananas}$pan_1 \\
a_3\text{namabanananas}$p_1 \\
a_4\text{nanananas}$panamab_1 \\
a_5\text{nas}$panamaban_2 \\
a_6\text{s}$panamabanar_3 \\
b_1\text{anananas}$panama_1 \\
m_1\text{abanananas}$pana_2 \\
n_1\text{amabanananas}$pa_3 \\
n_2\text{ananas}$panamaba_4 \\
n_3\text{as}$panamaban_5 \\
p_1\text{anamabanananas}$s_1 \\
s_1\text{$panamabananas}_6
\end{align*}
\]
Recalling Our Goal

• Suffix Tree Pattern Matching:
  – Runtime: $O(|Genome| + |Patterns|)$
  – Memory: $O(|Genome|)$
  – Problem: suffix tree takes $20 \times |Genome|$ space

• Can we use BWT($Genome$) as our data structure instead?
Finding Pattern Matches Using BWT

• Searching for **ana** in **panamabanananas**

\$\text{panamabanananas}_1$
\text{abananasspanama}_1$
\text{mabananasspana}_1$
\text{namabananasspana}_1$
\text{ananas$panamab$_1}$
\text{nas$panamaban$_2}$
\text{as$panamaban$_3}$
\text{bananas$panama$_1}$
\text{abananas$pana$_2}$
\text{mabananas$pana$_3}$
\text{bananas$panama$_4}$
\text{anas$panamaban$_5}$
\text{anamabanananas$pana$_1}$
\text{ananamabanananas$pana$_1}$
\text{anamabananas$pana$_1}$
\text{ananamabananas$pana$_1}$
\text{ananamabananas$pana$_1}$

586
Finding Pattern Matches Using BWT

• Searching for \texttt{ana} in \texttt{panamabanananas}

\begin{align*}
\texttt{s}_1 \texttt{panamabanananas}_1 \\
\texttt{a}_1 \texttt{bananas}_2 \texttt{p}_1 \texttt{anam}_1 \\
\texttt{a}_2 \texttt{mabanananas}_3 \texttt{p}_1 \\
\texttt{a}_3 \texttt{namabanananas}_4 \texttt{p}_1 \\
\texttt{a}_4 \texttt{nanas}_5 \texttt{panamab}_1 \\
\texttt{a}_5 \texttt{nas}_6 \texttt{panamaban}_2 \\
\texttt{a}_6 \texttt{s}_7 \texttt{panamabanan}_3 \\
\texttt{b}_1 \texttt{ananas}_8 \texttt{panama}_1 \\
\texttt{m}_1 \texttt{abanananas}_9 \texttt{pan}_2 \\
\texttt{n}_1 \texttt{amabanananas}_10 \texttt{pa}_3 \\
\texttt{n}_2 \texttt{anas}_11 \texttt{panamaba}_4 \\
\texttt{n}_3 \texttt{as}_12 \texttt{panamabana}_5 \\
\texttt{p}_1 \texttt{anamabanananas}_13 \texttt{s}_1 \\
\texttt{s}_1 \texttt{panamabanananas}_14
\end{align*}
Finding Pattern Matches Using BWT

• Searching for **ana** in panamabanananas

\[
\$ a_1 b a n a n a s \$ p a n a m a_1 \\
a_2 m a b a n a n a s \$ p a n a_1 \\
a_3 n a m a b a n a n a s \$ p a n a_1 \\
a_4 n a n a s \$ p a m a b a_1 \\
a_5 n a s \$ p a n a m a b a_2 \\
a_6 s \$ p a n a m a b a n a_3 \\
b_1 a n a n a s \$ p a n a m a_1 \\
m_1 a b a n a n a s \$ p a n a_2 \\
n_1 a m a b a n a n a s \$ p a n a_3 \\
n_2 a n a s \$ p a n a m a b a_4 \\
n_3 a s \$ p a n a m a b a n a_5 \\
p_1 a n a m a b a n a n a s \$ p a n a_1 \\
s_1 \$ p a n a m a b a n a n a_6
\]
Finding Pattern Matches Using BWT

• Searching for **ana** in **panamabanananas**

$\sigma_1$panamabanananas\n\a_1$bananass$panam_1\n\a_2$abananass$pan_1\n\a_3$namabananas$\p_1\n\a_4$ananass$panamab_1\n\a_5$anas$panamaban_2\n\a_6$as$panamaban_3\n\b_1$ananass$panama_1\n\m_1$abananass$pana_2\n\n_1$amabananas$\p_3\n\n_2$ananass$panamab_4\n\n_3$as$panamaban_5\n\p_1$anamabananas$\_1\n\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 \textbf{ana} occurs 3 times, but where?

\begin{verbatim}
$1panamabananas\ s_1$
a$1bananas$ pan$1$
a$2mabanananas$ pan$1$
\textbf{a}$3na$ mabanananas$ p$1$
a$4na$ nas$ panamab$1$
a$5na$ s$ panamaban$2$
a$6s$ panamabanana$3$
b$1ananas$ panama$1$
m$1abanananas$ pan$2$
n$1amabananas$ pa$3$
n$2ananas$ panamaba$4$
n$3as$ panamabana$5$
p$1anamabanananas$ p$1$
s$1$ panamabana$6$
\end{verbatim}
Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

\[
\begin{array}{ll}
\text{\$} & \text{panamabanananas}_1 \\
\text{a}_1 & \text{bananas\$panam}_1 \\
\text{a}_2 & \text{mabananas\$pan}_1 \\
\text{a}_3 & \text{namabananas\$p}_1 \\
\text{a}_4 & \text{nanas\$panamab}_1 \\
\text{a}_5 & \text{nas\$panamaban}_2 \\
\text{a}_6 & \text{ss\$panamaban}_3 \\
\text{b}_1 & \text{ananas\$panama}_1 \\
\text{m}_1 & \text{abananas\$pana}_2 \\
\text{n}_1 & \text{amabananas\$pa}_3 \\
\text{n}_2 & \text{anas\$panamaba}_4 \\
\text{n}_3 & \text{as\$panamaban}_5 \\
\text{p}_1 & \text{nanamabananas}_1 \\
\text{s}_1 & \text{$panamabanana}_6 \\
\end{array}
\]
Using the Suffix Array to Find Matches

- **Suffix array**: holds starting position of each suffix beginning a row.

```
panamabanananas$
```

```
1 3
$s_1 panamabanananas$
$a_1 bananas$ panam$
$a_2 mabanananas$ pan$
$a_3 namabanananas$ p$
$a_4 nanas$ panamab$
$a_5 nas$ panamaban$
$a_6 s$ panamabanana$
$b_1 ananas$ panama$
$m_1 abanananas$ pana$
$n_1 amabanananas$ pa$
$n_2 anas$ panamaba$
$n_3 as$ panamabana$
$p_1 panamabananas$ 
$s_1$ panamabanana$
```
Using the Suffix Array to Find Matches

- **Suffix array**: holds starting position of each suffix beginning a row.

```
panamabanananas$
```

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\text{a}_1 \text{bananas}$ $\text{panam}_1$
$\text{a}_2 \text{mabanananas}$ $\text{pan}_1$
$\text{a}_3 \text{namabanananas}$ $\text{p}_1$
$\text{a}_4 \text{nanas}$ $\text{panam}_1$
$\text{a}_5 \text{nas}$ $\text{panamaban}_2$
$\text{a}_6 \text{s}$ $\text{panamabanan}_3$
$\text{b}_1 \text{ananas}$ $\text{panama}_1$
$\text{m}_1 \text{abanananas}$ $\text{pana}_2$
$\text{n}_1 \text{amabanananas}$ $\text{pa}_3$
$\text{n}_2 \text{anas}$ $\text{panamaba}_4$
$\text{n}_3 \text{as}$ $\text{panamabana}_5$
$\text{p}_1 \text{anamabanananas}$ $\text{p}_1$
$\text{s}_1 \text{$panamabana}_6$
Using the Suffix Array to Find Matches

- **Suffix array**: holds starting position of each suffix beginning a row.

```
panamabanananas$
```

```
1 3
5
3
```

```
$1_{panamabanananas}$
a1_{bananas}$panam$
a2_{mabanananas}$pana$
a3_{namabanananas}$p$
a4_{nanas}$panama$
a5_{nas}$panamaban$
a6_{s}$panamabanana$
b1_{ananas}$panama$
m1_{abanananas}$pana$
n1_{amabanananas}$pa$
n2_{ananas}$panamaba$
n3_{as}$panamabana$
p1_{anamabanananas}$p$
s1_{}$panamabana
```
Using the Suffix Array to Find Matches

- **Suffix array:** holds starting position of each suffix beginning a row.

\[
\text{panamabanananas}$
\]
Using the Suffix Array to Find Matches

- **Suffix array**: holds starting position of each suffix beginning a row.

```plaintext
panamab\underline{ana}\underline{nas}\underline{s}$
```

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>13</th>
<th>17</th>
<th>31</th>
<th>35</th>
<th>53</th>
<th>59</th>
<th>71</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$1_{panamabana}$nasa</td>
<td>$a_{1}_{bananas}\text{ }a$</td>
<td>$a_{2}_{mabananasa}\text{ }a$</td>
<td>$a_{3}_{namabananasa}\text{ }a$</td>
<td>$a_{4}_{nanasa}\text{ }a$</td>
<td>$a_{5}_{nas}\text{ }a$</td>
<td>$a_{6}_{s}\text{ }a$</td>
<td>$b_{1}_{ananas}\text{ }a$</td>
<td>$m_{1}_{abananasa}\text{ }a$</td>
<td>$n_{1}_{amabananasa}\text{ }a$</td>
<td>$n_{2}_{anas}\text{ }a$</td>
<td>$n_{3}_{as}\text{ }a$</td>
</tr>
</tbody>
</table>
Using the Suffix Array to Find Matches

• **Suffix array**: holds starting position of each suffix beginning a row.
Using the Suffix Array to Find Matches

- **Suffix array**: holds starting position of each suffix beginning a row.

\[\text{panamabanananas}^1\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$1\text{panamabanananas}_1$</th>
<th>$a_1\text{bananas}_2$</th>
<th>$a_2\text{mabanananas}_3$</th>
<th>$a_3\text{namabanananas}_4$</th>
<th>$a_4\text{nanas}_5$</th>
<th>$a_5\text{nas}_6$</th>
<th>$a_6\text{s}_7$</th>
<th>$b_1\text{ananasan}_8$</th>
<th>$m_1\text{abananas}_9$</th>
<th>$n_1\text{amabananas}_10$</th>
<th>$n_2\text{anas}_11$</th>
<th>$n_3\text{as}_12$</th>
<th>$p_1\text{anamabananas}_13$</th>
<th>$s_1\text{panamabanana}_14$</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>5</td>
<td>16</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

599
Using the Suffix Array to Find Matches

- **Suffix array**: holds starting position of each suffix beginning a row.

```
$ 1 \text{panamabananas}_1
a 1 \text{bananas}_1$
\text{panam}_1
a 2 \text{mabananas}_1$
\text{panam}_1
a 3 \text{namabanananas}_1$
p 1
a 4 \text{nanas}_1$
\text{panamab}_1
a 5 \text{nas}_1$
\text{panamaban}_2
a 6 \text{s}_1$
\text{panamabanana}_3
b 1 \text{anananas}_1$
\text{panama}_1
m 1 \text{abananas}_1$
p 2
n 1 \text{amabananas}_1$
p 3
n 2 \text{anas}_1$
\text{panamaba}_4
n 3 \text{as}_1$
\text{panamabana}_5
p 1 \text{anamabananas}_1$
p 1
s 1 \text{$panamabanana}_6
```
Using the Suffix Array to Find Matches

- **Suffix array**: holds starting position of each suffix beginning a row.
Using the Suffix Array to Find Matches

- **Suffix array**: holds starting position of each suffix beginning a row.

```
panamabananas$
```

```
<table>
<thead>
<tr>
<th></th>
<th>$1 \text{panamabanananas}$</th>
<th>$a_1 \text{bananas}$</th>
<th>$a_2 \text{mabanananas}$</th>
<th>$a_3 \text{namabanananas}$</th>
<th>$a_4 \text{ananas}$</th>
<th>$a_5 \text{nas}$</th>
<th>$a_6 \text{s}$</th>
<th>$b_1 \text{anananas}$</th>
<th>$m_1 \text{abanananas}$</th>
<th>$n_1 \text{amabanananas}$</th>
<th>$n_2 \text{anas}$</th>
<th>$n_3 \text{as}$</th>
<th>$p_1 \text{anamabanananas}$</th>
<th>$s_1 \text{spanamabananana}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$a_1 \text{bananas}$</td>
<td>$a_2 \text{mabanananas}$</td>
<td>$a_3 \text{namabanananas}$</td>
<td>$a_4 \text{ananas}$</td>
<td>$a_5 \text{nas}$</td>
<td>$a_6 \text{s}$</td>
<td>$b_1 \text{anananas}$</td>
<td>$m_1 \text{abanananas}$</td>
<td>$n_1 \text{amabanananas}$</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$a_2 \text{mabanananas}$</td>
<td>$a_3 \text{namabanananas}$</td>
<td>$a_4 \text{ananas}$</td>
<td>$a_5 \text{nas}$</td>
<td>$a_6 \text{s}$</td>
<td>$b_1 \text{anananas}$</td>
<td>$m_1 \text{abanananas}$</td>
<td>$n_1 \text{amabanananas}$</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$a_3 \text{namabanananas}$</td>
<td>$a_4 \text{ananas}$</td>
<td>$a_5 \text{nas}$</td>
<td>$a_6 \text{s}$</td>
<td>$b_1 \text{anananas}$</td>
<td>$m_1 \text{abanananas}$</td>
<td>$n_1 \text{amabanananas}$</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$a_4 \text{ananas}$</td>
<td>$a_5 \text{nas}$</td>
<td>$a_6 \text{s}$</td>
<td>$b_1 \text{anananas}$</td>
<td>$m_1 \text{abanananas}$</td>
<td>$n_1 \text{amabanananas}$</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$a_5 \text{nas}$</td>
<td>$a_6 \text{s}$</td>
<td>$b_1 \text{anananas}$</td>
<td>$m_1 \text{abanananas}$</td>
<td>$n_1 \text{amabanananas}$</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>$a_6 \text{s}$</td>
<td>$b_1 \text{anananas}$</td>
<td>$m_1 \text{abanananas}$</td>
<td>$n_1 \text{amabanananas}$</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$b_1 \text{anananas}$</td>
<td>$m_1 \text{abanananas}$</td>
<td>$n_1 \text{amabanananas}$</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$m_1 \text{abanananas}$</td>
<td>$n_1 \text{amabanananas}$</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$n_1 \text{amabanananas}$</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$n_2 \text{anas}$</td>
<td>$n_3 \text{as}$</td>
<td>$p_1 \text{anamabanananas}$</td>
<td>$s_1 \text{spanamabananana}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using the Suffix Array to Find Matches

- **Suffix array**: holds starting position of each suffix beginning a row.

<table>
<thead>
<tr>
<th>Index</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$<em>{1}$panamabanananas$</em>{1}$</td>
</tr>
<tr>
<td>2</td>
<td>$_{1}$bananas$panam$_1</td>
</tr>
<tr>
<td>3</td>
<td>$_{2}$mabananas$panam$_1</td>
</tr>
<tr>
<td>4</td>
<td>$<em>{3}$namabananas$</em>{p1}$</td>
</tr>
<tr>
<td>5</td>
<td>$_{4}$nanas$panamab$_1</td>
</tr>
<tr>
<td>6</td>
<td>$_{5}$nas$panamaban$_2</td>
</tr>
<tr>
<td>7</td>
<td>$_{6}$s$panamabanana$_3$</td>
</tr>
<tr>
<td>8</td>
<td>$_{7}$ananas$panama$_1</td>
</tr>
<tr>
<td>9</td>
<td>$_{8}$bananas$panama$_2</td>
</tr>
<tr>
<td>10</td>
<td>$<em>{9}$amabananas$</em>{p3}$</td>
</tr>
<tr>
<td>11</td>
<td>$_{10}$anas$panamaba$_4</td>
</tr>
<tr>
<td>12</td>
<td>$_{11}$as$panamaban$_5</td>
</tr>
</tbody>
</table>

panamabanananas$
Using the Suffix Array to Find Matches

• **Suffix array:** holds starting position of each suffix beginning a row.

```
panamabanananas$
```

<table>
<thead>
<tr>
<th>1 3</th>
<th>$1_{panamabanananas}$ 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>a1_{bananas}$panam$ 1</td>
</tr>
<tr>
<td>3</td>
<td>a2_{mabananas}$pan$ 1</td>
</tr>
<tr>
<td>1</td>
<td>a3_{namabananas}$p$ 1</td>
</tr>
<tr>
<td>7</td>
<td>a4_{nanas}$panamab$ 1</td>
</tr>
<tr>
<td>9</td>
<td>a5_{nas}$panamaban$ 2</td>
</tr>
<tr>
<td>11</td>
<td>a6_{s}$panamabanana$ 3</td>
</tr>
<tr>
<td>6</td>
<td>b1_{ananas}$panama$ 1</td>
</tr>
<tr>
<td>4</td>
<td>m1_{abananas}$pana$ 2</td>
</tr>
<tr>
<td>2</td>
<td>n1_{amabananas}$pa$ 3</td>
</tr>
<tr>
<td>8</td>
<td>n2_{anas}$panamaba$ 4</td>
</tr>
<tr>
<td>10</td>
<td>n3_{as}$panamabana$ 5</td>
</tr>
<tr>
<td>12</td>
<td>p1_{anamabananas}$ 1</td>
</tr>
<tr>
<td></td>
<td>s1$_{panamabana}$a_6</td>
</tr>
</tbody>
</table>
```
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$`. 

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>12</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
</tbody>
</table>

panamabananas$
The Suffix Array: Memory Once Again

- Memory: $\sim 4 \times |\text{Genome}|$. 

[Diagram of suffix array with labels and indices]

[Output array: \[13 \ 5 \ 3 \ 1 \ 7 \ 9 \ 11 \ 6 \ 4 \ 2 \ 8 \ 10 \ 0 \ 12\]]
The Suffix Array: Memory Once Again

- Memory: $\sim 4 \times |\text{Genome}|$. 

[diagram of suffix array with nodes labeled with suffixes and numbers]
The Suffix Array: Memory Once Again

• Memory: \( \sim 4 \times |Genome| \).
Returning to Our Original Problem

• We need to look at INEXACT matching in order to find variants.

• **Approx. Pattern Matching Problem:**
  – **Input:** A string \( \text{Pattern} \), a string \( \text{Genome} \), and an integer \( d \).
  – **Output:** All positions in \( \text{Genome} \) where \( \text{Pattern} \) 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 \textit{Pattern} appears in \textit{Genome} with 1 mismatch:

\[\text{Pattern} \quad \text{acttggct} \]
\[\text{Genome} \quad \ldots ggca c acta gggct cc \ldots\]
Method 1: Seeding

• Say that *Pattern* appears in *Genome* with 1 mismatch:

  \[
  \begin{align*}
  &\text{Pattern} & \text{acttggct} \\
  &\text{Genome} & \ldots\text{ggcacactagggctcc} \ldots
  \end{align*}
  \]

• 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 ana in panamabanananas

Now we extend all strings with at most 1 mismatch.

<table>
<thead>
<tr>
<th>String</th>
<th>Mismatches</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 panamabanana s_1</td>
<td>1</td>
</tr>
<tr>
<td>a_1 bananas $ panam_1</td>
<td>0</td>
</tr>
<tr>
<td>a_2 mabananas $ pan_1</td>
<td>0</td>
</tr>
<tr>
<td>a_3 namabananas $ p_1</td>
<td>1</td>
</tr>
<tr>
<td>a_4 nanas $ panama b_1</td>
<td>1</td>
</tr>
<tr>
<td>a_5 nas $ panamaban_2</td>
<td>0</td>
</tr>
<tr>
<td>a_6 s $ panamabanana_3</td>
<td>0</td>
</tr>
<tr>
<td>b_1 ananas $ panama_1</td>
<td>0</td>
</tr>
<tr>
<td>m_1 abananas $ pana_2</td>
<td>0</td>
</tr>
<tr>
<td>n_1 amabananas $ pa_3</td>
<td>0</td>
</tr>
<tr>
<td>n_2 anas $ panamaba_4</td>
<td>0</td>
</tr>
<tr>
<td>n_3 a $ panamabana_5</td>
<td>0</td>
</tr>
<tr>
<td>p_1 anamabananas $ _1</td>
<td>0</td>
</tr>
<tr>
<td>s_1 $ panamabanana_a_6</td>
<td>0</td>
</tr>
</tbody>
</table>
Method 2: BWT Saves the Day Again

• Recall: searching for \textbf{ana} in \textit{panamabanananas}

One string produces a second mismatch (the $), so we discard it.
Method 2: BWT Saves the Day Again

• Recall: searching for *ana* in *panamabanananas*

In the end, we have five 3-mers with at most 1 mismatch.

<table>
<thead>
<tr>
<th>$s_1$panamabananas</th>
<th>$a_1bana$nas$panam$_1</th>
<th>$a_2mabananas$pan$_1$</th>
<th>$a_3nabananas$pan$_1$</th>
<th>$a_4nana$s$panamab$_1$</th>
<th>$a_5nas$panamaban$_2$</th>
<th>$a_6$s$panamabanan$_3$</th>
<th>$b_1ananas$panama$_1$</th>
<th>$m_1abananas$pana$_2$</th>
<th>$n_1amabananas$pana$_3$</th>
<th>$n_2anas$panamaba$_4$</th>
<th>$n_3as$panamaban$_5$</th>
<th>$p_1anamabananas$$_1$</th>
<th>$s_1$panamabanana$$_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Mismatches</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Method 2: BWT Saves the Day Again

• Recall: searching for $\textit{ana}$ in $\textit{panamabanananas}$

In the end, we have five 3-mers with at most 1 mismatch.

Suffix Array

<table>
<thead>
<tr>
<th>Suffix Array</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{s}_1$panamabananas</td>
<td>1</td>
</tr>
<tr>
<td>$\text{a}_1$bananashpanam</td>
<td>5</td>
</tr>
<tr>
<td>$\text{a}_2$mabananashpan</td>
<td>3</td>
</tr>
<tr>
<td>$\text{a}_3$namabananashp</td>
<td>1</td>
</tr>
<tr>
<td>$\text{a}_4$nanashpanamab</td>
<td>7</td>
</tr>
<tr>
<td>$\text{a}_5$nas$\text{p}$namaban</td>
<td>9</td>
</tr>
<tr>
<td>$\text{a}_6$s$\text{p}$namaban</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Suffix Array</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$1panamabananas$</td>
<td>5</td>
</tr>
<tr>
<td>ananas$panam1</td>
<td>3</td>
</tr>
<tr>
<td>mabananas$pana1</td>
<td>1</td>
</tr>
<tr>
<td>namabananas$p1</td>
<td>7</td>
</tr>
<tr>
<td>nanas$panamab1</td>
<td>9</td>
</tr>
<tr>
<td>a5$panamaban2</td>
<td></td>
</tr>
<tr>
<td>a6$panamaban2</td>
<td></td>
</tr>
<tr>
<td>b1ananas$panama1</td>
<td></td>
</tr>
<tr>
<td>m1abananas$pana2</td>
<td></td>
</tr>
<tr>
<td>m1a$panamaban3</td>
<td></td>
</tr>
<tr>
<td>n1amabananas$pa3</td>
<td></td>
</tr>
<tr>
<td>n2anas$panamaba4</td>
<td></td>
</tr>
<tr>
<td>n3as$panamaban5</td>
<td></td>
</tr>
<tr>
<td>p1anamabananas$1</td>
<td></td>
</tr>
<tr>
<td>s1$panamabanana6</td>
<td></td>
</tr>
</tbody>
</table>
Useful link for computing BWT:
http://www.allisons.org/ll/AlgDS/Strings/BWT/

➢ Chapter 9 Vol 2 (pag 120-173)
Section 7

Algorithms to find parts

- Algorithm: Viterbi
- Algorithm: Forward
- Algorithm: Backward
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.
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
**Definition:** A hidden Markov model (HMM)

- **Alphabet** \( \Sigma = \{ b_1, b_2, ..., b_M \} \)
- **Set of states** \( Q = \{ 1, ..., K \} \)
- **Transition probabilities** between any two states
  \[ a_{ij} = \text{transition prob from state } i \text{ to state } j \]
  \[ a_{i1} + ... + a_{iK} = 1, \text{ for all states } i = 1...K \]
- **Start probabilities** \( a_{0i} \)
  \[ a_{01} + ... + a_{0K} = 1 \]
- **Emission probabilities** within each state
  \[ e_i(b) = P( x_i = b | \pi_i = k) \]
  \[ e_i(b_1) + ... + e_i(b_M) = 1, \text{ for all states } i = 1...K \]
At each time step $t$, the only thing that affects future states is the current state $\pi_t$

\[
P(\pi_{t+1} = k \mid \text{“whatever happened so far”}) = P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) = P(\pi_{t+1} = k \mid \pi_t)
\]
A parse of a sequence

Given a sequence $x = x_1 \ldots x_N$,

A **parse** of $x$ is a sequence of states $\pi = \pi_1, \ldots, \pi_N$.
Given a sequence \( x = x_1 \ldots x_N \) and a parse \( \pi = \pi_1, \ldots, \pi_N, \)

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

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

\[
P(x_N, \pi_N | \pi_{N-1}) P(x_{N-1}, \pi_{N-1} | \pi_{N-2}) \cdots P(x_2, \pi_2 | \pi_1)
\]

\[
P(x_1, \pi_1) = P(x_N | \pi_N) P(\pi_N | \pi_{N-1}) \cdots P(x_2 | \pi_2) P(\pi_2 | \pi_1) P(x_1 | \pi_1) P(\pi_1) =
\]

\[
a_{0\pi_1} a_{\pi_1\pi_2} \cdots a_{\pi_{N-1}\pi_N} e_{\pi_1}(x_1) \cdots e_{\pi_N}(x_N)
\]
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

\[ \pi = \text{Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair} \]

(say initial probs \( a_{0\text{Fair}} = \frac{1}{2} \), \( a_{0\text{Loaded}} = \frac{1}{2} \))

\[
\frac{1}{2} \times P(1 \mid \text{Fair}) \times P(\text{Fair} \mid \text{Fair}) \times P(2 \mid \text{Fair}) \times P(\text{Fair} \mid \text{Fair}) \times \ldots \times P(4 \mid \text{Fair}) = \\
\frac{1}{2} \times \left(\frac{1}{6}\right)^{10} \times (0.95)^9 = 0.00000000521158647211 = 0.5 \times 10^{-9}
\]
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

$\text{Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?}$

$\frac{1}{2} \times P(1 \mid \text{Loaded}) \cdot P(\text{Loaded, Loaded}) \ldots \cdot P(4 \mid \text{Loaded}) =
\frac{1}{2} \times (\frac{1}{10})^8 \times (\frac{1}{2})^2 \times (0.95)^9 = \frac{1}{2} \times \frac{1}{1000000000} \times \frac{1}{4} \times (0.95)^9 = 0.00000000078781176215 = 7.9 \times 10^{-10}$

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 \( \pi = F, F, \ldots, F \)?

\[
\frac{1}{2} \times \left(\frac{1}{6}\right)^{10} \times (0.95)^9 = 0.5 \times 10^{-9}, \text{ same as before}
\]

What is the likelihood \( \pi = L, L, \ldots, L \)?

\[
\frac{1}{2} \times \left(\frac{1}{10}\right)^4 \times \left(\frac{1}{2}\right)^6 \times (0.95)^9 = .00000049238235134735 = 0.5 \times 10^{-7}
\]

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 $\text{Prob}[x \mid M]$ 

2. Decoding
   GIVEN a HMM M, and a sequence x,
   FIND the sequence $\pi$ of states that maximizes $P[x, \pi \mid M]$ 

3. Learning
   GIVEN a HMM M, with unspecified transition/emission probs., and a sequence x,
   FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize $P[x \mid \theta]$
Let’s not be confused by notation

\[ P( x \mid M ) : \text{ The probability that sequence } x \text{ was generated by the model} \]

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 \( \theta \) that maximizes \( P( x \mid \theta ) \)
Decoding

Given \( x = x_1 x_2 \ldots \ldots x_N \)

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

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

We can use dynamic programming!

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

= Probability of most likely sequence of states ending at state \( \pi_i = k \)
Decoding – main idea

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

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

What is \( V_k(i+1) \)?

From definition,

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

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

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

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

Input: \( x = x_1 \ldots x_N \)

Initialization:

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

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

Iteration:

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

\[
\text{Ptr}_j(i) = \arg\max_k a_{kj} V_k(i-1)
\]

Termination:

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

Traceback:

\[
\pi_N^* = \arg\max_k V_k(N)
\]

\[
\pi_{i-1}^* = \text{Ptr}_{\pi_i}(i)
\]
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.
Underflows are a significant problem

$$P[ x_1, ..., x_i, \pi_1, ..., \pi_i ] = a_{0\pi_1} a_{\pi_1 \pi_2} ... a_{\pi_i} e_{\pi_1}(x_1) ... e_{\pi_i}(x_i)$$

These numbers become extremely small – underflow

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

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

Let \(x\) be a sequence with a portion of ~ \(\frac{1}{6}\) 6’s, followed by a portion of ~ \(\frac{1}{2}\) 6’s...

\[x = 123456123456...12345 \text{ } 6626364656...1626364656\]

Then, it is not hard to show that optimal parse is (exercise):

\[\text{FFF} \text{...} \text{F} \text{ } \text{LLL} \text{...} \text{L}\]

6 nucleotides “123456” parsed as F, contribute \(.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}\)

parsed as L, contribute \(.95^6 \times (1/2)^1 \times (1/10)^5 = 0.4 \times 10^{-5}\)

“162636” parsed as F, contribute \(.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}\)

parsed as L, contribute \(.95^6 \times (1/2)^3 \times (1/10)^3 = 9.0 \times 10^{-5}\)
Given a HMM, we can generate a sequence of length n as follows:

Start at state \( \pi_1 \) according to prob \( a_{0\pi_1} \)
1. Emit letter \( x_1 \) according to prob \( e_{\pi_1}(x_1) \)
2. Go to state \( \pi_2 \) according to prob \( a_{\pi_1\pi_2} \)
3. \( \ldots \) until emitting \( x_n \)
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 $x = 123416231\textbf{626163}64616234161221341$

Most likely path: $\pi = \text{FF} \ldots \text{F}$
However: marked letters more likely to be L than unmarked letters
We will develop algorithms that allow us to compute:

\[ P(x) \] Probability of \( x \) given the model

\[ P(x_1...x_j) \] Probability of a substring of \( x \) given the model

\[ P(\pi_i = k \mid x) \] Probability that the \( i^{th} \) state is \( k \), given \( x \)

A more refined measure of which states \( x \) may be in
The Forward Algorithm

We want to calculate

\[ P(x) = \text{probability of } x, \text{ given the HMM} \]

Sum over all possible ways of generating \( x \):

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

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

\[ f_k(i) = P(x_1...x_i, \pi_i = k) \quad \text{(the forward probability)} \]
Define the forward probability:

\[ f_l(i) = P(x_1...x_i, \pi_i = l) \]

\[ = \sum_{\pi_1...\pi_{i-1}} P(x_1...x_{i-1}, \pi_1,..., \pi_{i-1}, \pi_i = l) \cdot e_l(x_i) \]

\[ = \sum_k \sum_{\pi_1...\pi_{i-2}} P(x_1...x_{i-1}, \pi_1,..., \pi_{i-2}, \pi_{i-1} = k) \cdot a_{kl} \cdot e_l(x_i) \]

\[ = e_l(x_i) \sum_k f_k(i-1) \cdot a_{kl} \]
The Forward Algorithm

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

**Initialization:**

\[
f_0(0) = 1 \\
f_k(0) = 0, \text{ for all } k > 0
\]

**Iteration:**

\[
f_i(i) = e_i(x_i) \sum_k f_k(i-1) a_{kl}
\]

**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:

\[ V_0(0) = 1 \]
\[ V_k(0) = 0, \text{ for all } k > 0 \]

Iteration:

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

Termination:

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

FORWARD

Initialization:

\[ f_0(0) = 1 \]
\[ f_k(0) = 0, \text{ for all } k > 0 \]

Iteration:

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

Termination:

\[ P(x) = \sum_k f_k(N) a_{k0} \]
We want to compute

\[ P(\pi_i = k \mid x), \]

the probability distribution on the \(i^{th}\) position, given \(x\)

We start by computing

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

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

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

\[ \text{Forward, } f_k(i) \quad \text{Backward, } b_k(i) \]
Define the backward probability:

\[ b_k(i) = P(x_{i+1} \ldots x_N \mid \pi_i = k) = \sum_{\pi_{i+1} \ldots \pi_N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N \mid \pi_i = k) \]

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

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

\[ = \sum_l e_l(x_{i+1}) a_{kl} b_l(i+1) \]
We can compute $b_k(i)$ for all $k, i$, using dynamic programming

**Initialization:**

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

**Iteration:**

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

**Termination:**

$$P(x) = \sum_l a_{0l} e_l(x_1) b_l(1)$$
What is the running time, and space required, for Forward, and Backward?

Time: $O(K^2N)$
Space: $O(KN)$

Useful implementation technique to avoid underflows

Viterbi: sum of logs
Forward/Backward: rescaling at each position by multiplying by a constant
Application: GenScan

Length distributions of human introns and initial, internal and terminal exons

(a) Introns
(b) Initial exons
(c) Internal exons
(d) Terminal exons
• N - intergenic region
• P - promoter
• F - 5’ untranslated region
• $E_{\text{snegl}}$ – single exon (intronless) (translation start -> stop codon)
• $E_{\text{init}}$ – initial exon (translation start -> donor splice site)
• $E_{\text{term}}$ – terminal exon (acceptor splice site -> stop codon)
• $I_k$ – phase k intron: 0 – between codons; 1 – after the first base of a codon; 2 – after the second base of a codon
GenScan

Forward (+) strand

Reverse (-) strand

promoter

poly-A

intergenic region

5'UTR

E0  E1  E2

Einit

Esingle

Eter

3'UTR

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

...
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

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 the DNA text file 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:**
- **Sequence name (optional):**
- **Print options:** Predicted peptides only

657
TMHMM: Prediction of transmembrane topology of protein sequence
Model consists of submodels for:
- helix core and cap regions (cytoplasmic and extracellular)
- cytoplasmic and extracellular loop regions
- globular domain regions
Trained form 160 proteins with experimentally determined transmembrane helices.

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

<table>
<thead>
<tr>
<th>Sequence</th>
<th>TMHMM-2.0</th>
<th>outside</th>
<th>inside</th>
<th>TMHelix</th>
<th>outside</th>
<th>inside</th>
<th>TMHelix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.04171</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>2</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>3</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>4</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>5</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>6</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>7</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

http://www.cbs.dtu.dk/services/TMHMM-2.0/
Validation for exons, introns, genes, protein parts etc

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 \times 2 contingency table. In effect, MCC is the correlation coefficient between the observed and predicted binary classifications.

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity (Sn)</td>
<td>( \frac{TP}{TP + FN} )</td>
</tr>
<tr>
<td>Recall</td>
<td></td>
</tr>
<tr>
<td>True positive rate (TPR)</td>
<td></td>
</tr>
<tr>
<td>Specificity (Sp)</td>
<td>( \frac{TN}{TN + FP} )</td>
</tr>
<tr>
<td>True negative rate (TNR)</td>
<td></td>
</tr>
<tr>
<td>Precision</td>
<td>( \frac{TP}{TP + FP} )</td>
</tr>
<tr>
<td>Positive predictive value (PPV)</td>
<td>( \frac{TP}{TP + FN} )</td>
</tr>
<tr>
<td>False positive rate (FPR)</td>
<td>( \frac{FP}{FP + TN} )</td>
</tr>
<tr>
<td>False discovery rate (FDR)</td>
<td>( \frac{FP}{FP + TP} )</td>
</tr>
<tr>
<td>Negative predictive value (NPV)</td>
<td>( \frac{TN}{TN + FN} )</td>
</tr>
<tr>
<td>Accuracy (ACC), ( Q_2 )</td>
<td>( \frac{TP + TN}{TP + FN + TN + FP} )</td>
</tr>
<tr>
<td>F1 score</td>
<td>( \frac{2TP}{2TP + FP + FN} )</td>
</tr>
<tr>
<td>F score</td>
<td></td>
</tr>
<tr>
<td>F measure</td>
<td></td>
</tr>
<tr>
<td>Matthews correlation coefficient (MCC)</td>
<td>( \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} )</td>
</tr>
</tbody>
</table>
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** \( e_{I_j}(a) \)
  - Emission prob.
    - Usually back ground distribution \( q_a \).
  - Transition prob.
    - \( M_j \) to \( I_j \), \( I_j \) to itself, \( I_j \) to \( M_{j+1} \)
  - Log-odds score for a gap of length \( k \) (no log-odds from emission)
    \[
    \log a_{M_jI_j} + \log a_{I_jM_{j+1}} + (k - 1) \log a_{I_jI_j}
    \]

- **Delete states**
  - No emission prob.
  - Cost of a deletion
    - \( M \rightarrow D \), \( D \rightarrow D \), \( D \rightarrow M \)
    - Each \( D \rightarrow D \) might be different

- **Parameters**
  - the probabilities values: trivial if many of independent alignment sequences are given.
    
    \[
    a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \quad e_k(a) = \frac{E_k(a)}{\sum_{a'} E_k(a')}
    \]
(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:

<table>
<thead>
<tr>
<th></th>
<th>bat</th>
<th>rat</th>
<th>cat</th>
<th>gnat</th>
<th>goat</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>-</td>
<td>A G</td>
</tr>
<tr>
<td>R</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>-</td>
</tr>
<tr>
<td>Eddy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Profile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hidden Markov models</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bioinformatics 1998;14:755-63; Lemoine et al. COVID-Align: Accurate online alignment of hCoV-19 genomes using a profile HMM, Bioinformatics 2020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reference for this section

- Chapter 10 Vol 2 (pag 185-225)
- Chapter 3
Section 8

Algorithms for DNA computing and storage

- Algorithm: Adleman DNA Computing
- Algorithm: Random access in large-scale DNA data storage
Adleman's DNA computation approach (1994) solved a Hamilton problem on 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

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  GTGACATGGTTAATAATCGCA
City 4->5  CGCCGGATACGGCGCTATCA
City 5->6  GATCTAAAGGTATGCATACG
Etc.

L. Adelman, Scientific American, pp. 54-61 (Aug 1998);
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)

Vertices:
- Vertex 1
- Vertex 2
- Vertex 3
- Vertex 4
- Vertex 5
- Vertex 6
- Vertex 7

Routes:
- V1 to V2
- V2 to V3
- V3 to V2
- V4 to V3
- V5 to V2
- V6 to V2
- V1 to V4
- V2 to V4
- V3 to V4
- V4 to V5
- V5 to V6
- V6 to V7
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

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.
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 encoded with a certain number of base pairs of DNA, knowing the length of the itinerary gives us the number of cities.
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.
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.
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 $1 \times 10^{14}$ 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 $2 \times 10^{19}$ 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.
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.
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.
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.
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.
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.
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.
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.
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


L. Adelman, Scientific American, pp. 54-61 (Aug 1998);

Published: 19 February 2018

Random access in large-scale DNA data storage


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
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 $\tau$.

**System State:**

<table>
<thead>
<tr>
<th></th>
<th>$t$</th>
<th>$t+\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$B$</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$C$</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

- $R_1$: $A + B \rightarrow C$
- $R_2$: $C \rightarrow A + B$
- $R_3$: $A \rightarrow B$
- $R_4$: $C \rightarrow$
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 $t + \tau$ 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.
A propensity function $a_i$ is associated to each reaction step. These probabilities 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 $a_0=\sum_{k=1}^{M}a_k$.

(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

\[
\sum_{k=1}^{\mu-1}a_k < r_2a_0 \leq \sum_{k=1}^{\mu}a_k,
\]

which is equivalent to choosing from reactions $[1, \ldots, M]$ with the $k$th reaction having probability $a_k/a_0$.

(7) Set $t=t+\tau$ and update the number of each molecular species according to reaction $\mu$.

(8) Return to step 2 or quit.

$\mu$ is the integer for which $\sum_{i=1}^{\mu-1}a_i < r_2a_0 \leq \sum_{i=1}^{\mu}a_i$. 

Details on step 6
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

\[
\begin{align*}
    h_0(x, c) &= \sum_{i=1}^{v} h_i(x, c_i).
\end{align*}
\]

It is clear that the time to the next reaction is \( \text{Exp}(n_0(x, c)) \), 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 \( h_0(x, c) = \sum_{i=1}^{v} h_i(x, c_i) \), the combined reaction hazard.
4. Simulate time to next event, \( t' \), as an \( \text{Exp}(h_0(x, c)) \) 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 \( j \)th column of the stoichiometry matrix \( S \).
8. Output \( x \) and \( t \).
9. If \( t < T_{\text{max}} \), 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, \(\mu\), \(\tau\), 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.

GillesPy2

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.
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.
Bioinformatics

(a) Discuss the space–time complexity of dynamic programming algorithms in sequence alignment.  

(b) Discuss with one example how to score a multiple sequence alignment.
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
Bioinformatics (PL)

Given the two DNA sequences: GCACCTT 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]
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]
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 \( TP = PP \cap AP \)

(vi) True Negative \( TN = PN \cap AN \)

(vii) False Negative \( FN = PN \cap AP \)

(viii) False Positive \( FP = PP \cap AN \)

(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 \( Sp = TP / (TP + FP) \)