At the core of life there is a sort of programming; the DNA sequence contains both the code for the structure of the 3d parts (usually proteins, programmed self assembly process) and the code that represents the manual of instructions -how much, where, when a certain part should be produced.

Bioinformatics is about algorithms and machine learning methods to identify the coding elements in the DNA sequences and characterise the parts.

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

5'-CCTGAGCCAACATTTGATGAA-3
3'-GGACTCGGTTGATAACTCTT-5

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

What is BioInformatics

Biology and Medicine

Machine learning

Algorithms

Great Data

Cool Questions

Powerful tools

Drug Discovery Pipeline

Accelerated by Bioinformatics

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

— Sydney Brenner —
Bioinformatics: a central position in medicine

Local
- Cohorts & Biobanks
- NGS Bioinformatics
- Health Data
- Imaging
- Metadata & Curation
- Multi-organ chips
- Nanosensors
- Synthetic biology

National
- Cybersecurity
- NGS Bioinformatics
- Citizen Science
- CRISPR
- Text Mining
- Lifestyle interventions
- Big Data Handling
- Artificial Intelligence

International
- Multimodal data analytics
- Databases & Data Sharing
- Bioinformatics
- NGS
- Epigenetics
- Adaptive Therapy
- Artificial Intelligence
- Computer simulation, personal avatars

NOW
1-5 years
5-10 years

NGS= next generation sequencing
DNA for genomic diagnostics

Impact on 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

https://www.genome.gov/sequencingcosts
High-performance computing

1979

Who has a computer?

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

today

Genome sequencing

2006

Whose genome has been sequenced?

- 1996: First bacterium (E. coli)
- 2001: Human reference genome
- 2007: First personal genomes
- 2019: Millions personal genomes
today
Garage genomics

Oxford nanopore

Bento Lab: A DNA laboratory for everybody

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

Pre-Order now!

Equipped by Bento Lab

JDRF is supporting Bento Lab to help find a cure for type 1 diabetes.

welcome to you
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></td>
<td>Real-time processing</td>
<td>Metadata analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Massive volumes</td>
<td></td>
<td></td>
<td></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 base pair take a couple of bits to encode (because you have to choose between G, A, T and C.

You have 46 chromosomes in each (autosomal) cell (3 billion base pairs, 2 meters long, 2nm thick, folded into a 6µm ball). If you teased out those 46 strands and placed 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.

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) — DNA in human population 20 million light years long (the Andromeda Galaxy is 2.5 Million light years).

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, they further estimated Earth's computational power: $10^{15}$ yottaNOPS (1024 Nucleotide Operations Per Seconds).
Genetic Code

Central Dogma

1. **DNA**
   - CCTGAGCCAACTATTGATGAA
   - GCACTCGGTTGATAACTACTT

2. **mRNA**
   - CCUGAGCCACACU AUUGAUGAA

3. **Protein**
   - PEPTIDE

**Transcription**

**Translation**

<table>
<thead>
<tr>
<th>1st position</th>
<th>2nd position</th>
<th>3rd position</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>C</td>
<td>AG</td>
</tr>
<tr>
<td>U</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>U</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>Ile</td>
<td>Thr</td>
<td>Asn</td>
</tr>
<tr>
<td>Ile</td>
<td>Thr</td>
<td>Asn</td>
</tr>
<tr>
<td>Ile</td>
<td>Thr</td>
<td>Lys</td>
</tr>
<tr>
<td>Met</td>
<td>Thr</td>
<td>Lys</td>
</tr>
<tr>
<td>Leu</td>
<td>Pro</td>
<td>His</td>
</tr>
<tr>
<td>Leu</td>
<td>Pro</td>
<td>Gln</td>
</tr>
<tr>
<td>Leu</td>
<td>Pro</td>
<td>Gln</td>
</tr>
<tr>
<td>Leu</td>
<td>Pro</td>
<td>Arg</td>
</tr>
<tr>
<td>Leu</td>
<td>Pro</td>
<td>Arg</td>
</tr>
<tr>
<td>Leu</td>
<td>Pro</td>
<td>Arg</td>
</tr>
</tbody>
</table>

- **Phe**
- **Ser**
- **Tyr**
- **Cys**
- **STOP**
- **STOP**
- **Trp**
- **Val**
- **Ala**
- **Asp**
- **Gly**
- **Val**
- **Ala**
- **Asp**
- **Gly**
- **Val**
- **Ala**
- **Asp**
- **Gly**
Healthy Individual

sequences in Fasta format

>gi|28302128|ref|NM_000518.4| Homo sapiens hemoglobin, beta (HBB), mRNA
ACATTGCTTCTGACACA ACT GTGTTCACTAGCAACCTCAAACAGACACC
ATG GTGCATCTGACTCCTGA

>gi|4504349|ref|NP_000509.1| beta globin [Homo sapiens]
MVHLPTEEKSAVTALWGKVNVDEVVGGEALGRLLVVYPWTQRSTFFESFGDSLTPDAVMGMPKVAHGGKKVLG
AFSDGLAHLNDLKGTFTALSHELCDKLHVDRENFRLLGNNVLVCLAHHFGEFTPPVQAAYQKVAGVAN
ALAHKYH
Individual with Sickle Cell Anemia

>gi|28302128|ref|NM_000518.4| Homo sapiens hemoglobin, beta (HBB), mRNA

ACATTTGCTTCTGACACAACTGTGGTTCAACTAGCAACCTCAAACAGACACC
ATGGTGCATCTGACTCCTGA

GGTGAAAGTCTCTGCCTGACACAACTGTGGTTCAACTAGCAACCTCAAACAGACACC
ATGGTGCATCTGACTCCTGA

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

MVHLTPV

EKSAVTALWGKVNVDEVGGAEALGRLLVVYPWTQRFFESFGDLSTPDAMGNPKVKAHGKKV

AFSDGLAHLDNLKGTFLATLSELHDCLHLVDPPNFRLLGNVLVCVLAHFFGKEFTPPVQAAYK

VAVANALAHKYH
Genes are activated or repressed by regulatory proteins which bind to gene flanking sequences (promoter) and are coded by the same or other genes.

Gene and protein interactions as graphs

[Diagram A: Protein binds to DNA, RNA polymerase reads the information of the gene, Promoter, Gene]

[Diagram B: Regulatory Element, Protein binds to DNA, New protein, RNA polymerase]

[Diagram C: Regulatory Element, Gene, Transcription Factor, New protein]

[Diagram D: Regulatory Element, Gene, Transcription Factor, RNA polymerase]
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)
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 monitor the activity of all the genes in different conditions (gene expression).
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.
Scales of electronic and bio devices

(proteins inside a bacterium)

Bacterium

Human chromosone.

(a) NAND gate layout geometry.

$\lambda = 0.25 \text{ micron in Pentium II}$

Scale in $\lambda$

0 1 2 3 4 5 6

1 micron
The network level: can you spot the difference?
Nature is programmed for self-assemble; Bioinformatics is needed to identify the key elements

- DNA, RNA and proteins can:
  - Organize 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
Macroscale

IKEA: not self assembly
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.

5. 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.
Cells versus Computers

- Base-4 (ACGT)
- DNA
- Bases
- Codons (triplets of bases for each amino acid)
- Genetic Code (translate codons into amino acids)
- Gene/Protein
- Chromosome
- Genome Size

- Base-2 (101010)
- Magnetic tape/Disk
- Bits/Transistors
- Bytes
- Instruction Set

- File, Program
- Hard Disk
- Disk Capacity
### Cells versus Computers

<table>
<thead>
<tr>
<th>Biology</th>
<th>Computer science</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>
If you want to know more about biology

A free book is this: cell biology by the numbers
http://book.bionumbers.org/

- Genetics for Computer Scientists
- Molecular Biology for Computer Scientists:
  http://tandy.cs.illinois.edu/Hunter_MolecularBiology.pdf
- Biology and Computers: A lesson in what is possible
  https://ethw.org/
  https://www.wehi.edu.au/wehi-tv/
General references for course

Partly based on book: Compeau and Pevzner Bioinformatics algorithms (chapter 3,5,7-10 chapter).

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

No biology in the exam questions (You need to know only the reason of the algorithms).
Alignment 1

DNA Computing/storage information

Hidden Markov Models 6

Clustering 5

Phylogeny 2

Genome sequencing 3

Genome Assembly 4

Phylogeny is a sort of clustering

phylogeny to improve multi sequence alignment

identify all elements

close links between genome sequencing and assembly

not only strings: we use clustering for gene expression represented as matrix data (usually genes x conditions or genes x patients)

more species sequencing improve

Structure of the course
• how to align two sequences?
• Trees (what is the relationships of multiple sequences and what has to do with species evolutionary history)
• Genome sequence (how to analyse a genome)
How Do We Compare Biological Sequences?

- From Sequence Comparison to Biological Insights
- The Alignment Game and the Longest Common Subsequence
- Dynamic Programming and Backtracking Pointers
- From Global to Local Alignment
- Penalising Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Nussinov folding algorithm (RNA 2dimensional folding)
Summary for alignment lectures

Algorithms in this lecture: Longest common subsequence, Needleman-Wunsch, Smith-Waterman, Affine gap, Hirschberg, Nussinov RNA folding. Typical tasks: align genome and protein sequences; we want to detect all differences at the single base to block of bases levels. In the RNA folding problem we want to align a molecule with itself.

Data: DNA or protein (amino acid) sequences considered as strings; input: two strings (Nussinov accepts one string in input and search for internal similarities). Output: a set of aligned positions that makes easy the identification of conserved patterns. Note that each string belongs to a double helix so the information could be related to one of the two strands and read in one or the opposite orientation.

Many events (mutations) could lead to sequence changes. Therefore the conservation of a substring between two strings may suggest to a crucial functional role for the cell. The dynamic programming algorithms could be used to detect similarities within a single string (last section of the lecture). This is particularly useful to find the folding of RNA molecules (in a RNA molecule the T is replaced by U).

Main question in this lecture: how similar are these two sequences?
Alignment of two sequences is a two-row matrix:

1\textsuperscript{st} row: symbols of the 1\textsuperscript{st} sequence (in order) interspersed by “-”
2\textsuperscript{nd} row: symbols of the 2\textsuperscript{nd} sequence (in order) interspersed by “-”
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 $w$:

$v : \quad A \ T \ G \ T \ T \ A \ T \quad m = 7$

$w : \quad A \ T \ C \ G \ T \ A \ C \quad n = 7$

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

<table>
<thead>
<tr>
<th>Letters of $v$</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>Letters of $w$</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

\[
\begin{align*}
v: & \quad 1 \leq i_1 < i_2 < \ldots < i_t \leq m \\
\text{and a sequence of positions in} \quad w: & \quad 1 \leq j_1 < j_2 < \ldots < j_t \leq n
\end{align*}
\]

such that \( i_t \)-th letter of \( v \) equals to \( j_t \)-th letter of \( w \) and \( t \) is maximal.
Longest Common Subsequence

i coords: 0 1 2 2 3 3 4 5 6 7 8

elements of v
A T -- C -- T G A T C

elements of w
-- T G C A T -- A -- C

j coords: 0 0 1 2 3 4 5 5 6 6 7

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

Matches shown in red
positions in v: 2 < 3 < 4 < 6 < 8
positions in w: 1 < 3 < 5 < 6 < 7

Every common subsequence is a path in 2-D grid
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 $w$

$V = \text{ATATATATA}$

$W = \text{TATATATA}$

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 $w$

$V = \text{-ATATATATA}$

$W = \text{TATATATA-}$

Edit distance:

$d(v, w) = 2$

Computing edit distance is a non-trivial task
Edit Distance: Example

TGCATAT $\rightarrow$ ATCCGAT in 4 steps

- TGCATAT $\rightarrow$ (insert A at front)
- ATGCATAT $\rightarrow$ (delete 6\textsuperscript{th} T)
- ATGCATA $\rightarrow$ (substitute G for 5\textsuperscript{th} A)
- ATGCCTTA $\rightarrow$ (substitute C for 3\textsuperscript{rd} G)
- ATCCGAT (Done)
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:

$$s_{i,j} = \text{MAX}$$

$$s_{i-1,j} + 0$$
$$s_{i,j-1} + 0$$
$$s_{i-1,j-1} + 1, \text{ if } v_i = w_j$$

Every Path in the Grid Corresponds to an Alignment
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).
Dynamic Programming Recurrence for the Alignment Graph

\[ s_{i,j} = \max \]

\[
\begin{align*}
    s_{i-1,j} - \sigma \\
    s_{i,j-1} - \sigma \\
    s_{i-1,j-1} + 1, \text{ if } v_i = w_j \\
    s_{i-1,j-1} - \mu, \text{ if } v_i \neq w_j
\end{align*}
\]
All genomes are littered with repeats so alignment of large sequences is difficult.

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

**Figure:** Type and frequency of mutations (replacements, insertions, deletions) in the human genome per generation; mutations change single DNA bases (SNP polymorphism) or rearrange DNA strings at different length scales. In sequence alignment we compare sequences that are different because of mutations.
Towards an algorithm to align biological sequences (note I am using a DIFFERENT NOTATION!)

Notice three possible cases:

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

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

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

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

$$F(i,j) = F(i-1, j) - d, \text{ if not}$$

$$F(i,j) = F(i, j-1) - d$$
• How do we know which case is correct?

**Inductive assumption:**

\[ F(i, j-1), F(i-1, j), F(i-1, j-1) \] are optimal

Then,

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

Where \( F(x_i, y_j) = m, \) if \( x_i = y_j; \) \(-s, \) if not
• The **Global Alignment Problem** tries to find the longest path between vertices 
  
  \[(0,0)\] \text{ and } \[(n,m)\] in the edit graph.

• The **Local Alignment Problem** tries to find the longest path among paths 
  between **arbitrary vertices** \((i,j)\) and \((i',j')\) in the edit graph.

**Global Alignment**

\[
\begin{array}{c}
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC \\
\text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \\
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C
\end{array}
\]

**Local Alignment**—better alignment to find highly conserved segments

\[
tccCAGTTATGTCAGgggacacgcagcatgcagagac \\
\text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \text{| } \\
aattgccgccgtcgttttcagCAGTTATGTCAGatc
\]
local alignment to detect regulatory sites
Global Alignment

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

- **Input:** Strings $v$ and $w$ as well as a matrix $\text{score}$.
- **Output:** An alignment of $v$ and $w$ whose alignment score (as defined by the scoring matrix $\text{score}$) 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
   \( \text{For each } j = 1 \ldots M \)
   \( \text{For each } j = 1 \ldots N \)
   \[ F(i, j) = \max \begin{cases} F(i-1, j) - d & \text{[case 1]} \\ F(i, j-1) - d & \text{[case 2]} \\ F(i-1, j-1) + s(x_i, y_j) & \text{[case 3]} \end{cases} \]

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

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:

```
CTATCACCTGACCTCAGGCAGATGGCCCTTCCCGGC
GCGAGTTCATCTATCAC--GACCGC--GCTCG--
```

Changes:

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

2. **Termination**
   
   \( F_{\text{OPT}} = \max \left\{ \max_i F(i, N), \max_j F(M, j) \right\} \)
Can we use a similar algorithm to align entire genomes?
Local Alignment = Global Alignment in a subrectangle

local alignment to detect regulatory sites
Local Alignment Problem: Find the highest-scoring local alignment between two strings.

- **Input:** Strings \( v \) and \( w \) as well as a matrix \( \text{score} \).

- **Output:** Substrings of \( v \) and \( w \) whose global alignment (as defined by the matrix \( \text{score} \)), is maximal among all global alignments of all substrings of \( v \) and \( w \).
Idea: Ignore badly aligning regions: Modifications to Needleman-Wunsch

\[ \text{e.g. } x = \text{aaaacccccggg} \]
\[ y = \text{cccgggaaccaacc} \]

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

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

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

David Waterman
Which Alignment is Better?

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

\[
\text{GCC--C-A} \text{GT---T} \text{ATGT--CAGGGGCGACG--A-G} \text{CATGCAGA-GCGCCG} \text{C-GT} \text{CGT--T} \text{TTCAG----CA-GTTATG--T-CAGAT}
\]

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

\[
\text{---G-----C-----C---CAGTTATGTCAGGGGCACGAGCATGCAGA-GCCGCCG} \text{TCGTTTTCAGCAGTTATGTCAG} \text{----A-----T-----}
\]

the local alignment detects a biological finding: two genes are regulated by the same protein
• 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:

  \[
  \begin{array}{ccc}
  \text{two gaps} & \text{GATCCAG} & \text{GATCCAG} \\
  \text{(lower score)} & \text{GA–C–AG} & \text{GA––CAG} \\
  \text{a single gap} & \text{GA–C–AG} & \text{GA––CAG} \\
  \text{(higher score)} & \text{or maybe 2 events} \\
  \end{array}
  \]
Mismatches and Indel Penalties

#matches – \( \mu \cdot \#\text{mismatches} – \sigma \cdot \#\text{indels} \)

\[
\begin{align*}
A & \rightarrow G & T & \rightarrow T & A & \rightarrow T & A \\
A & \rightarrow T & C & \rightarrow G & T & \rightarrow C & – C \\
+1+1-2+1+1-2-3-2-3=-7
\end{align*}
\]

<table>
<thead>
<tr>
<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>–(\mu)</td>
<td>–(\mu)</td>
<td>–(\mu)</td>
</tr>
<tr>
<td>C</td>
<td>–(\mu)</td>
<td>+1</td>
<td>–(\mu)</td>
<td>–(\mu)</td>
</tr>
<tr>
<td>G</td>
<td>–(\mu)</td>
<td>–(\mu)</td>
<td>+1</td>
<td>–(\mu)</td>
</tr>
<tr>
<td>T</td>
<td>–(\mu)</td>
<td>–(\mu)</td>
<td>–(\mu)</td>
<td>+1</td>
</tr>
<tr>
<td>–</td>
<td>–(\sigma)</td>
<td>–(\sigma)</td>
<td>–(\sigma)</td>
<td>–(\sigma)</td>
</tr>
</tbody>
</table>

Scoring matrix

Even more general scoring matrix
How to compare amino acids: scoring matrices

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>S</th>
<th>T</th>
<th>P</th>
<th>A</th>
<th>G</th>
<th>N</th>
<th>D</th>
<th>E</th>
<th>Q</th>
<th>H</th>
<th>R</th>
<th>K</th>
<th>M</th>
<th>I</th>
<th>L</th>
<th>V</th>
<th>F</th>
<th>Y</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</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>
<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>
<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>T</td>
<td>-2</td>
<td>1</td>
<td>3</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>
<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>
<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>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>
<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>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>
<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>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>
<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>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>
<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>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>
<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>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>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td>1</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>-3</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td>-2</td>
<td>-2</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td>-2</td>
<td>-2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>-4</td>
<td>-3</td>
<td>-3</td>
<td>-5</td>
<td>-5</td>
<td>-5</td>
<td>-4</td>
<td>-6</td>
<td>-5</td>
<td>-5</td>
<td>-2</td>
<td>4</td>
<td>-5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>-5</td>
<td>-3</td>
<td>-5</td>
<td>-2</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
<td>0</td>
<td>-4</td>
<td>-4</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>7</td>
<td>10</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>
<td>-5</td>
<td>-3</td>
<td>2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-2</td>
<td>-6</td>
<td>0</td>
<td>0</td>
<td>17</td>
</tr>
</tbody>
</table>

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

Affine gap penalty for a gap of length \( k \): \( \sigma + \varepsilon \cdot (k-1) \)

\( \sigma \) - the gap opening penalty

\( \varepsilon \) - the gap extension penalty

\( \sigma > \varepsilon \), since starting a gap should be penalized more than extending it.
• Thinking on 3 levels

bottom level
(insertions)

middle level
(matches/mismatches)

upper level
(deletions)
How can we emulate this path in the 3-level?

\[
\text{lower}_{i,j} = \max \left\{ \text{lower}_{i-1,j} - \varepsilon, \text{middle}_{i-1,j} - \sigma \right\}
\]

\[
\text{middle}_{i,j} = \max \left\{ \text{middle}_{i-1,j-1} + \text{score}(v_i, w_j), \text{upper}_{i,j}, \text{lower}_{i,j} \right\}
\]

\[
\text{upper}_{i,j} = \max \left\{ \text{upper}_{i,j-1} - \varepsilon, \text{middle}_{i,j-1} - \sigma \right\}
\]
• Modelling Affine Gap Penalties by Long Edges

double gap: 2 events

double gap: 1 event
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:**
\[
F(i, j) = \max \left\{ F(i-1, j-1) + s(x_i, y_j), \max_{k=0 \ldots i-1} F(k,j) - \gamma(i-k), \max_{k=0 \ldots j-1} F(i,k) - \gamma(j-k) \right\}
\]

**Termination:** same

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

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

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

<table>
<thead>
<tr>
<th>gap</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>extend</td>
</tr>
</tbody>
</table>

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

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

**Initialization:**

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

**Iteration:**

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

\[ G(i, j) = \max\left\{ G(i - 1, j - 1) - e, G(i, j - 1) - e \right\} \]

**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:** $\# \text{gaps}(x, y) < k(N)$ (say $N > M$)

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

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

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

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

**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>
</tbody>
</table>

**Match = 2, Mismatch = -1, Gap = -1**

A match occurs at position 0, with a score of 2.

-1 indicates a mismatch at the next position.
<table>
<thead>
<tr>
<th>match=2</th>
<th>mismatch=-1</th>
<th>gap=-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

ACGCTG

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

**Alignment:**

```
-----
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>C</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>
</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
<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><strong>0</strong></td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A</strong></td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>G</strong></td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Match: 2
Mismatch: -1
Gap: -1

AC
-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>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</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>

**ACG**
<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 1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>A 2</td>
<td>-2</td>
<td></td>
<td></td>
<td></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

ACGC
---C

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

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

ACGCTG- -C-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>
</tbody>
</table>

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

**ACGCTG-**

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

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

<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>T 1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T 4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Local Alignment Example

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

match = 1
mismatch = -1
gap = -1

\[
\begin{array}{l}
\text{match} = 1 \\
\text{mismatch} = -1 \\
\text{gap} = -1 \\
\end{array}
\]
Local Alignment Example

\begin{align*}
\text{match} &= 1 \\
\text{mismatch} &= -1 \\
\text{gap} &= -1
\end{align*}

\[y = \text{TAATA} - \]
\[x = \text{TACTAA}\]

\begin{tabular}{c|ccccccc}
\hline
 & T & A & C & T & A & A \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
T & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
A & 0 & 0 & 2 & 0 & 0 & 2 & 1 \\
A & 0 & 0 & 1 & 1 & 0 & 1 & 3 \\
T & 0 & 0 & 0 & 0 & 2 & 0 & 1 \\
A & 0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
\end{tabular}
Local Alignment Example

match=1  
mismatch=-1  
gap=-1

\[
y = \text{---TAA}TA
\]

\[
x = \text{TACTAA--}
\]

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

\[
y = \text{---TAA}TA
\]

\[
x = \text{TACTAA--}
\]
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($i$)

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

\[
\begin{array}{ccc}
\text{0} & \text{m/2} & \text{m} \\
\text{} & \text{} & \text{} \\
\text{} & \text{..} & \text{..} \\
\text{..} & \text{..} & \text{..} \\
\text{..} & \text{..} & \text{..} \\
\end{array}
\]

store $\text{prefix}(i)$ column
Computing Suffix\( (i) \)

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

store \( suffix(i) \) column
Length(i) = Prefix(i) + Suffix(i)

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

Only two columns of scores are saved at any given time

memory for column 1 is used to calculate column 3

memory for column 2 is used to calculate column 4
Crossing the Middle Line

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

Define

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

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

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

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

middle column
(middle=\#columns/2)

A C G G A A

A A T T C A A
Middle Node of the Alignment

(a node where an optimal alignment path crosses the middle column; note that different longest paths may have different middle nodes, and a given longest path may have more than one middle node.)
Divide and Conquer Approach to Sequence Alignment

**AlignmentPath**(*source, sink*)

find *MiddleNode*
Divide and Conquer Approach to Sequence Alignment

**AlignmentPath** *(source, sink)*

find *MiddleNode*

**AlignmentPath** *(source, MiddleNode)*
Divide and Conquer Approach to Sequence Alignment

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

**AlignmentPath**(source, MiddleNode)

**AlignmentPath**(MiddleNode, sink)

The only problem left is how to find this middle node in **linear space**!
Computing Alignment Score in Linear Space

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

Finding the **length of the longest path** in the alignment graph **does not require** storing any backtracking pointers – $O(n)$ memory.
Recycling the Columns in the Alignment Graph
Can We Find the Middle Node without Constructing the Longest Path?

\[ i\text{-path} \quad \text{– a longest path among paths that visit the } i\text{-th node in the middle column} \]

\[ 4\text{-path} \quad \text{that visits the node } (4,\text{middle}) \]
\[ \text{In the middle column} \]
Can We Find The Lengths of All $i$-paths?

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

$\text{length}(i) = \text{fromSource}(i) + \text{toSink}(i)$
Computing *FromSource* and *toSink*

**fromSource(i)**

Computing *FROMSOURCE*(i) for all i can be done in $O(n)$ space and $O(n \cdot m/2)$ time. Computing *TOSINK*(i) for all i can also be done in $O(n)$ space and $O(n \cdot m/2)$ time; this requires reversing the direction of all edges and treating the sink as the source. Instead of reversing the edges, we can reverse the strings $v = v_1 \ldots v_n$ and $w = w_1 \ldots w_m$ and find $s_{n-i,m-middle}$ in the alignment graph for $v_n \ldots v_1$ and $w_m \ldots w_1$. 

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

In total, we can compute all values \( \text{LENGTH}(i) = \text{FROMSOURCE}(i) + \text{TOSINK}(i) \) in linear space with runtime proportional to \( n \cdot m/2 + n \cdot m/2 = n \cdot m \), which is the total area of the alignment graph.
Laughable Progress: $O(nm)$ Time to Find **ONE** Node!

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

\[
\text{area}/4 + \text{area}/4 = \text{area}/2
\]

How much time would it take to conquer 2 subproblems?
Laughable Progress: $O(nm+nm/2)$ Time to Find THREE Nodes!

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

$\text{area}/8 + \text{area}/8 + \text{area}/8 + \text{area}/8 = \text{area}/4$

How much time would it take to conquer 4 subproblems?
\(O(nm+nm/2+nm/4)\) Time to Find **NEARLY ALL** Nodes!

How much time would it take to conquer ALL subproblems?

\[
\text{area} + \frac{\text{area}}{2} + \frac{\text{area}}{4} + \frac{\text{area}}{8} + \frac{\text{area}}{16} + \ldots + \frac{\text{area}}{\text{area}} < 2\cdot\text{area}
\]
The Middle Edge (just to save memory a little bit more)

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

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

- **Input:** Two strings and matrix *score*.
- **Output:** A middle edge in the alignment graph of these strings (as defined by the matrix *score*).
A middle edge (shown in bold) starts at the middle node (shown as a black circle). The optimal path travels inside the first highlighted rectangle, passes the middle edge, and travels inside the second highlighted rectangle afterwards.
We can eliminate the remaining parts of the alignment graph, which takes up over half of the area formed by the graph, from further consideration.

Finding middle edges (shown in bold) within previously identified rectangles.
Recursive \textbf{LinearSpaceAlignment}

\begin{verbatim}
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)
\end{verbatim}
A: space complexity

B: time complexity

Total Time: $area + area/2 + area/4 + area/8 + area/16 + \ldots$
Can we compute the edit distance faster than $O(nm)$?

- yes: The Four Russians Technique
- Arlazarov, V.; Dinic, E.; Kronrod, M.; Faradžev, I.
- The basic idea is to precompute parts of the computation involved in filling out the dynamic programming table.
- time $O(n^2/\log n)$
- Assume the block-function $b(A, B, C, X[i+1..i+t], Y[j+1..j+t])$ has been precomputed for all possible inputs.
- Article in Russian, easier to look at Aho, Alfred V.; Hopcroft, John E.; Ullman, Jeffrey D. (1974), The design and analysis of computer algorithms, Addison-Wesley
Self Alignment

Pairing rules:
C-G
A-U
(in RNA T is replaced by U)

https://www.sciencedirect.com/science/article/pii/S0958166916301082#fig0020
dot-bracket representation for a pseudoknot free structure, as well as the extended pseudoknot representation for a structure containing a pseudoknot.
usually the more the links the more the binding energy. Above: Ensemble of all possible structures for a given RNA sequence, with the corresponding binding energy. The potential energy is negative because you need to give energy to break the links (i.e. the structure), for example by heating.
RNA Secondary Structure

Secondary Structure:
- Set of paired positions on interval $[i,j]$
- This tells which bases are paired in the subsequence from $x_i$ to $x_j$

Every optimal structure can be built by extending optimal substructures.

Suppose we know all optimal substructures of length less than $j-i+1$.
The optimal substructure for $[i,j]$ must be formed in one of four ways:
1. $i,j$ paired
2. $i$ unpaired
3. $j$ unpaired
4. combining two substructures

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

![Diagrams of RNA secondary structures](image)
RNA Secondary Structure: The Nussinov Folding Algorithm

Example: GGGAAAUCC

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

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

Starting with all subsequences of length 2, to length \( L \):

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

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

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

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

<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></td>
<td></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>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A A
\[\xrightarrow{A \to U} \] A U
\[\xrightarrow{G \to C} \] G C
\[\xrightarrow{G \to C} \] G C

Diagram of RNA secondary structure with scores and subsequence folding results.
**Nussinov Folding Algorithm:**

After scores for subsequences of length 4

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

\[
\max_{i \leq k < j} [\gamma(i, k) + \gamma(k+1, j)]
\]

<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></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>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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></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>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

\[ \begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]
### Nussinov Folding Algorithm:

After scores for subsequences of length 6

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

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

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

<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></td>
<td></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></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></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></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \gamma(i, j) = \gamma(i + 1, j) \quad \gamma(i, j - 1) \quad \gamma(i + 1, j - 1) + \delta(i, j) \quad \max_{i \leq k < j} [\gamma(i, k) + \gamma(k + 1, j)] \]
Nussinov Folding Algorithm

After scores for subsequences of length 8

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

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 2 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & \\
0 & 0 & 0 & 1 & 1 & 1 & \\
0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & \\
0 & 0 & 0 & \\
\end{array}
\]


**Nussinov Folding Algorithm**

After scores for subsequences of length 9

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

<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>1</td>
<td>2</td>
<td>3</td>
<td></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>3</td>
<td></td>
</tr>
<tr>
<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>
<td></td>
</tr>
<tr>
<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>
<td></td>
</tr>
<tr>
<td>0</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>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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>0</td>
<td>1</td>
<td>2</td>
<td>2</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>1</td>
<td>1</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>1</td>
<td>1</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>1</td>
<td>1</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>1</td>
<td>1</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>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
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:
\[
\delta(i, j) = 1 \text{ for all RNA Watson-Crick base-pairs including G-U else } \delta(i, j) = 0.
\]

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 $i \geq j$ continue;
else if $\gamma(i + 1, j) = \gamma(i, j)$ push $(i + 1, j)$;
else if $\gamma(i, j - 1) = \gamma(i, j)$ push $(i, j - 1)$;
else if $\gamma(i + 1, j - 1) + \delta_{i, j} = \gamma(i, j)$:
  - record $i, j$ base pair.
  - push $(i + 1, j - 1)$.
else for $k = i + 1$ to $j - 1$: if $\gamma(i, k) + \gamma(k + 1, j) = \gamma(i, j)$:
  - push $(k + 1, j)$.
  - push $(i, k)$.
  - break.

current record stack

1, 9
1, 8
1, 8 1, 4 5, 8
1, 4 1, 4 2, 3 5, 8
2, 3 2, 3 3, 2 5, 8
3, 2 5, 8
5, 8 5, 8 6, 7
6, 7 6, 7 7, 6
7, 6
Figure 1 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 GGGAAUCC after initialization, after the recursive fill, and after an optimal structure with three base pairs has been traced back.
RNA Secondary Structure: The Nussinov Folding Algorithm


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

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

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.
Initialise:
- Sequence: GGGAAUCC, length \( L \) = 9.
- \( N_{i,i-1} = 0 \) for \( i = 2 - L \)
- \( N_{i,i} = 0 \) for \( i = 1 - L \)

\[
\begin{array}{cccccccc}
G & 0 & & & & & & & \\
G & 0 & 0 & & & & & & \\
G & 0 & 0 & & & & & & \\
A & 0 & 0 & & & & & & \\
A & 0 & 0 & & & & & & \\
A & 0 & 0 & & & & & & \\
U & & & & & & & & \\
C & & & & & & & & \\
C & & & & & & & & \\
\end{array}
\]

Recursion:
- \( \rho(i,j) = 1 \) if \( s_i \) and \( s_j \) are complementary, otherwise \( \rho(i,j) = 0 \).

\[
\begin{array}{cccccccc}
G & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
G & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
G & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
A & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
A & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
A & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
U & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
N_{i,j} = \max \begin{cases} 
N_{i+1,j-1} + \rho(i,j), & \text{i, j pair} \\
N_{i+1,j}, & \text{i unpaired} \\
N_{i,j-1}, & \text{j unpaired} \\
\max_{i<k<j} [N_{i,k} + N_{k+1,j}], & \text{bifurcation}
\end{cases}
\]

Traceback:
Phylogeny

species tree by Darwin

Ancestral Node
or ROOT of the Tree

Branches or Lineages

Internal Nodes

Terminal Nodes

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

- tree of life based on mitochondrial sequences
- tracing influenza strain variations
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)
**EXAMPLE**: Phylogenetic-inspired techniques for reverse engineering and detection of malware families

For example, given an execution trace of instructions,

```assembly
push ebp
mov ebp, esp
mov eax, dword ptr [ebp-0x4]
jmp +0x14
```

it is abstracted as a sequence of mnemonics, i.e.

```assembly
push, mov, mov, jmp
```

ignoring the operands. Each mnemonic is then mapped to a unique alphabet-pair, e.g. `mov` = `MO`, `push` = `PH`, `jmp` = `JM`. The resulting sequence is thus `PHMOMOJM`.

Sequence alignment (dbg: with debugging symbols, def: default settings, spd: optimised for speed). (a) Before alignment. (b) After alignment using an identity substitution matrix. (c) After alignment using a substitution matrix.
Trees and Phylogeny
Outline

- Transforming Distance Matrices into Evolutionary Trees
- Toward an Algorithm for Distance-Based Phylogeny Construction
- 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
### 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>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>TCGAAAGCAT</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
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>AGTACGCCT</td>
<td>0</td>
</tr>
<tr>
<td>Human</td>
<td>ATGTAAGACT</td>
<td>3</td>
</tr>
<tr>
<td>Seal</td>
<td>TCGAGAGCAGC</td>
<td>6</td>
</tr>
<tr>
<td>Whale</td>
<td>TCGAAAGGCAT</td>
<td>4</td>
</tr>
</tbody>
</table>
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>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>ATgtAAAGACT</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>TCgAAGGCAC</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?
Trees

Tree: Connected graph containing no cycles.

Leaves (degree = 1): present-day species

Internal nodes (degree ≥ 1): ancestral species
**Rooted tree:** one node is designated as the root (most recent common ancestor)
Distance-Based Phylogeny

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

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

![Diagram showing tree structure with distances]
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?
Exercise Break: Try fitting a tree to the following 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>
No Tree Fits a Matrix

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

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

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 can be fit to the same 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>

![Graph showing relationships between Chimp, Human, Seal, and Whale with distances labeled as 0.5, 1, 1.5, and 3.]
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>
An Idea for Distance-Based Phylogeny

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

**Theorem**: Every simple tree with at least two nodes has at least one pair of neighboring leaves.
An Idea for Distance-Based Phylogeny

<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

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

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

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

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

\[
d_{\text{Seal},m} = \frac{D_{\text{Seal,Chimp}} + D_{\text{Seal,Whale}} - D_{\text{Whale,Chimp}}}{2}
\]
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>

\[ d_{\text{Seal},m} = 2 \]
### 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>
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](attachment:image.png)
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>$m$</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:

- Chimp
- Human
- Seal
- Whale

Distances:

- Chimp to Seal: 4
- Chimp to Whale: 5
- Human to Seal: 2
- Human to Whale: 0
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>
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>

\[
d_{\text{Chimp},a} = \left( D_{\text{Chimp},m} + D_{\text{Chimp,Human}} - D_{\text{Human},m} \right) / 2
\]
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>

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

- Chimp
- Human
- Whale
- Seal

Distances:
- Chimp to Human: 3
- Chimp to Whale: 1
- Human to Whale: 2
- Chimp to Seal: 1
- Human to Seal: 2
- Whale to Seal: 0
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: A graph showing the relationships between Chimp, Human, Seal, and Whale, with distances indicated on the edges.
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>
An Idea for Distance-Based Phylogeny

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

\[
\begin{array}{cccc}
  & i & j & k & l \\
  i & 0 & 13 & 21 & 22 \\
  j & 13 & 0 & 12 & 13 \\
  k & 21 & 12 & 0 & 13 \\
  l & 22 & 13 & 13 & 0 \\
\end{array}
\]
What Was Wrong With Our Algorithm?

<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?

<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>

![Graph diagram with nodes i, j, k, l and edges with weights: i to j (2), i to k (11), j to k (4), j to l (2), k to l (7), and k to i (6).]
What Was Wrong With Our Algorithm?

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

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

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.
From Neighbors to Limbs

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

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

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

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

\[ d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2 \]
From Neighbors to Limbs

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

\[ d_{k,m} = \frac{[ (d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m}) ]}{2} \]
\[ d_{k,m} = \frac{(d_{i,k} + d_{j,k} - d_{i,j})}{2} \]
\[ d_{k,m} = \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2} \]
\[ d_{i,m} = D_{i,k} - \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2} \]
\[ d_{i,m} = \frac{(D_{i,k} + D_{i,j} - D_{j,k})}{2} \]
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 $(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{j,k})/2$ 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>

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

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

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

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

\[
\frac{(D_{\text{chimp, whale}} + D_{\text{chimp, seal}} - D_{\text{whale, seal}})}{2} = \frac{(6 + 4 - 2)}{2} = 4
\]
### 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>

\[
(D_{\text{human}, \text{chimp}} + D_{\text{chimp}, \text{seal}} - D_{\text{human}, \text{seal}}) / 2 = (3 + 6 - 7) / 2 = 1
\]
\[
(D_{\text{human}, \text{chimp}} + D_{\text{chimp}, \text{whale}} - D_{\text{human}, \text{whale}}) / 2 = (3 + 4 - 5) / 2 = 1
\]
\[
(D_{\text{whale}, \text{chimp}} + D_{\text{chimp}, \text{seal}} - D_{\text{whale}, \text{seal}}) / 2 = (6 + 4 - 2) / 2 = 4
\]
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>
Additive Phylogeny In Action

**Matrix** $D$

<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>

**Tree** $\text{TREE}(D)$
## Additive Phylogeny In Action

1. Pick an arbitrary leaf $j$. 

<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

\[ D \]

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

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

2. Compute its limb length, \( \text{LimbLength}(j) \).
Additive Phylogeny In Action

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

3. Subtract \( \text{LimbLength}(j) \) from each row and column to produce \( D^{\text{bald}} \) in which \( j \) is a \textbf{bald limb} (length 0).
4. Remove the $j$-th row and column of the matrix to form the $(n - 1) \times (n - 1)$ matrix $D_{\text{trim}}$. 

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

Additive Phylogeny In Action

$$D_{\text{trim}}$$

$$\begin{array}{cccc}
  i & j & k & l \\
  i & 0 & 11 & 21 & 22 \\
  j & 11 & 0 & 10 & 11 \\
  k & 21 & 10 & 0 & 13 \\
  l & 22 & 11 & 13 & 0 \\
\end{array}$$
Additive Phylogeny In Action

6. Identify the point in $Tree(D_{\text{trim}})$ where leaf $j$ should be attached.
### 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>

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

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, $\text{LimbLength}(j)$.
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 $\text{Tree}(D)$. 
AdditivePhylogeny

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

#### Table

<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>

#### Graph

![Tree Diagram](attachment:image.png)

#### Limb Length Theorem

The length of the limb of $j$ is equal to the minimum value of $(D_{i,j}^{\text{bald}} + D_{j,k}^{\text{bald}} - D_{i,k}^{\text{bald}})/2$ over all leaves $i$ and $k$. 

---

**Note:** The graph shows a tree structure with nodes $i$, $j$, $k$, and $l$, and edge weights as indicated.
Attaching a Limb

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

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

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

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

\[
\left( D_{\text{bald}}^\text{i,j} + D_{\text{bald}}^\text{j,k} - D_{\text{bald}}^\text{i,k} \right)/2 = 0
\]

\[
\text{TREE}(D_{\text{trim}})
\]
### 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 \(\text{Tree}(D)\).

Sum of Squared Errors

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

\[ = 1^2 + 1^2 = 2 \]
**Exercise Break:** Assign lengths to edges in $T$ in order to minimize $\text{Discrepancy}(T, D)$.

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

$$
\begin{array}{cccc}
  i & j & k & l \\
  i & 0 & ? & ? & ? \\
  j & ? & 0 & ? & ? \\
  k & ? & ? & 0 & ? \\
  l & ? & ? & ? & 0 \\
\end{array}
$$
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.

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

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

- Squirrel
- Baboon
- Orangutan
- Gorilla
- Chimpanzee
- Bonobo
- Human

- 33
- 23
- 10
- 10
- 6
- 1
- 2
- 6
- 2
- 2
- 6

Example:

- Baboon:
  - Distance to root: 23
  - Edge weights: 10

- Orangutan:
  - Distance to root: 23
  - Edge weights: 10

- Gorilla:
  - Distance to root: 10
  - Edge weights: 23

- Chimpanzee:
  - Distance to root: 10
  - Edge weights: 23

- Bonobo:
  - Distance to root: 13
  - Edge weights: 23

- Human:
  - Distance to root: 6
  - Edge weights: 1
Ultrametric tree: distance from root to any leaf is the same (i.e., age of root).
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

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

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

<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: A Clustering Heuristic

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

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

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

UPGMA: A Clustering Heuristic
UPGMA: A Clustering Heuristic

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

\[
\begin{array}{ccc}
  i & j & \{k, l\} \\
  i & 0 & 3 & 3.5 \\
  j & 3 & 0 & 4.5 \\
  \{k, l\} & 3.5 & 4.5 & 0 \\
\end{array}
\]

![Clustering diagram]
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.

<table>
<thead>
<tr>
<th></th>
<th>{i, j}</th>
<th>{k, l}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{i, j}</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>{k, l}</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

![Clustering Diagram](image.png)
UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.

<table>
<thead>
<tr>
<th>{i, j}</th>
<th>{k, l}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

![Tree diagram showing distances between clusters and species.](image)
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
   
   \[
   D_{\text{avg}}(C_1, C_2) = \frac{\sum_{i \in C_1, j \in C_2} D_{i,j}}{|C_1| \cdot |C_2|}
   \]

   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

\[
\begin{array}{cccc}
  i & j & k & l \\
  i & 0 & 3 & 4 & 3 \\
  j & 3 & 0 & 4 & 5 \\
  k & 4 & 4 & 0 & 2 \\
  l & 3 & 5 & 2 & 0 \\
\end{array}
\]
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>
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
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.
Neighbor-Joining Theorem

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

<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>

<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>56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$j$</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k$</td>
<td>46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$l$</td>
<td>48</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<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>-68</td>
<td>-60</td>
<td>-60</td>
</tr>
<tr>
<td>$j$</td>
<td>-68</td>
<td>0</td>
<td>-60</td>
<td>-60</td>
</tr>
<tr>
<td>$k$</td>
<td>-60</td>
<td>-60</td>
<td>0</td>
<td>-68</td>
</tr>
<tr>
<td>$l$</td>
<td>-60</td>
<td>-60</td>
<td>-68</td>
<td>0</td>
</tr>
</tbody>
</table>
## 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>Total Distance$_D$</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>-60</td>
<td>0</td>
<td>48</td>
</tr>
</tbody>
</table>
2. Find a minimum element $D^*_{i,j}$ of $D^*$. 

\begin{align*}
\begin{array}{cccc}
 i & j & k & l \\
 i & 0 & -68 & -60 & -60 & 56 \\
 j & -68 & 0 & -60 & -60 & 38 \\
 k & -60 & -60 & 0 & -68 & 46 \\
 l & -60 & -60 & -68 & 0 & 48 \\
\end{array}
\end{align*}
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

#### 3. Compute $\Delta_{i,j} = \frac{\text{TotalDistance}_D(i) - \text{TotalDistance}_D(j)}{(n - 2)}$.

<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>

$\Delta_{i,j} = \frac{56 - 38}{4 - 2} = 9$
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 LimbLength<sub>i</sub> equal to \( \frac{1}{2}(D_{i,j} + \Delta_{i,j}) \) and LimbLength<sub>j</sub> equal to \( \frac{1}{2}(D_{i,j} - \Delta_{j,i}) \).
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 $k$, $D_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$. 

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

\[
\begin{array}{c}
\text{TotalDistance}_D \\
21 \\
23 \\
24 \\
\end{array}
\]
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 $Tree(D')$. 
Neighbor-Joining in Action

<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>

$LimbLength(i) = \frac{1}{2}(13 + 9) = 11$

$LimbLength(i) = \frac{1}{2}(13 - 9) = 2$

7. Reattach limbs of $i$ and $j$ to obtain $Tree(D)$. 
Neighbor-Joining in Action

$D'$

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

7. Reattach limbs of $i$ and $j$ to obtain $Tree(D)$.
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 \(\frac{1}{2}(D_{i,j} - \Delta_{j,i})\).
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 \(\text{Tree}(D)\).

**Code Challenge:** Implement **NeighborJoining**.
Neighbor-Joining

**Exercise Break, check the following:** Neighbor joining on a set of $r$ taxa 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 $O(r^3)$. 
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
Example (different notation)

Distance matrix

<table>
<thead>
<tr>
<th></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>

Step 1

$S$ calculations

$S_A = \frac{(5+4+7+6+8)}{4} = 7.5$

$S_B = \frac{(5+10+4+11)}{4} = 10.5$

$S_C = \frac{(4+7+6+8)}{4} = 8$

$S_D = \frac{(5+7+5+9)}{4} = 9$

$S_E = \frac{(6+9+6+5+8)}{4} = 8.5$

$S_F = \frac{(8+11+8+9+8)}{4} = 11$

Step 2

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

Smallest are

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

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

Choose one of these (AB here).

Step 3

Create a node $(U_1)$ that joins pair with lowest $M_i$ such that $S_{U1} = D_{ij}/2 + (S_i - S_j)/2$.

$U_1$ joins A and B:

$S_{U1A} = D_{AB}/2 + (S_A - S_B)/2 = 1$

$S_{U1B} = D_{AB}/2 + (S_B - S_A)/2 = 4$

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 $D_{ij} = D_{ij} + D_{ip} - D_{pj}$, where $i$ and $j$ are those selected from above.

Example tree with branch length 5.

Notes:

- $U_1$, $U_2$, $U_3$, $U_4$ are nodes.

- $D_{ij}$ is the distance between taxa $i$ and $j$.

- $S_{ij}$ is the sum of distances from taxa $i$ to all other taxa.

- $M_{ij}$ is the minimum distance from taxa $i$ to any other taxa.

- The tree is drawn in unrooted form here.
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>TCGAAAGGCAT</td>
<td>4 5 2 0</td>
</tr>
</tbody>
</table>
An Alignment As a Character Table

<table>
<thead>
<tr>
<th>SPECIES</th>
<th>ALIGNMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>ACGTAGGCCT</td>
</tr>
<tr>
<td>Human</td>
<td>ATGTAAGACT</td>
</tr>
<tr>
<td>Seal</td>
<td>TCGAGAGCAC</td>
</tr>
<tr>
<td>Whale</td>
<td>TCGAAAGCAT</td>
</tr>
</tbody>
</table>

\[ n \text{ species} \]

\[ m \text{ characters} \]
Toward a Computational Problem

Species: Chimp, Human, Seal, Whale

Characters: ACGTAGGCCT, ATGTAAGACT, TCGAGAGCAC, TCGAAAGCAT

$n$ species, $m$ characters
Toward a Computational Problem

Chimp
Human
Seal
Whale

ACGTAGGCCT
ATGTAAGACT
TCGAGAGCAC
TCGAAAGCAT
Toward a Computational Problem
Toward a Computational Problem

**Parsimony score:** sum of Hamming distances along each edge.
Toward a Computational Problem

Parsimony score: sum of Hamming distances along each edge.

Parsimony Score: 8

Chimp
Human
Seal
Whale
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 minimizes the tree’s parsimony score.
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 minimizes 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.
Toward a Computational Problem
A Dynamic Programming Algorithm

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

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

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

For symbols $i$ and $j$, define

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

**Exercise Break:** Prove the following recurrence relation:

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

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

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

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

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

Exercise Break: “Backtrack” to fill in the remaining nodes of the tree.
A Dynamic Programming Algorithm

Code Challenge: Solve the Small Parsimony Problem.
Exercise Break, check the following: 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 $O(mnk^2)$. 

David Sankoff
Exercise Break, check the following: 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 $O(mnk^2)$.

COMMENT: if each mutation costs the same then a simplified, earlier version of this algorithm from Walter Fitch gives a run time complexity of $O(mnk)$. If Each mutation $a \leftrightarrow b$ costs differently you have a weighted edit distance (particularly for amino acid sequences) then your complexity is likely to be $O(mnk^2)$.
Simple example

\[
R_i = \begin{cases} 
R_j \cap R_k & \text{if } R_j \cap R_k \neq \emptyset \\
R_j \cup R_k & \text{otherwise}
\end{cases}
\]

simple case Sankoff equivalent to computing this using this scoring matrix
Bottom-UP phase

\[
R_i = \begin{cases} 
R_j \cap R_k \text{ if } R_j \cap R_k \neq \emptyset \\
R_j \cup R_k \text{ otherwise}
\end{cases}
\]

Top-down phase

\[
s_j = \begin{cases} 
s_i \text{ if } s_i \in R_j \\
a r b i t a r y \ state \in R_j \text{ otherwise}
\end{cases}
\]

Complexity: \(O(mnk)\)

score = 3
How to compare amino acids: scoring matrices

<table>
<thead>
<tr>
<th>C</th>
<th>Cys</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Ser</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>Thr</td>
<td>-2</td>
</tr>
<tr>
<td>P</td>
<td>Pro</td>
<td>-3</td>
</tr>
<tr>
<td>A</td>
<td>Ala</td>
<td>-2</td>
</tr>
<tr>
<td>G</td>
<td>Gly</td>
<td>-3</td>
</tr>
<tr>
<td>N</td>
<td>Asn</td>
<td>-4</td>
</tr>
<tr>
<td>D</td>
<td>Asp</td>
<td>-5</td>
</tr>
<tr>
<td>E</td>
<td>Glu</td>
<td>-5</td>
</tr>
<tr>
<td>Q</td>
<td>Gln</td>
<td>-5</td>
</tr>
<tr>
<td>H</td>
<td>His</td>
<td>-3</td>
</tr>
<tr>
<td>R</td>
<td>Arg</td>
<td>-4</td>
</tr>
<tr>
<td>K</td>
<td>Lys</td>
<td>-5</td>
</tr>
<tr>
<td>M</td>
<td>Met</td>
<td>-5</td>
</tr>
<tr>
<td>I</td>
<td>Ile</td>
<td>-2</td>
</tr>
<tr>
<td>L</td>
<td>Leu</td>
<td>-6</td>
</tr>
<tr>
<td>V</td>
<td>Val</td>
<td>-2</td>
</tr>
<tr>
<td>F</td>
<td>Phe</td>
<td>-4</td>
</tr>
<tr>
<td>Y</td>
<td>Tyr</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>Trp</td>
<td>-8</td>
</tr>
</tbody>
</table>

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

Top-down phase

Pick states for each internal node

- Select minimal cost character for root ($s$ minimizing $R_{\text{root}}(s)$)
- Do pre-order (from root to leaves) traversal of tree:
  - For internal node $j$, with parent $i$, select state that produced minimal cost at $i$ (use pointers kept in 1st stage)

Complexity: $O(mnk^2)$
Measuring SP and MP complexity in terms of basic operations. SP and MP algorithms work by computing some information for every internal vertex of the input phylogeny. This information, as well as the complexity of its computation, depend on the scoring scheme employed by the parsimony algorithm. Thus, in what follows, we will use the term basic operation to denote the work invested in the computation of the information of a single vertex of a considered phylogeny for a specific scoring scheme. For example, in the Fitch SP algorithm (Fitch, 1971), which computes a minimal Hamming distance SP score, an $O(m)$-time basic operation is applied, while in the Sankoff algorithm (Sankoff, 1975), which optimizes an SP score of minimal weighted edit distance, an $O(m\Sigma^2)$-time basic operation is applied, where $\Sigma$ denotes the size of the alphabet spelling the input sequences.
Why is interesting to know internal node’s composition?
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 minimizes the tree’s parsimony score.

**Code Challenge:** Solve this problem.
Finding the Most Parsimonious Tree

Parsimoney Score: 8
Finding the Most Parsimonious Tree

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 minimizes 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 minimizes 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.
A Greedy Heuristic for Large Parsimony

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

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

Code Challenge: Solve this problem.
A Greedy Heuristic for Large Parsimony

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

Code Challenge: Implement the nearest-neighbor interchange heuristic.
Tree validation: the bootstrap algorithm

- If there are m sequences, each with n nucleotides, a phylogenetic tree can be reconstructed using some tree building methods.
- 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.
- A tree is then reconstructed with these new sequences using the same tree building method as before.
- Next 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.

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. 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

(b) Subhypothesis 1

Bootstrap value

95% † is significantly positive
**EXAMPLE**: Phylogenetic-inspired techniques for reverse engineering and detection of malware families

For example, given an execution trace of instructions,

```c
push ebp
mov ebp, esp
mov eax, dword ptr [ebp-0x4]
jmp +0x14
```

it is abstracted as a sequence of mnemonics, i.e.

```c
push, mov, mov, jmp
```

ignoring the operands. Each mnemonic is then mapped to a unique alphabet-pair, e.g. \texttt{mov} = \texttt{MO}, \texttt{push} = \texttt{PH}, \texttt{jmp} = \texttt{JM}. The resulting sequence is thus \texttt{PHMOMOJM}.

Sequence alignment (\texttt{dbg}: with debugging symbols, \texttt{def}: default settings, \texttt{spd}: optimised for speed). (a) Before alignment. (b) After alignment using an identity substitution matrix. (c) After alignment using a substitution matrix.
Distance algorithm in computer science
A) A sequence logo for the FakeAV-DO function “ F1 ”. Positions with large characters indicate invariant parts of the function; positions with small characters vary due to code metamorphism
B) A neighbour joining tree of FakeAV-DO set of procedures F1.
C) Neighbor joining tree of FakeAV-DO set of procedures F2 from the same samples of B.

(W.M. Khoo and P. Lio' Unity in diversity: Phylogenetic-inspired techniques for reverse engineering and detection of malware families. 2011 First SysSec Workshop)
More species increases power to detect conserved sequence elements: the phylogeny becomes a weight

Data from Eric Green at NGHRI, alignments by Webb Miller
• Alignment of 2 sequences is a 2-row matrix.
• Alignment of 3 sequences is a 3-row matrix

\[
\begin{align*}
\text{AT} & - \quad \text{GCG} - \\
\text{A} & - \quad \text{CGT} - \quad \text{A} \\
\text{ATC} & \quad \text{ACC} - \quad \text{A}
\end{align*}
\]

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

• Alignment of ATGC, AATC, and ATGC

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

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

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

#symbols up to a given position
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

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

- \( \delta(x, y, z) \) is an entry in the 3-D scoring matrix.
Multiple Alignment: Running Time

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

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

Every multiple alignment induces pairwise alignments:

```
AC - GCGG - C
AC - GC - GA G
GCCGC - GA G
```

\[ \downarrow \]

```
ACGCGG - C  AC - GCGG - C  AC - GC CGAG
ACGC - GAC  GCCGC - GA G  GCCGCGAG
```
Idea: Construct Multiple from Pairwise Alignments

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

\[
\begin{align*}
\text{AAAATTTT} & \quad \text{----} & \quad \text{----} & \quad \text{AAAATTTT} & \quad \text{TTTTGGGG} & \quad \text{----} \\
\text{----TTTTGGGG} & \quad \text{GGGGAAAA} & \quad \text{----} & \quad \text{GGGGAAAA} & \quad \text{----} & \quad \text{GGGGAAAA}
\end{align*}
\]
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

Not all the pairwise alignments build well into a multiple sequence alignment (compare the alignments on the left and right)
The progressive alignment builds a final alignment by merging sub-alignments (bottom to top) with a guide 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.
Main question in this lecture: how similar is this group of sequences?

Example of complexity in alignment: bacterial genomes

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

- 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
Why Do We Sequence Personal Genomes?

• **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.
The Newspaper Problem

stack of NY Times, June 27, 2000

stack of NY Times, June 27, 2000
on a pile of dynamite

this is just hypothetical

BOOM

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

Breaking the Genomes at Random Positions
Generating “Reads”

CTGATGA TGGACTACGCTAC TACTGCTAG CTGTATTACG ATCAGCTACCACA TCGTAGCTACG ATGCATTAGCAA GCTATCGGA TCAGCTACCA CATCGTAGC
CTGATGATG GACTACGCTAC TACTGCTAGCT ACATCGTAGCT ATCAGCTACC ACATCGTAGCT ACATCGTAGCT ATCAGCTACC ACATCGTAGCT
CTGATGATG ACTACGCTAC TACTGCTAGCT ACATCGTAGCT ACATCGTAGCT ACATCGTAGCT ACATCGTAGCT ACATCGTAGCT
CTGATGATGACT ACGCTACTAC TACTGCTAGCT ACATCGTAGCT ACATCGTAGCT ACATCGTAGCT ACATCGTAGCT ACATCGTAGCT

“Burning” Some Reads
No Idea What Position Every Read Comes From
Multiple (unsequenced) genome copies

Read generation

Reads

Genome assembly

Assembled genome

...GGCATGCGTCAGAAACTATCATAGCTAGATCGTACGTAGCC...
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.
What Is k-mer Composition?

\[ \text{Composition}_3(\text{TAA-TGCCATGGGATGT}) = \]

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

\[ \text{Composition}_3(TAATGCCATGGGATGT) = \]
\[ \text{TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT} = \]
\[ \text{AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT} \]

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
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-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

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.
A Slightly Different Path

TAATGCCATGGGATGTT

3-mers as nodes

3-mers as edges

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

prefix of TAA TAA

suffix of TAA AA
Labeling Nodes in the New Path

TA\textsc{ATGCCATGGGATGTT}

3-mers as nodes

TA\textsc{AATATGTCGCACCATGTT}

3-mers as edges and 2-mers as nodes
Labeling Nodes in the New Path

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

TAATGCCATGGGATGT
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.
Eulerian Path Problem

Eulerian Path Problem. Find an Eulerian path in a graph.

• Input. A graph.

• Output. A path visiting every edge in the graph exactly once.
Eulerian Versus Hamiltonian Paths

Eulerian Path Problem. Find an Eulerian path in a graph.

- Input. A graph.
- Output. A path visiting every edge in the graph exactly once.

Hamiltonian Path Problem. Find a Hamiltonian path in a graph.

- Input. A graph.
- Output. A path visiting every node in the graph exactly once.

Find a difference!
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

TAATGCCATGGGATGTT

332
What We Want: From Reads (k-mers) to Genome

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

TAATGCCATGGATGTT

TA → AA → AT → TG → GC → CC → CA → AT → TG → GG → GG → GA → AT → TG → GT → TT
Constructing de Bruijn when Genome Is Unknown

<table>
<thead>
<tr>
<th>TAA</th>
<th>ATG</th>
<th>GCC</th>
<th>CAT</th>
<th>TGG</th>
<th>GGA</th>
<th>ATG</th>
<th>GTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAT</td>
<td>TGC</td>
<td>CCA</td>
<td>ATG</td>
<td>GGG</td>
<td>GAT</td>
<td>TGT</td>
<td></td>
</tr>
</tbody>
</table>

Composition\textsubscript{3}(TAATGCCATGGGATGT)

336
Representing Composition as a Graph Consisting of Isolated Edges

Composition_3(TAACATGGATGGT)
Constructing de Bruijn Graph from k-mer Composition

Composition$_3$(TAATGCCATGGGATGT)
Gluing Identically Labeled Nodes
We Are Not Done with Gluing Yet
Gluing Identically Labeled Nodes
Gluing Identically Labeled Nodes
Gluing Identically Labeled Nodes
The Same de Bruijn Graph:
\[ \text{DeBruiin(Genome)} = \text{DeBruiin(Genome Composition)} \]
Constructing de Bruijn Graph

De Bruijn graph of a collection of $k$-mers:

- Represent every $k$-mer as an edge between its prefix and suffix.
- Glue ALL nodes with identical labels.
Universal String Problem (Nicolaas de Bruijn, 1946). Find a circular string containing each binary k-mer exactly once.

000 001 010 011 100 101 110 111
From Hamilton to Euler to de Bruijn

Universal String Problem (Nicolaas de Bruijn, 1946). Find a circular string containing each binary k-mer exactly once.

000 001 010 011 100 101 110 111

Diagram showing connections between k-mers.
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 \textit{indegree} = \textit{outdegree} for each node
Euler’s Theorem

- Every Eulerian graph is balanced
- Every balanced* graph is Eulerian

(*) and strongly connected, of course!
Recruiting an Ant to Prove Euler’s Theorem

Let an ant randomly walk through the graph. The ant cannot use the same edge twice!
If Ant Was a Genius...

“Yay! Now can I go home please?”
A Less Intelligent Ant Would Randomly Choose a Node and Start Walking...

Can it get stuck? **In what node?**
The Ant Has Completed a Cycle BUT has not Proven Euler’s theorem yet...

The constructed cycle is not Eulerian. *Can we enlarge it?*
Let’s Start at a Different Node in the Green Cycle

Let’s start at a node with still unexplored edges.

“Why should I start at a different node? Backtracking? I’m not evolved to walk backwards! And what difference does it make???”
An Ant Traversing Previously Constructed Cycle

Starting at a node that has an unused edge, traverse the already constructed (green cycle) and return back to the starting node.

“Why do I have to walk along the same cycle again?? Can I see something new?”
I Returned Back BUT... I Can Continue Walking!

Starting at a node that has an unused edge, traverse the already constructed (green cycle) and return back to the starting node.

After completing the cycle, start random exploration of still untraversed edges in the graph.
Stuck Again!

No Eulerian cycle yet... can we enlarge the green-blue cycle?

The ant should walk along the constructed cycle starting at yet another node. Which one?
I Returned Back BUT... I Can Continue Walking!

“Hmm, maybe these instructions were not that stupid...”
I Proved Euler’s Theorem!

**EulerianCycle**(*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
Multiple Eulerian Paths

TA TGCCATGGGATGTT

A

TA TGGGATGCCATGTT

A
Breaking Genome into Contigs

TA TGCCATGGGATGTT
A

[Diagram of genome breaking into contigs with base pairs and annotations]
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: $\rightarrow \ldots \leftarrow$ rather than $\rightarrow \ldots \rightarrow$

2. In reality, the distance $d$ between reads is measured with errors.
What is PairedComposition(TA\textsc{ATGC}CAT\textsc{GGATGTT})?

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

\begin{verbatim}
 TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA
 GCC CCA CAT ATG TGG GGG GGA
\end{verbatim}
PairedComposition(TAATGCATGGATTT)

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

How Would de Bruijn Assemble Paired \( k \)-mers?
Representing Genome TATA\textbf{G}CCATGGG\textbf{A}TGTT as a Path

paired prefix of \textbf{CCA} GGG \quad \textbf{←} \quad \textbf{paired suffix of} \quad \textbf{CCA} GGG
Labeling Nodes by Paired Prefixes and Suffixes

paired prefix of

paired suffix of
Glue nodes with identical labels
Glue nodes with identical labels

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

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

```
>679__atgccgtatgggacaacgact
>680_atgccgtatg
gccgtatgga
gtatggacaa
gacaacgact
```

```
>681__atgccgtatgggacaacgact
>682_atgcc
>683_tgccg
gccgt
cgta
cgtat
gtatg	atatg	atatg	atatg	atatg
ggaca
gacaa
acaac
caacg
aacga
acgac
cgact
```
2\textsuperscript{nd} 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.
De Bruin Graph of *N. meningitidis* Genome AFTER Removing Bubbles

Red edges represent repeats
Example and RECAP

Input: GGC\textit{GCTATATCTCGGCTCTAGGCCCTCTATTTTTT}
Copy: GGC\textit{GCTATATCTCGGCTCTAGGCCCTCTATTTTTT}
GC\textit{GCTATATCTCGGCTCTAGGCCCTCTATTTTTT}
GC\textit{GCTATATCTCGGCTCTAGGCCCTCTATTTTTT}
GC\textit{GCTATATCTCGGCTCTAGGCCCTCTATTTTTT}

Fragment: GGC\textit{GCCTA TATCTCGG CTCTAGGCCCTC ATTTTTTT}
GGC\textit{GTCTATATCTCGGCTCTAGGCCCTCA TTTTTTT}
GGCG\textit{TC ATATCT CGGCTCTAGGCCCT CATT TT TT}
GGCG\textit{TC ATATCT ATCTCGGCTCTAG GCCCTCA TTTTTT}

\text{CTAGGCCCTCAATTTTTT}
\text{CTCTAGGCCCTCAATTTTTT}
\text{GGCTCTAGGCCCTCATTTTTT}
\text{CTCGGCTCTAGGCCCTCATTTTTT}
\text{TATCTCGACTCTAGGCCCTCA}
\text{TATCTCGACTCTAGGCC}
\text{TCTATATCTCGGCTCTAGG}
GGCG\textit{TCATATCTCGGCTCTAGG}
GGCG\textit{TCATATCTCGGCTCTAGG}
GGCG\textit{TCATATCTCGGCTCTAGG}
GGCG\textit{TCATATCTCGGCTCTAGG}

177 nucleotides
35 nucleotides

Average coverage = \frac{177}{35} = 7x
“k-mer” is a substring of length $k$

S: GGCGATTCATCG
A 4-mer of S: ATTC
All 3-mers of S:
- GGC
- GCG
- CGA
- GAT
- ATT
- TTC
- TCA
- CAT
- ATC
- TCG

I’ll use “$k$-mer” to refer to a substring of length $k - 1$

AAA, AAB, ABB, BBB, BBA

AAB is a $k$-mer ($k = 3$). AA is its left $k$-1-mer, and AB is its right $k$-1-mer.

AAB 3-mer

AA  AB
L  R

AAB’s left 2-mer  AAB’s right 2-mer
Example and RECAP

**Short-read sequencing**

![Diagram of genome and k-mers](image)

- **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)
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 $000011001011110101001000$. 

Example and RECAP
Example and RECAP

AAABBBBA

take all 3-mers: AAA, AAB, ABB, BBB, BBA

form L/R 2-mers: AA, AA, AA, AB, AB, BB, BB, BB, BB, BB, BA

Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

Each edge in this graph corresponds to a length-3 input string
An edge corresponds to an overlap (of length $k-2$) between two $k$-mers. More precisely, it corresponds to a \textit{k-mer} from the input.

If we add one more B to our input string: $\text{AAAABBBBA}$, and rebuild the De Bruijn graph accordingly, we get a \textit{multiedge}.
Node is *balanced* if indegree equals outdegree

Node is *semi-balanced* if indegree differs from outdegree by 1

Graph is *connected* if each node can be reached by some other node

*Eulerian walk* visits each edge exactly once

Not all graphs have Eulerian walks. Graphs that do are *Eulerian.* (For simplicity, we won’t distinguish Eulerian from semi-Eulerian.)

Example and RECAP

Is it Eulerian? Yes

Argument 1: AA → AA → AB → BB → BB → BA

Argument 2: AA and BA are semi-balanced, AB and BB are balanced
De Bruijn graph

A procedure for making a De Bruijn graph for a genome

Assume *perfect sequencing* where each length-$k$ substring is sequenced exactly once with no errors

Pick a substring length $k$: 5

Start with each read: `a_long_long_long_time`

Take each $k$ mer and split into left and right $k-1$ mers

Add $k-1$ mers as nodes to De Bruijn graph (if not already there), add edge from left $k-1$ mer to right $k-1$ mer
First 8 k-mer additions, $k = 5$

a_long_long_long_time
Example and RECAP

Last 5 k-mer additions, $k = 5$

\texttt{a_long_long_long_time}

Finished graph
De Bruijn graph

With perfect sequencing, this procedure always yields an Eulerian graph. Why?

Node for $k$-1-mer from left end is semi-balanced with one more outgoing edge than incoming *

Node for $k$-1-mer at right end is semi-balanced with one more incoming than outgoing *

Other nodes are balanced since # times $k$-1-mer occurs as a left $k$-1-mer = # times it occurs as a right $k$-1-mer

* Unless genome is circular
De Bruijn graph

Assuming perfect sequencing, procedure yields graph with Eulerian walk that can be found efficiently.

We saw cases where Eulerian walk corresponds to the original superstring. Is this always the case?
Example and RECAP

How much work to build graph?

For each k-mer, add 1 edge and up to 2 nodes

Reasonable to say this is O(1) expected work

Assume hash map encodes nodes & edges

Assume k-1-mers fit in O(1) machine words, and hashing O(1) machine words is O(1) work

Querying / adding a key is O(1) expected work

O(1) expected work for 1 k-mer, O(N) overall
Example and RECAP

In typical assembly projects, average coverage is ~ 30 - 50

Same edge might appear in dozens of copies; let’s use edge weights instead

Weight = # times k-mer occurs

Using weights, there’s one weighted edge for each distinct k-mer

Before: one edge per k-mer  After: one weighted edge per distinct k-mer

http://nbviewer.jupyter.org/github/BenLangmead/comp-genomics-class/blob/master/notebooks/CG_deBruijn.ipynb
Example and RECAP

- Errors at end of read
  - Trim off ‘dead-end’ tips

- Errors in middle of read
  - Pop Bubbles

- Chimeric Edges
  - Clip short, low coverage nodes
Example and RECAP

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


itwashebestoftimesitwastheworstofofimesitwasheageofwisdomitwasheageoffoolishness...

How do we assemble?

...etc. to 10’s of millions of reads
Step 1:
Convert reads into “Kmers”

Kmer: a substring of defined length

Reads: theageofwisthebestofasteageofworstoftimimesitwast

Kmers: theheasthworime

(k=3) thesthsaststhosmesheastheavrhrs

Step 2:
Build a De-Bruijn graph from the kmers

ast → sth → the → hea → eag → age → geo → eof
ast → sth → the → hea → eag → age → geo → eof → ofw → fwi

heb → ebe → bes → est → sto → tof

wor → ors → rst

oft → fti → tim

ime → mes

esi

twa

itw

sit
Step 3:
Simplify the graph as much as possible:

A De Bruijn Graph

De Bruijn assembles ‘broken’ by repeats longer than kmer

"It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, .... “
The final assembly ($k=3$)

wor times itwas the foolishness st wisdom
incredulity age epoch be of belief

Repeat with a longer “kmer” length

A better assembly ($k=20$)

itwas the best of times it was the worst of times it was the age of wisdom it was the age of foolishness...

Why not always use longest ‘k’ possible?

Sequencing errors:

sthebentof

$\text{k=3}$

sth the heb ebe ben nto tof Mostly unaffected kmers

$\text{k=10}$

sthebentof 100% wrong kmer

Slides from Presentation by Alicia Clum genomebiology.gji-psf.org/Content/MGM-13.Sep2012/.../3.clum.ppt
Clustering Algorithms

- Clustering as an optimization problem
- The Lloyd algorithm for $k$-means clustering
- From Hard to Soft Clustering
- From Coin Flipping to $k$-means Clustering
- Expectation Maximization
- Soft $k$-means Clustering
- Hierarchical Clustering
- Markov Clustering Algorithm
- Stochastic Neighbor Embedding
Measuring 3 Genes at 7 Checkpoints

Measure expression of various yeast genes at 7 checkpoints:

<table>
<thead>
<tr>
<th>Gene</th>
<th>-6h</th>
<th>-4h</th>
<th>-2h</th>
<th>0</th>
<th>+2h</th>
<th>+4h</th>
<th>+6h</th>
</tr>
</thead>
<tbody>
<tr>
<td>YLR258W</td>
<td>1.1</td>
<td>1.4</td>
<td>1.4</td>
<td>3.7</td>
<td>4.0</td>
<td>10.0</td>
<td>5.9</td>
</tr>
<tr>
<td>YPL012W</td>
<td>1.1</td>
<td>0.8</td>
<td>0.9</td>
<td>0.4</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>YPR055W</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
</tbody>
</table>

\[ e_{ij} = \text{expression level of gene } i \text{ at checkpoint } j \]
GSY2 | SGD
https://www.yeastgenome.org/locus/S000004248
30 ago 2005 - Standard Name: GSY2; Systematic Name: YLR258W; SGD ID: SGD... of yeast glycogen synthase-2 by COOH-terminal phosphorylation.

YLR258W | SGD-Wiki
23 gen 2012 - Description of YLR258W: Glycogen synthase, similar to Gsy/p expression... of yeast glycogen synthase-2 by COOH-terminal phosphorylation.

GSY2 Protein | SGD
https://www.yeastgenome.org/locus/S000004248/protein
... Database (SGD) provides comprehensive integrated biological information for the budding yeast Saccharomyces cerevisiae... GSY2 / YLR258W Protein.

GSY2 - Glycogen [starch] synthase isoform 2 - Saccharomyces cerevisiae
https://www.uniprot.org/uniprot/P27472
... Status: BioCyc, YEAST:YLR258W-MONOMER;... Ordered Locus Names: YLR258W.
Switching to Logarithms of Expression Levels

<table>
<thead>
<tr>
<th></th>
<th>YLR258W</th>
<th>1.1</th>
<th>1.4</th>
<th>1.4</th>
<th>3.7</th>
<th>4.0</th>
<th>10.0</th>
<th>5.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>YPL012W</td>
<td>1.1</td>
<td>0.8</td>
<td>0.9</td>
<td>0.4</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>YPR055W</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td></td>
</tr>
</tbody>
</table>

Taking logarithms (base-2)

<table>
<thead>
<tr>
<th></th>
<th>YLR258W</th>
<th>0.1</th>
<th>0.4</th>
<th>0.5</th>
<th>1.9</th>
<th>2.0</th>
<th>3.3</th>
<th>2.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>YPL012W</td>
<td>0.1</td>
<td>-0.3</td>
<td>-0.2</td>
<td>-1.2</td>
<td>-1.6</td>
<td>-3.0</td>
<td>-3.1</td>
<td></td>
</tr>
<tr>
<td>YPR055W</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td></td>
</tr>
</tbody>
</table>
### Gene Expression Matrix

<table>
<thead>
<tr>
<th>Gene</th>
<th>YLR361C</th>
<th>YMR290C</th>
<th>YNR065C</th>
<th>YGR043C</th>
<th>YLR258W</th>
<th>YPL012W</th>
<th>YNL141W</th>
<th>YJL028W</th>
<th>YKL026C</th>
<th>YPR055W</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.14</td>
<td>0.12</td>
<td>-0.10</td>
<td>-0.43</td>
<td>0.11</td>
<td>0.09</td>
<td>-0.16</td>
<td>-0.28</td>
<td>-0.19</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
<td>-0.23</td>
<td>-0.14</td>
<td>-0.73</td>
<td>0.43</td>
<td>-0.28</td>
<td>-0.04</td>
<td>-0.28</td>
<td>-0.23</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>-0.06</td>
<td>-0.24</td>
<td>-0.03</td>
<td>-0.06</td>
<td>0.45</td>
<td>-0.15</td>
<td>-0.07</td>
<td>-0.15</td>
<td>-0.19</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>0.07</td>
<td>-1.16</td>
<td>-0.06</td>
<td>-0.11</td>
<td>1.89</td>
<td>-1.18</td>
<td>-0.07</td>
<td>-0.19</td>
<td>0.03</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>-0.01</td>
<td>-1.40</td>
<td>-0.06</td>
<td>-0.16</td>
<td>2.64</td>
<td>-1.59</td>
<td>-0.26</td>
<td>-0.19</td>
<td>0.27</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>-0.06</td>
<td>-2.67</td>
<td>-0.14</td>
<td>3.47</td>
<td>2.00</td>
<td>-2.96</td>
<td>-1.20</td>
<td>-0.32</td>
<td>0.54</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>-0.01</td>
<td>-3.00</td>
<td>-0.04</td>
<td>-1.40</td>
<td>3.32</td>
<td>-3.08</td>
<td>-2.82</td>
<td>-0.18</td>
<td>2.74</td>
<td>0.07</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>
</tr>
</tbody>
</table>

**Gene Expression Vector**

![Gene Expression Vector Diagram](image)

**Figure 1.4** The rows of the matrix from Figure 1.3 partitioned into three clusters.

- **Green genes** exhibit increased expression,
- **red genes** exhibit decreased expression,
- **blue genes** exhibit flat behavior and are unlikely to be associated with the diauxic shift.

The element with the largest absolute value in each expression vector is shown in bold, and the mean $\mu$ and variance $s^2$ of each expression vector is shown in the rightmost two columns. (Bottom) The rows of the matrix visualized as plots.

**The Good Clustering Principle**

To identify groups of genes with similar expression, we will think of an expression vector of length $m$ as a point in $m$-dimensional space; genes with similar expression profiles will therefore 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.
Gene Expression Matrix

<table>
<thead>
<tr>
<th>Gene</th>
<th>YLR361C</th>
<th>YMR290C</th>
<th>YNR065C</th>
<th>YGR043C</th>
<th>YLR258W</th>
<th>YPL012W</th>
<th>YNL141W</th>
<th>YJL028W</th>
<th>YKL026C</th>
<th>YPR055W</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.14</td>
<td>0.12</td>
<td>-0.10</td>
<td>-0.43</td>
<td>0.11</td>
<td>0.09</td>
<td>-0.16</td>
<td>-0.28</td>
<td>-0.19</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
<td>-0.23</td>
<td>-0.14</td>
<td>-0.73</td>
<td>0.43</td>
<td>-0.28</td>
<td>-0.04</td>
<td>-0.23</td>
<td>-0.15</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>-0.06</td>
<td>-0.24</td>
<td>-0.03</td>
<td>-0.06</td>
<td>0.45</td>
<td>-0.15</td>
<td>-0.07</td>
<td>-0.19</td>
<td>0.03</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>0.07</td>
<td>-1.16</td>
<td>-0.06</td>
<td>-0.11</td>
<td>1.89</td>
<td>-1.18</td>
<td>-0.26</td>
<td>-0.19</td>
<td>0.27</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>-0.01</td>
<td>-1.40</td>
<td>-0.07</td>
<td>-0.16</td>
<td>2.00</td>
<td>-1.59</td>
<td>-1.20</td>
<td>-0.32</td>
<td>0.54</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>-0.06</td>
<td>-2.67</td>
<td>-0.14</td>
<td>3.47</td>
<td>3.32</td>
<td>-2.96</td>
<td>-2.82</td>
<td>-0.18</td>
<td>3.64</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>-0.01</td>
<td>-3.00</td>
<td>-0.04</td>
<td>2.64</td>
<td>2.56</td>
<td>-3.08</td>
<td>-3.13</td>
<td>-0.18</td>
<td>2.74</td>
<td>0.07</td>
</tr>
</tbody>
</table>

1997: Joseph deRisi measured expression of 6,400 yeast genes at 7 checkpoints before and after the diauxic shift.

**Goal:** partition all yeast genes into clusters so that:
- genes in the *same* cluster have similar behavior
- genes in *different* clusters have different behavior

**6,400 x 7 gene expression matrix**
Genes as Points in Multidimensional Space

<table>
<thead>
<tr>
<th>Gene</th>
<th>Expression Vector</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>

- **n x m** gene expression matrix
- **n** points in **m**-dimensional space
MammaPrint: a test that evaluates the likelihood of breast cancer recurrence based on the expression of just 70 genes.

But how did scientists discover these 70 human genes?
Toward a Computational Problem

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

- 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!
What is the “best” partition into three clusters?
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 minimize 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:

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

$$\text{MaxDistance}(\text{Data, Centers}) = \max \text{ all points DataPoint from Data} \ d(\text{DataPoint, Centers})$$
\textbf{\textit{k}-Center Clustering Problem.} Given a set of points \textit{Data}, find \textit{k} centers minimizing \textit{MaxDistance(\textit{Data, Centers})}.

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

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

**FarthestFirstTraversal**($Data$, $k$)

- **Centers** $\leftarrow$ the set consisting of a single $DataPoint$ from $Data$
- **while** **Centers** have fewer than $k$ points
- $DataPoint$ $\leftarrow$ a point in $Data$ maximizing $d(DataPoint, Centers)$ among all data points
- **add** $DataPoint$ to **Centers**
$k$-Center Clustering Heuristic

**FarthestFirstTraversal**($Data$, $k$)

1. $Centers \leftarrow$ the set consisting of a single $DataPoint$ from $Data$
2. while $Centers$ have fewer than $k$ points
   1. $DataPoint \leftarrow$ a point in $Data$ maximizing $d(DataPoint, Centers)$ among all data points
   2. add $DataPoint$ to $Centers$
What Is Wrong with **FarthestFirstTraversal**?

**FarthestFirstTraversal** selects *Centers* that minimize $\text{MaxDistance}(\text{Data}, \text{Centers})$.

But biologists are interested in *typical* rather than *maximum* deviations, since maximum deviations may represent *outliers* (experimental errors).
The **maximal distance** between *Data* and *Centers*:

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

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

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

**A single** data point contributes to *MaxDistance*

**All** data points contribute to *Distortion*
**k-Means Clustering Problem**

**k-Center Clustering Problem:**
- **Input:** A set of points $Data$ and an integer $k$.
- **Output:** A set of $k$ points $Centers$ that minimizes
  $MaxDistance(DataPoints, Centers)$
  over all choices of $Centers$.

**k-Means Clustering Problem:**
- **Input:** A set of points $Data$ and an integer $k$.
- **Output:** A set of $k$ points $Centers$ that minimizes
  $Distortion(Data, Centers)$
  over all choices of $Centers$.

*NP-Hard for $k > 1$*
$k$-Means Clustering for $k = 1$

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

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

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

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

$$((2+4+6)/3, (3+1+5)/3) = (4, 3)$$
The Lloyd Algorithm in Action

Select $k$ arbitrary data points as Centers
The Lloyd Algorithm in Action

Clusters

assign each data point to its nearest center

Centers
The Lloyd Algorithm in Action

new centers ← clusters’ centers of gravity
The Lloyd Algorithm in Action

assign each data point to its nearest center

Clusters

Centers

again!
The Lloyd Algorithm in Action

new centers ↔ clusters’ centers of gravity
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* minimizing the distortion (the Center of Gravity Theorem).
RECAP
Clustering Yeast Genes

Cluster 1

Cluster 2

Cluster 3

Cluster 4

Cluster 5

Cluster 6
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 $r_{\text{red}}$ ($r_{\text{blue}} + r_{\text{red}} = 1$)
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 maximized at $\theta = i/n$ (most likely bias).
Flipping Two Biased Coins

**Data**

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTTHTTTHTH</td>
<td>0.4</td>
</tr>
<tr>
<td>HHHHTHHHHH</td>
<td>0.9</td>
</tr>
<tr>
<td>HTTHTHHHTHH</td>
<td>0.8</td>
</tr>
<tr>
<td>HTTTTHTHTTT</td>
<td>0.3</td>
</tr>
<tr>
<td>THHHHTHHHTH</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>HTTTHTTHTH</td>
<td>0.4</td>
</tr>
<tr>
<td>HHHHTHHHHH</td>
<td>0.9</td>
</tr>
<tr>
<td>HTHHHHHHTHHH</td>
<td>0.8</td>
</tr>
<tr>
<td>HTTTTTTHHTTT</td>
<td>0.3</td>
</tr>
<tr>
<td>THHHHHHTHHHTH</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Goal: estimate Parameters = (θ_A, θ_B) when HiddenVector is given
If We Knew Which Coin Was Used in Each Sequence...

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

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

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

<table>
<thead>
<tr>
<th><em>HiddenVector</em></th>
<th>Data</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHTHTHHTTH</td>
<td>0.4</td>
<td>*</td>
</tr>
<tr>
<td>HHHHTHHHHH</td>
<td>0.9</td>
<td>*</td>
</tr>
<tr>
<td>HTHHHHHHTHH</td>
<td>0.8</td>
<td>*</td>
</tr>
<tr>
<td>HHTTTTTTHHTT</td>
<td>0.3</td>
<td>*</td>
</tr>
<tr>
<td>THHHTHHHTH</td>
<td>0.7</td>
<td>*</td>
</tr>
</tbody>
</table>

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

$\sum_{all \ data \ points} Data_i \times HiddenVector_i / \sum_{all \ data \ points} HiddenVector_i = 0.35$

*Data* $\times$ *HiddenVector* $= (1,1,...,1)*HiddenVector = 0.35$

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

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

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHTTHHTTH</td>
<td>0.4 * 1</td>
<td>(0.35, 0.80)</td>
</tr>
<tr>
<td>HHHHTHNNNN</td>
<td>0.9 * 0</td>
<td></td>
</tr>
<tr>
<td>HTHHHHHTHH</td>
<td>0.8 * 0</td>
<td></td>
</tr>
<tr>
<td>HHTTTTTHTTT</td>
<td>0.3 * 1</td>
<td></td>
</tr>
<tr>
<td>TTHHTTHHHTH</td>
<td>0.7 * 0</td>
<td></td>
</tr>
</tbody>
</table>

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

\[= \frac{(9+8+7)}{(10+10+10)} = \frac{(0.9+0.8+0.7)}{(1+1+1)} = 0.80\]

\[= \frac{(0.5*0+0.9*1+0.8*1+0.4*0+0.7*1)}{(0+1+1+0+1)} = 0.80\]

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

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

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

\(\text{HHTTHTTTTHTH}\)  0.4  *  1
\(\text{HHHHTTHHHHH}\)  0.9  *  0  
(0.35, 0.80)
\(\text{HHTHHHHTHTHH}\)  0.8  *  0
\(\text{HTTTTTHTHTT}\)  0.3  *  1
\(\text{THHHHTHHHTH}\)  0.7  *  0

\(\theta_A = \text{fraction of heads generated in all flips with coin } A\)
\(= (0.4+0.3)/2=0.35\)
\(= \text{Data} \times \text{HiddenVector} / 1 \times \text{HiddenVector}\)

\(\theta_B = \text{fraction of heads generated in all flips with coin } B\)
\(= (0.9+0.8+0.7)/3=0.80\)
\(= \text{Data} \times (1-\text{HiddenVector}) / 1 \times (1 - \text{HiddenVector})\)
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>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>

454
Data, HiddenVector, Parameters

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

<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>

455
From *Data & Parameters to Hidden Vector*

<table>
<thead>
<tr>
<th>Data</th>
<th>Hidden Vector</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)?

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

<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>?</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)?

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

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

<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>?</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 2\(^{nd}\) sequence (with 9 H)?

\[
\text{Pr}(2^{\text{nd}} \text{ sequence}|\theta_A) = \theta_A^9 \cdot (1-\theta_A)^1 = 0.35^9 \cdot 0.65^1 \approx 0.00005 < \\
\text{Pr}(2^{\text{nd}} \text{ sequence}|\theta_B) = \theta_B^9 \cdot (1-\theta_B)^1 = 0.80^9 \cdot 0.20^1 \approx 0.02684
\]
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>( \theta_A , \theta_B ) (= (0.35, 0.80) )</td>
</tr>
<tr>
<td>0.9</td>
<td>0</td>
<td>( \theta_A ) (= 0.35 ) &lt; ( \theta_B ) (= 0.80 )</td>
</tr>
<tr>
<td>0.8</td>
<td>?</td>
<td>( \theta_B ) (= 0.80 )</td>
</tr>
<tr>
<td>0.3</td>
<td>?</td>
<td>( \theta_A ) (= 0.35 ) &gt; ( \theta_B ) (= 0.80 )</td>
</tr>
<tr>
<td>0.7</td>
<td>?</td>
<td>( \theta_A ) (= 0.35 ) &gt; ( \theta_B ) (= 0.80 )</td>
</tr>
</tbody>
</table>

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

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

<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></td>
</tr>
<tr>
<td>0.9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>0</td>
<td>(0.35, 0.80)</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

Data

Iterate!

HiddenVector

Parameters

Iterate!
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.
Data \quad HiddenVector \quad Parameters = (\theta_A, \theta_B)

\begin{align*}
0.4 & \quad ? \\
0.9 & \quad ? \\
0.8 & \quad ? \quad \rightarrow (0.60, 0.82) \\
0.3 & \quad ? \\
0.7 & \quad ?
\end{align*}

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

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

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

\[
\begin{array}{c|c}
0.4 & 1 \\
0.9 & ? \\
0.8 & ? \\
0.3 & ? \\
0.7 & ? \\
\end{array}
\]

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

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

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

$$\frac{0.000531}{(0.000531 + 0.000015)} \approx 0.97$$
$$\frac{0.000015}{(0.000531 + 0.000015)} \approx 0.03$$
From *Data & Parameters to HiddenMatrix*

**Data** | **HiddenMatrix** | **Parameters** = \((\theta_A, \theta_B)\)
---|---|---
0.4  | 0.97 0.03
0.9  | 0.12 0.88
0.8  | ?      (0.60, 0.82)
0.3  | ?
0.7  | ?

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

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

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

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenMatrix</th>
<th>Parameters = (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>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 \(=(θ_A, θ_B)\)

| H T T T H T T H T H | 0.4 | * | 1 |
| H H H H T H H H H H | 0.9 | * | 0 |
| H T H H H H H T H H | 0.8 | * | 0 |
| H T T T T T T H H T | 0.3 | * | 1 |
| T H H H H H H H T H | 0.7 | * | 0 |

\[ θ_A = Data \times \text{HiddenVector} \div 1 \times \text{HiddenVector} \]

\[ θ_B = Data \times (1-\text{HiddenVector}) \div 1 \times (1-\text{HiddenVector}) \]
From *Data & HiddenMatrix* to *Parameters*

<table>
<thead>
<tr>
<th>HiddenVector</th>
<th>Data</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHHHTHHHHH</td>
<td>0.9</td>
<td>0</td>
</tr>
<tr>
<td>HTHHNTTHH</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>HHTTTHTHTH</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>TTHHTHHHTH</td>
<td>0.7</td>
<td>0</td>
</tr>
</tbody>
</table>

θ<sub>A</sub> = Data * HiddenVector / 1 * HiddenVector

θ<sub>B</sub> = Data * (1-HiddenVector) / 1 * (1-HiddenVector)

*HiddenVector* = (1 0 0 1 0)

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

*Data*    *HiddenVector*    *Parameters* = (θ<sub>A</sub>, θ<sub>B</sub>)

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTHHTTTTHHH</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>HHHHTHHHHHH</td>
<td>0.9</td>
<td>0</td>
</tr>
<tr>
<td>HTHHHHHTHH</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>HHTTTTTHTTT</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>THHHHTHHHTH</td>
<td>0.7</td>
<td>0</td>
</tr>
</tbody>
</table>

θ<sub>A</sub> = *Data* * HiddenVector / 1 * HiddenVector

θ<sub>A</sub> = *Data* * 1<sup>st</sup> row of *HiddenMatrix* / 1 * 1<sup>st</sup> row of *HiddenMatrix*

θ<sub>B</sub> = *Data* * (1-HiddenVector) / 1 * (1-HiddenVector)

θ<sub>B</sub> = *Data* * 2<sup>nd</sup> row of *HiddenMatrix* / 1 * 2<sup>nd</sup> row of *HiddenMatrix*

HiddenVector = ( 1  0  0  1  0 )

Hidden Matrix =  

1  0  0  1  0  0 = HiddenVector  
0  1  1  0  1  1 = 1 - HiddenVector  

"A77"
From *Data & Hidden Matrix* to **Parameters**

\[
\begin{align*}
\text{Data} & \quad \text{Hidden Matrix} & \quad \text{Parameters} = (\theta_A, \theta_B) \\
\begin{bmatrix} H & T & T & T & H & T & H & T & H \end{bmatrix} & \begin{bmatrix} 0.4 & 0.97 & 0.03 \\
0.9 & 0.12 & 0.88 \\
0.8 & 0.29 & 0.71 \\
0.3 & 0.99 & 0.01 \\
0.7 & 0.55 & 0.45 \\
\end{bmatrix} & \begin{bmatrix} \theta_A \end{bmatrix} = \text{Data} \times \text{Hidden Vector} / 1 \times \text{Hidden Vector} \\
\begin{bmatrix} \theta_B \end{bmatrix} = \text{Data} \times (1 - \text{Hidden Vector}) / 1 \times (1 - \text{Hidden Vector}) \\
\end{align*}
\]

\[
\begin{align*}
\text{Hidden Vector} &= \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \end{bmatrix} \\
\text{Hidden Matrix} &= \begin{bmatrix} 0.97 & 0.03 & 0.29 & 0.99 & 0.55 \\
0.03 & 0.97 & 0.71 & 0.01 & 0.45 \end{bmatrix}
\end{align*}
\]
From *HiddenVector* to *HiddenMatrix*

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

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

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

<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>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</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>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

![Diagram](image)
From HiddenVector to HiddenMatrix

Data: data points \( Data = \{Data_1, \ldots , Data_n\} \)

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

\(\text{HiddenMatrix}_{i,j}: \) responsibility of center \( i \) for data point \( j \)

\[
\begin{array}{cccccccc}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} & \text{G} \\
1 & 0.7 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
2 & 0.2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
3 & 0.1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[1 \rightarrow A \rightarrow 2 \rightarrow B \rightarrow 2 \rightarrow E \rightarrow 3 \rightarrow \text{F} \rightarrow 1 \rightarrow \text{D} \rightarrow 3 \rightarrow \text{G} \rightarrow 3 \rightarrow \text{H} \rightarrow 3 \]
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 \( j \)

<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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.70</td>
<td>0.15</td>
<td>0.73</td>
<td>0.40</td>
<td>0.15</td>
<td>0.80</td>
<td>0.05</td>
</tr>
<tr>
<td>2</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>
</tr>
<tr>
<td>3</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>
</tr>
</tbody>
</table>

*HiddenMatrix*
Responsibilities and the Law of Gravitation

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

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

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

The 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.

The Hidden Matrix $H_{ij}$ is defined as

$$HiddenMatrix_{ij} = \frac{Force_{i,j}}{\sum_{all\ centers\ j} Force_{i,j}}$$
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.
Clusters often have **subclusters**, which have sub-subclusters, and so on.
From Data to a Tree

To capture stratification, the **hierarchical clustering** algorithm organizes $n$ data points into a tree.
From a Tree to a Partition into 4 Clusters

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

To capture stratification, the **hierarchical clustering** algorithm first organizes $n$ 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.

### Distance Matrix

<table>
<thead>
<tr>
<th></th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>$g_3$</th>
<th>$g_4$</th>
<th>$g_5$</th>
<th>$g_6$</th>
<th>$g_7$</th>
<th>$g_8$</th>
<th>$g_9$</th>
<th>$g_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_1$</td>
<td>0.0</td>
<td>8.1</td>
<td>9.2</td>
<td>7.7</td>
<td>9.3</td>
<td>2.3</td>
<td>5.1</td>
<td>10.2</td>
<td>6.1</td>
<td>7.0</td>
</tr>
<tr>
<td>$g_2$</td>
<td>8.1</td>
<td>0.0</td>
<td>12.0</td>
<td>0.9</td>
<td>12.0</td>
<td>9.5</td>
<td>12.8</td>
<td>2.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>$g_3$</td>
<td>9.2</td>
<td>12.0</td>
<td>0.0</td>
<td>11.2</td>
<td>0.7</td>
<td>11.1</td>
<td>8.1</td>
<td>1.1</td>
<td>10.5</td>
<td>11.5</td>
</tr>
<tr>
<td>$g_4$</td>
<td>7.7</td>
<td>0.9</td>
<td>11.2</td>
<td>0.0</td>
<td>11.2</td>
<td>9.2</td>
<td>9.5</td>
<td>12.0</td>
<td>1.6</td>
<td>1.1</td>
</tr>
<tr>
<td>$g_5$</td>
<td>9.3</td>
<td>12.0</td>
<td>0.7</td>
<td>11.2</td>
<td>0.0</td>
<td>11.2</td>
<td>8.5</td>
<td>1.0</td>
<td>10.6</td>
<td>11.6</td>
</tr>
<tr>
<td>$g_6$</td>
<td>2.3</td>
<td>9.5</td>
<td>11.1</td>
<td>9.2</td>
<td>11.2</td>
<td>0.0</td>
<td>5.6</td>
<td>12.1</td>
<td>7.7</td>
<td>8.5</td>
</tr>
<tr>
<td>$g_7$</td>
<td>5.1</td>
<td>10.1</td>
<td>8.1</td>
<td>9.5</td>
<td>8.5</td>
<td>5.6</td>
<td>0.0</td>
<td>9.1</td>
<td>8.3</td>
<td>9.3</td>
</tr>
<tr>
<td>$g_8$</td>
<td>10.2</td>
<td>12.8</td>
<td>1.1</td>
<td>12.0</td>
<td>1.0</td>
<td>12.1</td>
<td>9.1</td>
<td>0.0</td>
<td>11.4</td>
<td>12.4</td>
</tr>
<tr>
<td>$g_9$</td>
<td>6.1</td>
<td>2.0</td>
<td>10.5</td>
<td>1.6</td>
<td>10.6</td>
<td>7.7</td>
<td>8.3</td>
<td>11.4</td>
<td>0.0</td>
<td>1.1</td>
</tr>
<tr>
<td>$g_{10}$</td>
<td>7.0</td>
<td>1.0</td>
<td>11.5</td>
<td>1.1</td>
<td>11.6</td>
<td>8.5</td>
<td>9.3</td>
<td>12.4</td>
<td>1.1</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Constructing the Tree

Identify the two closest clusters and merge them.

\[
\{g_3, g_5\}
\]
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

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

<table>
<thead>
<tr>
<th></th>
<th>g₁</th>
<th>g₂</th>
<th>g₃</th>
<th>g₄</th>
<th>g₅</th>
<th>g₆</th>
<th>g₇</th>
<th>g₈</th>
<th>g₉</th>
<th>g₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>g₁</td>
<td>0.0</td>
<td>7.7</td>
<td>9.2</td>
<td>2.3</td>
<td>5.1</td>
<td>10.2</td>
<td>6.1</td>
<td>7.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g₂</td>
<td>7.7</td>
<td>0.0</td>
<td>11.2</td>
<td>9.2</td>
<td>9.5</td>
<td>12.0</td>
<td>1.6</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g₃</td>
<td>9.2</td>
<td>11.2</td>
<td>0.0</td>
<td>11.1</td>
<td>8.1</td>
<td>1.0</td>
<td>10.5</td>
<td>11.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g₄</td>
<td>2.3</td>
<td>9.2</td>
<td>11.1</td>
<td>0.0</td>
<td>5.6</td>
<td>12.1</td>
<td>7.7</td>
<td>8.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g₅</td>
<td>5.1</td>
<td>9.5</td>
<td>8.1</td>
<td>5.6</td>
<td>0.0</td>
<td>9.1</td>
<td>8.3</td>
<td>9.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g₆</td>
<td>10.2</td>
<td>12.0</td>
<td>1.0</td>
<td>12.1</td>
<td>9.1</td>
<td>0.0</td>
<td>11.4</td>
<td>12.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g₇</td>
<td>6.1</td>
<td>1.6</td>
<td>10.5</td>
<td>7.7</td>
<td>8.3</td>
<td>11.4</td>
<td>0.0</td>
<td>1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g₈</td>
<td>7.0</td>
<td>1.0</td>
<td>11.5</td>
<td>8.5</td>
<td>9.3</td>
<td>12.4</td>
<td>1.1</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Constructing the Tree

Identify the two closest clusters and merge them.

\[
\{g_3, g_5, g_8\}, \{g_2, g_4\}, \{g_3, g_5, g_8\}, g_7, g_2, g_4, g_9, g_10
\]
Constructing the Tree

Iterate until all elements form a single cluster (root).
Constructing a Tree from a Distance Matrix $D$

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:

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

**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

Unlike most clustering algorithms, the MCL (micans.org/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.

You can find the code 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
The number of steps to converge is not proven, but experimentally shown to be 10 to 100 steps, and mostly consist of sparse matrices after the first few steps.

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

Input: A weighted undirected graph $G = (V, E)$, expansion parameter $e$, inflation parameter $r$

Output: A partitioning of $V$ into disjoint components

$M \leftarrow M(G)$

while $M$ is not fixpoint do
  $M \leftarrow M^e$
  forall $i \in V$ do
    forall $j \in V$ do
      $M[i][j] \leftarrow M[i][j]^r$
    forall $j \in V$ do
      $M[i][j] \leftarrow \frac{M[i][j]}{\sum_{k \in V} M[i][k]}$

$H \leftarrow$ graph induced by non-zero entries of $M$

$C \leftarrow$ clustering induced by connected components of $H$
A popular method for exploring high-dimensional data is something called t-SNE, introduced by van der Maaten and Hinton in 2008. The technique has become widespread in the field of machine learning, since it has an almost magical ability to create compelling two-dimensional “maps” from data with hundreds or even thousands of dimensions.

The goal is to take a set of points in a high-dimensional space and find a faithful representation of those points in a lower-dimensional space, typically the 2D plane. The algorithm is non-linear and adapts to the underlying data, performing different transformations on different regions. Those differences can be a major source of confusion.
A second feature of t-SNE is a tuneable parameter, “perplexity,” which says (loosely) how to balance attention between local and global aspects of your data. The parameter is, in a sense, a guess about the number of close neighbors each point has. The original paper says, “The performance of SNE is fairly robust to changes in the perplexity, and typical values are between 5 and 50.” But the story is more nuanced than that. Getting the most from t-SNE may mean analyzing multiple plots with different perplexities.
Stochastic Neighbor Embedding: key points

**t-SNE: The effect of various perplexity values on the shape**

An illustration of t-SNE on the two concentric circles and the S-curve datasets for different perplexity values.

We observe a tendency towards clearer shapes as the perplexity value increases.

The size, the distance and the shape of clusters may vary upon initialization, perplexity values and does not always convey a meaning.

As shown below, t-SNE for higher perplexities finds meaningful topology of two concentric circles, however the size and the distance of the circles varies slightly from the original. Contrary to the two circles dataset, the shapes visually diverge from S-curve topology on the S-curve dataset even for larger perplexity values.

For further details, "How to Use t-SNE Effectively" [http://distill.pub/2016/misread-tsne/](http://distill.pub/2016/misread-tsne/) provides a good discussion of the effects of various parameters, as well as interactive plots to explore those effects.
First convert each high-dimensional similarity into the probability that one data point will pick the other data point as its neighbor. To evaluate a map:

- Use the pairwise distances in the low-dimensional map to define the probability that a map point will pick another map point as its neighbor.
- Compute the Kullback-Leibler divergence between the probabilities in the high-dimensional and low-dimensional spaces.
- Each point in high-Dimension has a conditional probability of picking each other point as its neighbor.
- The distribution over neighbors is based on the high-Dimension pairwise distances.
Evaluate this representation by seeing how well the low-Dimension probabilities model the high-Dimension ones.
Stochastic Neighbor Embedding (SNE) is the process of constructing conditional probabilities representing the similarity between high dimensional data points using their Euclidean distances. The conditional probability \( p_{j|i} \) for points \( x_j \) and \( x_i \) is defined by the equation

\[
p_{j|i} = \frac{\exp\left(-\frac{||x_i - x_j||^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{||x_i - x_k||^2}{2\sigma_i^2}\right)}
\]
Similarity is ultimately the probability that $x_i$ would define $x_j$ as a neighbor, in which a neighborhood is defined by a Gaussian probability density centered at $x_i$. Where $\sigma_i$ is the variance of the $x_i$-centered distribution.

A large $p_{j|i}$ is indicative of close, or similar, data points, and a very small $p_{j|i}$ means that $x_j$ is not likely a neighbor of $x_i$.

Instead of using a Gaussian distribution, t-SNE assumes the closely-related Student-t distribution to compute the pairwise conditional probabilities in a low-dimensional space more efficiently.
Stochastic Neighbor Embedding

The t-SNE algorithm improves upon the original SNE algorithm by implementing a cost function with a simpler gradient that uses the Kullback-Leibler divergence (DKL) between the high-dimensional joint probability distribution $P$ and a low-dimensional Student-t based joint probability distribution $Q$ (Equation 2). The gradient is explicitly defined in Equation 3.

\[
q_{ij} = \frac{(1 + ||x_i - x_j||^2)^{-1}}{\sum_{k \neq l} (1 + ||y_k - y_l||^2)^{-1}}
\]

\[
\frac{\delta C}{\delta \mathbf{y}} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}
\]
With higher-dimensional data, one runs the risk of overcrowding the projection such that dissimilarities between points cannot be faithfully plotted due to a lack of space in the two-dimensional map to reduce the high-dimensional data.

The use of the heavy-tailed Student-t distribution mitigates this issue because it converts the moderate distances that, when mapped to a two-dimensional plane tend to be too close to $x_i$, to probabilities that map the points an appropriately greater distance away.
Stochastic Neighbor Embedding

**Algorithm 1**: Standard t-distributed Stochastic Neighbor Embedding Algorithm.

**Data**: data set $X = x_1, x_2, ..., x_n$,

cost function parameters: perplexity $Perp$;

optimization parameters: number of iterations $T$, learning rate $\eta$, momentum $\alpha(t)$;

**Result**: low-dimensional data representation $\mathcal{Y}(T) = y_1, y_2, y_n$.

begin

compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (Equation 1)

set $p_{i,j} = \frac{p_{j|i} + p_{i|j}}{2n}$;

sample initial solution $\mathcal{Y}(0) = y_1, y_2, y_n$ from $\mathcal{N}(0, 10^{-4}I)$;

for $t = 1$ to $T$ do

compute low-dimensional affinities $q_{ij}$ (Equation 2)

compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (Equation 3)

set $\mathcal{Y}(t) = \mathcal{Y}(t-1) + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)(\mathcal{Y}^{t-1} - \mathcal{Y}^{t-2})$;

end

end
References on t-SNE


• useful video: https://lvdmaaten.github.io/tsne/)https://youtu.be/RJVL80Gg3lA?list=UUtXKDgv1AVoG88PLl8nGXmw)

• how to use: https://distill.pub/2016/misread-tsne/
Burrows (left), Wheeler (right) both at the Computer Laboratory

Introduction

BWA is a software package for mapping low-divergent sequences against a large reference genome, such as the human genome. It consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. The first algorithm is designed for Illumina sequence reads up to 100bp, while the rest two for longer sequences ranged from 70bp to 1Mbp. BWA-MEM and BWA-SW share similar features such as long-read support and split alignment, but BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate. BWA-MEM also has better performance than BWA-backtrack for 70–100bp Illumina reads.

BWA:
- SF project page
- SF download page
- Mailing list
- BWA manual page
- Repository

Links:
- SAMtools
Burrows Wheeler Transform

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

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

To recreate T from BWT(T), repeatedly apply the rule: $T = \text{BWT}(\text{LF}(i)) + T$; $i = \text{LF}(i)$ 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 = \text{LF}[2] = 6; T = g\$. Third step: $s = \text{LF}[6] = 5; T = cg\$. 

Final T
Burrows-Wheeler Transform (BWT)

acaacg$

BWT

$acaacg
aacg$ac
acaacg$
acg$aca
caacg$a
cg$acaa
g$acaac

Burrows-Wheeler Matrix (BWM)
Burrows-Wheeler Matrix

$acaacg$

aacg$ac$

acaacg$

acg$aca$

caacg$a$

cg$aca$

g$aaca$
Burrows-Wheeler Matrix

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>aca</td>
<td>g</td>
<td>$ac</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>caa</td>
<td>acg$</td>
</tr>
<tr>
<td>3</td>
<td>aac</td>
<td>g</td>
<td>$ac</td>
</tr>
<tr>
<td>4</td>
<td>acg</td>
<td>$aca</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CG</td>
<td>$aca</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td>$acaac</td>
<td></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.
Burrow Wheeler Transform

(a)  
\[
\begin{align*}
\text{a c a a c g} & \rightarrow \text{a c g s a c a } \rightarrow \text{g c s a a a c} \\
\text{a c a a c g $} & \rightarrow \text{a c g $ s a c a } \rightarrow \text{g c $ s a a a c} \\
\end{align*}
\]

(b)  
\[
\begin{align*}
\text{g} & \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \\
\text{g} & \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \\
\end{align*}
\]

(c)  
\[
\begin{align*}
\text{a a c} & \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \\
\text{a a c} & \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \rightarrow \text{s a c a a c g} \\
\end{align*}
\]
Genome Assembly

• Why do we map reads?
• Using the Trie
• From a Trie to a Suffix Tree
• String Compression and the Burrows-Wheeler Transform
• Inverting Burrows-Wheeler
• Using Burrows-Wheeler for Pattern Matching
• Finding the Matched Patterns
• Setting Up Checkpoints
• Inexact Matching
Toward a Computational Problem

• **Reference genome**: database genome used for comparison.

• **Question**: How can we assemble individual genomes efficiently using the reference?

CTGA_{T}GATGGACTACGCTACTACTGCTAGCTGT_{AT} \quad \text{Individual}

CTGA_{G}GATGGACTACGCTACTACTG_{A}TAGCTGT_{TT} \quad \text{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

...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.

<table>
<thead>
<tr>
<th>CTGAGGATGGACTACGCTACTACTACTGATAGCTGT</th>
<th>Reference Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAGGA CCACCG TGA-A</td>
<td></td>
</tr>
</tbody>
</table>
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.

  panamabananana Genome
  nana Pattern

• **Note**: we use words instead of DNA strings for convenience.
Brute Force Is Too Slow

- The runtime of the brute force approach is too high!
  - Single *Pattern*: $O(|Genome| \times |Pattern|)$
  - 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.
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Using the Trie for Pattern Matching

- **TrieMatching**: Slide the trie down the genome.

- At each position, walk down the trie and see if we can reach a leaf by matching symbols.

- Analogy: bus stops
Success!

• Runtime of Brute Force:
  – Total: $O(|Genome| \times |Patterns|)$

• Runtime of Trie Matching:
  – Trie Construction: $O(|Patterns|)$
  – Pattern Matching: $O(|Genome| \times |LongestPattern|)$
Son completely forgot about memory!

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 $n$, $|\text{Suffixes}| = n(n - 1)/2 = O(n^2)$

Suffixes
panamabananas$
\text{anamabananas}$
\text{namabanananas}$
\text{amabanananas}$
\text{mabananas}$
\text{abananas}$
\text{bananas}$
\text{anananas}$
\text{nanas}$
\text{anas}$
\text{nas}$
as$
s$
$
Compressing the Trie

• This doesn’t mean that our idea was bad!

• To reduce memory, we can compress each “nonbranching path” of the tree into an edge.
• This data structure is called a **suffix tree**.

• For any **Genome**, # nodes < 2|Genome|.
  – # **leaves** = |Genome|;
  – # **internal nodes** < |Genome| – 1
Runtime and Memory Analysis

• 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.
Runtime and Memory Analysis

• 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: $O(|\text{Genome}| + |\text{Patterns}|)$
  • Memory: $O(|\text{Genome}|)$

• However, big-O notation ignores constants!
  • The best known suffix tree implementations require ~ 20 times the length of $|\text{Genome}|$.
  • 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.

\[
\text{Genome} \\
\text{GGGGGGGGGGCCCCCCCCCCCAAAAAAATTTTTTTTTTTTTTTTCCCCCG} \\
\downarrow \\
\text{10G11C7A15T5C1G} \\
\text{Run-length encoding}
\]

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

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

How do we do this step?

1. **Genome**
2. Convert repeats to runs
3. **Genome***
4. Run-length encoding
5. **CompressedGenome***
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabananas$”

The Burrows-Wheeler Transform

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

Form all cyclic rotations of “panamabanananas$”

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

```
panamabanananas$
$panamabanananas
$s$panamabanana
$as$panamabanan
$nas$panamabana
$anas$panamaban
$nanas$panamaba
$ananas$panamab
$bananas$panama
$abananas$panam
$mabanananas$pana
$amabanananas$pan
$namabanananas$pa
$ananabananas$p
```

```
$panamabanananas
abananas$panam
amabanananas$pan
anamabanananas$p
ananas$panamab
anas$panamaban
as$panamabana
bananas$panama
mabananas$pana
namabanananas$p
nanas$panamaba
nas$panamaban
panamabanananas$
```

Form all cyclic rotations of “panamabananas$”

Burrows-Wheeler Transform:
Last column = smnpbnnaaaaa$
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

We now know 2-mer composition of the circular string banana$

Sorting gives us the first 2 columns of the matrix.
Reconstructing banana

- We now know 3-mer composition of the circular string banana$
- Sorting gives us the first 3 columns of the matrix.
Reconstructing banana

\[
\begin{array}{@{}ccc@{}c@{}c@{}}
\$banana & a \$ ba & \$ban \\
a\$banana & na \$ b & a\$bb \\
anana\$ban & nana & ana a \\
anana\$b & bana & ana a \\
banana\$ & $ban & bann \\
a\$bana & ana$ & na\$b \\
nana\$ba & anan & nana \\
\end{array}
\]

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

\[
\begin{align*}
\text{$banana$} & \quad \text{a$ban$} & \quad \text{$bana$} \\
\text{a$banan$} & \quad \text{na$ba$} & \quad \text{a$bbn$} \\
\text{ana$ban$} & \quad \text{nana$}$ & \quad \text{anaab} \\
\text{anana$b$} & \quad \text{banan} & \quad \text{anaaa} \\
\text{banana$}$ & \quad \text{$bana$} & \quad \text{bannn} \\
\text{na$ba$} & \quad \text{ana$b$} & \quad \text{na$ba$} \\
\text{nana$ba$} & \quad \text{nana$}$ & \quad \text{nana$}$
\end{align*}
\]

- We now know 5-mer composition of the circular string \text{banana$}$
- Sorting gives us the first 5 columns of the matrix.
Reconstructing banana

\[
\begin{array}{ccc}
$banana$ & a$banana$ & $banan$ \\
a$banan$ & na$ban$ & a$bbna$ \\
anan$ban$ & nana$ba$ & anaaba$ \\
anana$ban$ & banana & anaaa$a$ \\
banana$ & $banan$ & bannna$ \\
nana$ba$ & ana$ba$ & na$ban$ \\
nana$ba$ & anana$ & na$ba$
\end{array}
\]

\[6\text{-mers}\]

\[\text{Sort}\]

• We now know 6-mer composition of the circular string banana$

• Sorting gives us the first 6 columns of the matrix.
Reconstructing banana

$\text{banana}$

\begin{align*}
\text{a$\text{banan}$} & \quad \text{a$\text{banan}$} & \quad \text{a$\text{bbna}$} \\
\text{ana$\text{ban}$} & \quad \text{na$\text{ban}$} & \quad \text{ana$\text{aba}$} \\
\text{anana$\text{b}$} & \quad \text{bana$\text{na}$} & \quad \text{ana$\text{aaa}$} \\
\text{bana$\text{na}$} & \quad \text{banan$\text{a}$} & \quad \text{bann$\text{na}$} \\
\text{na$\text{bana}$} & \quad \text{ana$\text{ba}$} & \quad \text{na$\text{ban}$} \\
\text{nana$\text{ba}$} & \quad \text{anana$\text{a}$} & \quad \text{nana$\text{b}$}
\end{align*}

• We now know 6-mer composition of the circular string $\text{banana}$

• Sorting gives us the first 6 columns of the matrix.
Reconstructing banana

• We now know the entire matrix!

• Taking all elements in the first row (after $\$\$) produces banana.
Reconstructing *Genome* from $BWT(\text{Genome})$ required us to store $|\text{Genome}|$ copies of $|\text{Genome}|$.

\[
\begin{align*}
&\text{banana} \\
&a\text{banana} \\
&\text{anana}b \\
&\text{anana}b \\
&\text{banan}a$ \\
&\text{nana}ba \\
&\text{nana}ba
\end{align*}
\]

Can we invert BWT with less space?
A Strange Observation

$\text{panama}\text{bananas}\text{s}
\text{abanas}\text{s}\text{panam}
\text{amabanas}\text{s}\text{pan}
\text{anamabanas}\text{s}\text{p}
\text{anananas}\text{s}\text{panamab}
\text{anas}\text{s}\text{panamaban}
\text{as}\text{s}\text{panamaban}
\text{bananas}\text{s}\text{panama}
\text{mananas}\text{s}\text{pana}
\text{ananas}\text{s}\text{panamab}
\text{anas}\text{s}\text{panamaban}
\text{panamabanas}\text{s}
\text{s}\text{panamabanana}\n$
A Strange Observation

$s$ panamabanananas
abanananas$s$ panam
amabanananas$p$ an
anamabanananas$p$
ananas$s$ panamab
anas$s$ panamaban
anas$s$ panamaban
bananas$s$ panama
mabananas$s$ pan
namabanananas$p$a
nanas$s$ panamaba
nas$s$ panamaban
panamabanananas$s$
s$s$ panamabanana
Is It True in General?

These strings are sorted
Is It True in General?

These strings are sorted
Is It True in General?

These strings are sorted

$\text{panama} \text{bananas}$
1. $\text{abana} \text{nana} \text{sa}$
2. $\text{amanana} \text{a} \text{s}$
3. $\text{anana} \text{bananana}$
4. $\text{anana} \text{s} \text{pamab}$
5. $\text{anasa} \text{panamaban}$
6. $\text{as} \text{s} \text{panamaban}$

Chop off $a$

These strings are sorted

$\text{bananas} \text{panama}$
1. $\text{bananas} \text{panama}$
2. $\text{banana} \text{as}$
3. $\text{anana} \text{sa}$
4. $\text{anana} \text{s} \text{panamaba}$
5. $\text{anasa} \text{panamaban}$
6. $\text{as} \text{s} \text{panamaban}$

Still sorted

Add $a$ to end

These strings are sorted

$\text{banana} \text{sa}$
1. $\text{banana} \text{sa}$
2. $\text{bananana} \text{sa}$
3. $\text{anana} \text{sa}$
4. $\text{anana} \text{s} \text{panamaba}$
5. $\text{anasa} \text{panamaban}$
6. $\text{as} \text{s} \text{panamaban}$

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

$s_1$ panamabananas $s_1$
$a_1$ bananas $p_1$
$a_2$ mabananas $p_1$
$a_3$ namabananas $p_1$
$a_4$ nanas $panamab_1$
$a_5$ nas $panamaban_2$
$a_6$ s $panamaban_3$
b_1 ananas $panama_1$
m_1 abanananas $pana_2$
n_1 amabananas $pa_3$
n_2 anas $panamaba_4$
n_3 as $panamaban_5$
p_1 anamabananas $s_1$
s_1 $panamabanana_6$
More Efficient BWT Decompression

$s_1$ panamabananas
$a_1$ bananas$panam_1$
a_2$ mabananas$pan_1$
a_3$ namabanananas$ p_1$
a_4$ nanas$ panamab_1$
a_5$ nas$ panamaban_2$
a_6$ s$ panamaban_3$
b_1$ ananas$ panama_1$
m_1$ abananas$ panab_2$
n_1$ amabananas$ pa_3$
n_2$ anas$ panamaba_4$
n_3$ as$ panamabana_5$
p_1$ anamabananas$ s_1$
s_1$ panamabanana_6
More Efficient BWT Decompression

$1_{panamabananas}s_1$
$a_1_{bananas}s_{panam}_1$
$a_2_{mabanananas}s_{pan}_1$
$a_3_{namabanananas}s_{p_1}$
$a_4_{nanas}s_{panamab}_1$
$a_5_{nas}s_{panamaban}_2$
$a_6_{s}s_{panamabanan}_3$
$b_1_{ananas}s_{panama}_1$
$m_1_{abanananas}s_{pan}_2$
$n_1_{amabananas}s_{pa}_3$
$n_2_{anas}s_{panamaba}_4$
$n_3_{as}s_{panamaban}_5$
$p_1_{anamabananas}s_{1}$
$s_1_{s}s_{panamabanana}_6$

• Memory: $2|Genome| = O(|Genome|)$. 

579
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**

\[
\begin{align*}
\$_{1} & \text{panamabanananas}_{1} \\
a_{1} & \text{bananas}$\text{panam}_{1} \\
a_{2} & \text{mabanananas}$\text{pan}_{1} \\
a_{3} & \text{namabanananas}$\text{p}_{1} \\
a_{4} & \text{ananas}$\text{panamaba}_{1} \\
a_{5} & \text{nas}$\text{panamaban}_{2} \\
a_{6} & \text{s}$\text{panamabanan}_{3} \\
b_{1} & \text{ananas}$\text{panama}_{1} \\
m_{1} & \text{bananas}$\text{pana}_{2} \\
n_{1} & \text{amabanananas}$\text{pa}_{3} \\
n_{2} & \text{ananas}$\text{panamaba}_{4} \\
n_{3} & \text{as}$\text{panamabana}_{5} \\
p_{1} & \text{anamabanananas}_{1} \\
s_{1} & \text{$\text{panamabana}_{a}_{6}\$} \\
\end{align*}
\]
Finding Pattern Matches Using BWT

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

\[
\begin{align*}
$s_1$ & panamabanananas \\
a_1 & bananas $p_1$ \\
a_2 & mabananas $p_1$ \\
a_3 & namabananas $p_1$ \\
a_4 & nanas $p_1$ \\
a_5 & nas $p_1$ \\
a_6 & s $p_1$ \\
b_1 & ananas $p_1$ \\
m_1 & abananas $p_2$ \\
n_1 & amabananas $p_3$ \\
n_2 & anas $p_4$ \\
n_3 & as $p_5$ \\
p_1 & anamabananas $s_1$ \\
s_1 & $s_1$ \\
582
\end{align*}
\]
Finding Pattern Matches Using BWT

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

```
$1 panamabanananas
a1 bananas$ panam1
a2 mabanananas$ pan1
a3 namabanananas$ p1
a4 nanas$ panamab1
a5 nas$ panamaban2
a6 s$panamabana3
b1 ananas$ panama1
m1 abanananas$ pan2
n1 amabanananas$ pa3
n2 anas$ panamaba4
n3 as$ panamaban5
p1 anamabanananas$1
s1 $ panamabananana6
```
Finding Pattern Matches Using BWT

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

\[
\begin{align*}
\text{s} & \quad \text{p} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \quad \text{a} \\
\text{a} & \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{s} \quad \text{p} \quad \text{a} \\
\text{a} & \quad \text{m} \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{s} \\
\text{a} & \quad \text{n} \quad \text{a} \quad \text{m} \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \\
\text{a} & \quad \text{n} \quad \text{a} \quad \text{m} \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \\
\text{a} & \quad \text{n} \quad \text{a} \quad \text{m} \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \\
\text{a} & \quad \text{n} \quad \text{a} \quad \text{m} \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \\
\text{a} & \quad \text{n} \quad \text{a} \quad \text{m} \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \\
\text{a} & \quad \text{n} \quad \text{a} \quad \text{m} \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \\
\text{a} & \quad \text{n} \quad \text{a} \quad \text{m} \quad \text{a} \quad \text{b} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \\
\end{align*}
\]
Where Are the Matches?

• **Multiple Pattern Matching Problem:**
  – **Input:** A collection of strings *Patterns* and a string *Genome*.
  – **Output:** All *positions* in *Genome* where one of *Patterns* appears as a substring.

• Where are the *positions*? BWT has not revealed them.
Where Are the Matches?

• Example: We know that **ana** occurs 3 times, but where?

\[
\begin{align*}
&$1_{panamabanananas} \\
&a_1_{bananas}panam1 \\
&a_2_{mabanananas}pan1 \\
&a_3_{ana mabanananas}p1 \\
&a_4_{nana s}panamab1 \\
&a_5_{na s}panamaban2 \\
&a_6_{s}panamaban3 \\
&b_1_{ananas}panama1 \\
&m_1_{abananas}pana2 \\
&n_1_{amabanananas}pa3 \\
&n_2_{anas}panamab4 \\
&n_3_{as}panamabana5 \\
&p_1_{anamabanananas}1 \\
&s_1_{panamabanana}6
\end{align*}
\]
Using the Suffix Array to Find Matches

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

```
<table>
<thead>
<tr>
<th>Position</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$panamabananas$</td>
</tr>
<tr>
<td>1</td>
<td>panamabananas</td>
</tr>
<tr>
<td>2</td>
<td>mabananas</td>
</tr>
<tr>
<td>3</td>
<td>namabananas</td>
</tr>
<tr>
<td>4</td>
<td>nanas</td>
</tr>
<tr>
<td>5</td>
<td>nas</td>
</tr>
<tr>
<td>6</td>
<td>s</td>
</tr>
<tr>
<td>7</td>
<td>panama</td>
</tr>
<tr>
<td>8</td>
<td>bananas</td>
</tr>
<tr>
<td>9</td>
<td>panama</td>
</tr>
<tr>
<td>10</td>
<td>mabananas</td>
</tr>
<tr>
<td>11</td>
<td>namabananas</td>
</tr>
<tr>
<td>12</td>
<td>nanas</td>
</tr>
<tr>
<td>13</td>
<td>nas</td>
</tr>
<tr>
<td>14</td>
<td>s</td>
</tr>
<tr>
<td>15</td>
<td>panama</td>
</tr>
<tr>
<td>16</td>
<td>bananas</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>1</th>
<th>panamabananas$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>bananas$panam</td>
</tr>
<tr>
<td>3</td>
<td>banananas$pana</td>
</tr>
<tr>
<td>4</td>
<td>mabananas$panamab</td>
</tr>
<tr>
<td>5</td>
<td>namabananas$panamab</td>
</tr>
<tr>
<td>6</td>
<td>nanas$panamabanan</td>
</tr>
<tr>
<td>7</td>
<td>nas$panamabanana</td>
</tr>
<tr>
<td>8</td>
<td>p</td>
</tr>
<tr>
<td>9</td>
<td>anamabananas$panamabana</td>
</tr>
<tr>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td>11</td>
<td>ananas$panama</td>
</tr>
<tr>
<td>12</td>
<td>m</td>
</tr>
<tr>
<td>13</td>
<td>abanananas$pana</td>
</tr>
<tr>
<td>14</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>amabananas$pana</td>
</tr>
<tr>
<td>16</td>
<td>n</td>
</tr>
<tr>
<td>17</td>
<td>anas$panamabana</td>
</tr>
<tr>
<td>18</td>
<td>n</td>
</tr>
<tr>
<td>19</td>
<td>as$panamabana</td>
</tr>
<tr>
<td>20</td>
<td>p</td>
</tr>
<tr>
<td>21</td>
<td>anamabanananas$panamabana</td>
</tr>
<tr>
<td>22</td>
<td>s</td>
</tr>
</tbody>
</table>
Using the Suffix Array to Find Matches

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

```
panamabanananas$
```

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$ _1 $panamabanananas</td>
</tr>
<tr>
<td>2</td>
<td>a _1 bana$as$panam</td>
</tr>
<tr>
<td>3</td>
<td>a _2 mabananas$pan</td>
</tr>
<tr>
<td>4</td>
<td>a _3 namabananas$ p</td>
</tr>
<tr>
<td>5</td>
<td>a _4 nanas$panama $</td>
</tr>
<tr>
<td>6</td>
<td>a _5 nas$panamaban</td>
</tr>
<tr>
<td>7</td>
<td>a _6 s$panamabanana</td>
</tr>
<tr>
<td>8</td>
<td>b _1 ananas$panama</td>
</tr>
<tr>
<td>9</td>
<td>m _1 abanananas$pana</td>
</tr>
<tr>
<td>10</td>
<td>n _1 amabananas$pa</td>
</tr>
<tr>
<td>11</td>
<td>n _2 anas$panamaba</td>
</tr>
<tr>
<td>12</td>
<td>n _3 as$panamabana</td>
</tr>
<tr>
<td>13</td>
<td>p _1 anamabananas$</td>
</tr>
<tr>
<td>14</td>
<td>s _1 $panamabanana</td>
</tr>
</tbody>
</table>
```
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>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>2</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
```

Using the Suffix Array to Find Matches

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

```
panamabanananas$
```

<table>
<thead>
<tr>
<th>13</th>
<th>$1panamabanananas1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>a1bananas$panam1</td>
</tr>
<tr>
<td>3</td>
<td>a2mabanananas$pan1</td>
</tr>
<tr>
<td>1</td>
<td>a3namabanananas$pan1</td>
</tr>
<tr>
<td>1</td>
<td>a4nanas$panamab1</td>
</tr>
<tr>
<td>2</td>
<td>a5nas$panamaban2</td>
</tr>
<tr>
<td>3</td>
<td>a6s$panamabanana3</td>
</tr>
<tr>
<td>1</td>
<td>b1ananas$panama1</td>
</tr>
<tr>
<td>2</td>
<td>m1abanananas$pana2</td>
</tr>
<tr>
<td>3</td>
<td>n1amabanananas$pa3</td>
</tr>
<tr>
<td>4</td>
<td>n2anas$panamaba4</td>
</tr>
<tr>
<td>5</td>
<td>n3as$panamabana5</td>
</tr>
<tr>
<td>1</td>
<td>p1anamabanananas$1</td>
</tr>
<tr>
<td>6</td>
<td>s1$panamabananana6</td>
</tr>
</tbody>
</table>
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>$</th>
<th>panamabanananas</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s1</td>
<td>panamabanananas1</td>
</tr>
<tr>
<td>1</td>
<td>a1</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>3</td>
<td>a2</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>4</td>
<td>a3</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>1</td>
<td>a4</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>2</td>
<td>a5</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>1</td>
<td>a6</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>1</td>
<td>b1</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>2</td>
<td>m1</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>3</td>
<td>n1</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>4</td>
<td>n2</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>5</td>
<td>n3</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>1</td>
<td>p1</td>
<td>panamabanananas</td>
</tr>
<tr>
<td>6</td>
<td>s1</td>
<td>panamabanananas</td>
</tr>
</tbody>
</table>
```

Using the Suffix Array to Find Matches

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

```
panamabanananas$
```

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1$ panamabanananas</td>
<td></td>
</tr>
<tr>
<td>a₁bananas$panam</td>
<td></td>
</tr>
<tr>
<td>a₂mabanananas$pan</td>
<td></td>
</tr>
<tr>
<td>a₃namabanananas$p</td>
<td></td>
</tr>
<tr>
<td>a₄nanas$panamaban</td>
<td></td>
</tr>
<tr>
<td>a₅nas$panamaban</td>
<td></td>
</tr>
<tr>
<td>a₆s$panamabanana</td>
<td></td>
</tr>
<tr>
<td>b₁ananas$panama</td>
<td></td>
</tr>
<tr>
<td>m₁abanananas$pana</td>
<td></td>
</tr>
<tr>
<td>n₁amabanananas$pa</td>
<td></td>
</tr>
<tr>
<td>n₂anas$panamaba</td>
<td></td>
</tr>
<tr>
<td>n₃as$panamabana</td>
<td></td>
</tr>
<tr>
<td>p₁anamabanananas$</td>
<td></td>
</tr>
<tr>
<td>s₁$panamabana</td>
<td></td>
</tr>
</tbody>
</table>

593
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}$
```

```
\begin{array}{|c|}
\hline
1 & \text{$_1$panamabanananas}_1 \\
3 & a_1\text{bananas}$\text{panam}_1 \\
5 & a_2\text{mabanananas}$\text{pan}_1 \\
7 & a_3\text{namabanananas}$p_1 \\
9 & a_4\text{nanas}$\text{panamab}_1 \\
11 & a_5\text{nas}$\text{panamaban}_2 \\
13 & a_6\text{s}$\text{panamabananan}_3 \\
\hline
\end{array}
```
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</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1_{panamabananas}$</td>
<td>$a_1_{bananas}$</td>
<td>$a_2_{mabanananas}$</td>
<td>$a_3_{namabanananas}$</td>
<td>$a_4_{nanas}$</td>
<td>$a_5_{nas}$</td>
</tr>
<tr>
<td>2</td>
<td>$a_6_{ss}$</td>
<td>$b_1_{ananas}$</td>
<td>$m_1_{abanananas}$</td>
<td>$n_1_{amabananas}$</td>
<td>$n_2_{ananas}$</td>
<td>$n_3_{as}$</td>
</tr>
<tr>
<td>3</td>
<td>$p_1_{anamabanananas}$</td>
<td>$p_{1}$</td>
<td>$s_1_{s}$</td>
<td>$s_{1}$</td>
<td>$s_{1}$</td>
<td>$s_{1}$</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>13</th>
<th>$1_{\text{panamabanananas}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$a_1_{\text{bananas}}$panam$1$</td>
</tr>
<tr>
<td>3</td>
<td>$a_2_{\text{mabanananas}}$pan$1$</td>
</tr>
<tr>
<td>1</td>
<td>$a_3_{\text{nabanananas}}$p$1$</td>
</tr>
<tr>
<td>7</td>
<td>$a_4_{\text{nana}}$panamab$1$</td>
</tr>
<tr>
<td>9</td>
<td>$a_5_{\text{nas}}$panamaban$2$</td>
</tr>
<tr>
<td>11</td>
<td>$a_6_{\text{s}}$panamabanana$3$</td>
</tr>
<tr>
<td>6</td>
<td>$b_1_{\text{nanas}}$panama$1$</td>
</tr>
<tr>
<td>4</td>
<td>$m_1_{\text{abananas}}$pana$2$</td>
</tr>
<tr>
<td>2</td>
<td>$n_1_{\text{amabanananas}}$pa$3$</td>
</tr>
<tr>
<td>8</td>
<td>$n_2_{\text{anas}}$panamaba$4$</td>
</tr>
<tr>
<td>10</td>
<td>$n_3_{\text{as}}$panamabana$5$</td>
</tr>
<tr>
<td>0</td>
<td>$p_1_{\text{ananamabanananas}}$1</td>
</tr>
<tr>
<td></td>
<td>$s_1_{\text{panamabanana}}$6</td>
</tr>
</tbody>
</table>
Using the Suffix Array to Find Matches

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

|   |   | $1\text{panamabanananas}_1$
|---|---|---|---|---|---|---|---|
|   |   | $a_1\text{bananas}_1\text{panam}_1$
|   |   | $a_2\text{mabanananas}_1\text{pan}_1$
|   |   | $a_3\text{namabanananas}_1\text{p}_1$
|   |   | $a_4\text{nanas}_1\text{panamab}_1$
|   |   | $a_5\text{nas}_1\text{panamaban}_2$
|   |   | $a_6\text{ss}_1\text{panamabanan}_3$
|   |   | $b_1\text{ananas}_1\text{panama}_1$
|   |   | $m_1\text{abanananas}_1\text{pana}_2$
|   |   | $n_1\text{amabanananas}_1\text{pa}_3$
|   |   | $n_2\text{nanas}_1\text{panamaba}_4$
|   |   | $n_3\text{as}_1\text{panamabana}_5$
|   |   | $p_1\text{anamabanananas}_1$
|   |   | $s_1\text{panamabana}_6$

\text{panamabanananas}_1$s
Using the Suffix Array to Find Matches

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

<table>
<thead>
<tr>
<th></th>
<th>( a_1 )bananas$panam_1</th>
<th>( a_2 )mabananas$pan_1</th>
<th>( a_3 )namabananas$pan_1</th>
<th>( a_4 )nanas$panamaban_1</th>
<th>( a_5 )nas$panamaban_2</th>
<th>( a_6 )s$panamabanana_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1)panamabananas_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( a_1 )bananas$panam_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( a_2 )mabananas$pan_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( a_3 )namabananas$pan_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( a_4 )nanas$panamaban_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>( a_5 )nas$panamaban_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>( a_6 )s$panamabanana_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>( b_1 )bananas$panam_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>( m_1 )ababananas$pana_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>( n_1 )amabantananas$pan_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>( n_2 )ananas$panamaban_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>( n_3 )as$panamabanana_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>( p_1 )anamabananas$pan_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>( s_1 )s$panamabanana_6</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.

- Thus, **ana** occurs at positions 1, 7, 9 of "panamabanananas $".

\[
\begin{array}{cccccccccccc}
13 & 1 & \text{panamabanananas} \\
5 & a & 1 & \text{bananas} & $ & \text{panama} \\
3 & a & 2 & \text{mabanananas} & $ & \text{pana} \\
1 & a & 3 & \text{mabananas} & $ & \text{pana} \\
7 & a & 4 & \text{na} & \text{nasan} & $ & \text{panamab} \\
9 & a & 5 & \text{na} & \text{nas} & $ & \text{panamaban} \\
11 & a & 6 & \text{as} & \text{panaman} & \text{n} & \text{a} \\
6 & b & 1 & \text{anan} & \text{as} & \text{panama} \\
4 & m & 1 & \text{abananas} & \text{pana} \\
2 & n & 1 & \text{amabanananas} & \text{pana} \\
8 & n & 2 & \text{ana} & \text{nas} & \text{panamaba} \\
10 & n & 3 & \text{as} & \text{panamaban} & \text{a} \\
0 & p & 1 & \text{anamabanananas} & $ \\
12 & s & 1 & \text{panamabananana} \\
\end{array}
\]
The Suffix Array: Memory Once Again

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

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

- Memory: $\sim 4 \times |Genome|$. 

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

- Memory: $\sim 4 \times |Genome|$.
Reducing Suffix Array Size

• We don’t want to have to store all of the suffix array; can we store only part of it? Show how checkpointing can be used to store 1/100 the suffix array.

A Return to Constants

• Explain that using a checkpointed array increases runtime by a constant factor, but in practice it is a worthwhile trade-off.
Returning to Our Original Problem

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

• Approx. Pattern Matching Problem:
  – **Input**: A string *Pattern*, a string *Genome*, and an integer *d*.
  – **Output**: All positions in *Genome* where *Pattern* appears as a substring with at most *d* mismatches.
Returning to Our Original Problem

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

• **Multiple Approx. Pattern Matching Problem:**
  – **Input:** A *collection* of strings *Patterns*, a string *Genome*, and an integer *d*.
  – **Output:** All positions in *Genome* where a string from *Patterns* appears as a substring with at most *d* mismatches.
Method 1: Seeding

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

  \[
  \begin{align*}
  \text{Pattern} & \quad \quad \text{acttggct} \\
  \text{Genome} & \quad \quad \ldots\text{ggcacacta}\text{ggctcc}\ldots
  \end{align*}
  \]
Method 1: Seeding

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

```
Pattern    actttggct
Genome    ...ggcactaggctcc...
```

• 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 \textbf{ana} in \textit{panamabanananas}

<table>
<thead>
<tr>
<th>String</th>
<th># Mismatches</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1\text{panamabanananas}_1$</td>
<td>1</td>
</tr>
<tr>
<td>$a_1\text{bananass}_1\text{panam}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$a_2\text{mabananas}_1\text{pan}_1$</td>
<td>1</td>
</tr>
<tr>
<td>$a_3\text{namabanananas}_1\text{p}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$a_4\text{nanas}_1\text{panamab}_1$</td>
<td>1</td>
</tr>
<tr>
<td>$a_5\text{nas}_1\text{panamabana}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$a_6\text{as}_1\text{panamabana}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$b_1\text{ananass}_1\text{panam}_1$</td>
<td>1</td>
</tr>
<tr>
<td>$m_1\text{bananass}_1\text{pana}_2$</td>
<td>0</td>
</tr>
<tr>
<td>$n_1\text{amabananass}_1\text{pa}_3$</td>
<td>1</td>
</tr>
<tr>
<td>$n_2\text{ananass}_1\text{panamaba}_4$</td>
<td>0</td>
</tr>
<tr>
<td>$n_3\text{as}_1\text{panamabana}_5$</td>
<td>0</td>
</tr>
<tr>
<td>$p_1\text{anamabananass}_1\text{s}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$s_1\text{panamabananan}_6$</td>
<td>0</td>
</tr>
</tbody>
</table>

Now we extend all strings with at most 1 mismatch.
Method 2: BWT Saves the Day Again

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

One string produces a second mismatch (the $), so we discard it.

```
$1 pan am ab an a na s
a1 b a na na s $ p a n a m
a2 m a b a na na s $ p a n
a3 n a m a b a na na s $ p1
a4 n a n a s $ p a n a m a b
a5 n a s $ p a n a m a b a n
a6 s $ p a n a m a b a n a
b1 a n a na s $ p a n a m a
m1 a b a na na s $ p a n a
n1 a m a b a na na s $ p a
n2 a n a s $ p a n a m a b a
n3 a s $ p a n a m a b a n a
p1 a n a m a b a na na s $1
s1 $ p a n a m a b a n a n a
```

# Mismatches

- $1 pan am ab an a na s: 1 mismatch
- a1 b a na na s $ p a n a m: 1 mismatch
- a2 m a b a na na s $ p a n: 0 mismatches
- a3 n a m a b a na na s $ p1: 0 mismatches
- a4 n a n a s $ p a n a m a b: 0 mismatches
- a5 n a s $ p a n a m a b a n: 0 mismatches
- a6 s $ p a n a m a b a n a: 2 mismatches
- b1 a n a na s $ p a n a m a: 1 mismatch
- m1 a b a na na s $ p a n a: 1 mismatch
- n1 a m a b a na na s $ p a: 0 mismatches
- n2 a n a s $ p a n a m a b a: 0 mismatches
- n3 a s $ p a n a m a b a n a: 0 mismatches
- p1 a n a m a b a na na s $1: 2 mismatches
- s1 $ p a n a m a b a n a n a: 0 mismatches
Method 2: BWT Saves the Day Again

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

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

\begin{align*}
$1_{\text{panamabanananas}} & \quad 1 \\
a_1_{\text{bananas}}$\text{panam}_1 & \quad 1 \\
a_2_{\text{abana}}$\text{pana}_1 & \quad 0 \\
a_3_{\text{namabana}}$\text{p}_1 & \quad 0 \\
a_4_{\text{anas}}$\text{panama}_1 & \quad 0 \\
a_5_{\text{nasa}}$\text{panama}_2 & \\
a_6_{\text{as}}$\text{panama}_3 & \\
b_1_{\text{anana}}$\text{panama}_1 & \\
b_2_{\text{abana}}$\text{pana}_2 & \\
m_1_{\text{anan}}$\text{panama}_3 & \\
m_2_{\text{as}}$\text{panama}_4 & \\
m_3_{\text{as}}$\text{panama}_5 & \\
p_1_{\text{anama}}$\text{panama}_1 & \\
s_1_{\text{panama}}$\text{panama}_6 & \\
s_6_{\text{panama}}$\text{panama}_6 & \\
\end{align*}

# Mismatches
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>Suffix Array</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| $p$           | $a$ | $m$ | $b$
| panamabanas   | bananas | panam | $s$
| a1bananas$panam1 | $a2mabananas$pan1 | $a3namabanananas$pan1 |
| a4anas$panamaban1 | $a5nas$panamaban2 | $a6s$panamaban2 |
| b1ananas$panama3 | $m1abanananas$pana2 | $n1amabanananas$pana3 |
| $n2anas$panamaba4 | $n3as$panamaban5 | $p1anamabanananas$pana6 |
Method 2: BWT Saves the Day Again

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

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

Suffix Array

\begin{align*}
\text{$s_1\text{panamabananas}_1$} & \quad 5 \\
\text{a_1\text{bananas}_1$panam_1$} & \quad 3 \\
\text{a_2\text{mabanananas}_1$pan_1$} & \quad 1 \\
\text{a_3\text{namabananas}_1$p_1$} & \quad 7 \\
\text{a_4\text{nana}$panamab_1$} & \quad 9 \\
\text{a_5\text{nas}_1$panamab_1$} & \\
\text{a_6\text{as}_1$panamab_2$} & \\
\text{b_1\text{anananas}_1$panama_1$} & \\
\text{m_1\text{bananas}_1$pana_2$} & \\
\text{n_1\text{amabananas}_1$pa_3$} & \\
\text{n_2\text{nas}_1$panamaba_4$} & \\
\text{n_3\text{as}_1$panamaba_5$} & \\
\text{p_1\text{anamabananas}_1$} & \\
\text{s_1$panamabananaa}_6$} & \\
\end{align*}
Burrows Wheeler Transform (BWT)

Example:

\[ S = \text{agcagcagact} \$

where the end of sequence pseudo-symbol, $\$, is less than all proper symbols.

\[ S \rightarrow \text{BWT}(S) \]

Sort the suffixes of $S$; the Burrows Wheeler transform [BW94] of $S$, $\text{BWT}(S)$, consists of the symbols before each sorted suffix in turn. Note that $\$ comes before $S[0]$.

Equivalently (with $\$), sort the rotations of $S$; $\text{BWT}(S)$ consists of the last symbol of each sorted rotation in turn.

<table>
<thead>
<tr>
<th>suffix#</th>
<th>BWT(S)</th>
<th>suffix/rotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>$\text{agcagcagact}$</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>act$\text{agcagcag}$</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>agact$\text{agcagc}$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>agcagact$\text{agc}$</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>agcagcagact$$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>cagact$\text{agcag}$</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>cagcact$\text{agcag}$</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>ct$\text{agcagcag}$</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>act$\text{agcagcag}$</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>agcagact$\text{agcag}$</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>agcagact$\text{agcag}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ch</th>
<th>$$</th>
<th>a</th>
<th>c</th>
<th>g</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank(ch)</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>
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.
Splice Sites
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** $\Sigma = \{ b_1, b_2, \ldots, b_M \}$
- **Set of states** $Q = \{ 1, \ldots, K \}$
- **Transition probabilities** between any two states
  
  $a_{ij} = \text{transition prob from state } i \text{ to state } j$

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

- **Start probabilities** $a_{0i}$

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

- **Emission probabilities** within each state

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

  $e_i(b_1) + \ldots + e_i(b_M) = 1, \quad \text{for all states } i = 1\ldots K$
At each time step \( t \),
the only thing that affects future states
is the current state \( \pi_t \)

\[
\begin{align*}
P(\pi_{t+1} = k \mid "\text{whatever happened so far}") &= \\
P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) &= \\
P(\pi_{t+1} = k \mid \pi_t) &= 
\end{align*}
\]
Given a sequence $x = x_1 \ldots x_N$, a parse of $x$ is a sequence of states $\pi = \pi_1, \ldots, \pi_N$. 
Likelihood of a parse

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)

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

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

$$P(x_1, \pi_1) =$$

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

$$a_{0\pi_1} a_{\pi_1\pi_2} \ldots a_{\pi_{N-1}\pi_N} e_{\pi_1}(x_1) \ldots e_{\pi_N}(x_N)$$
Example: the dishonest casino

Let the sequence of rolls be:

\[ x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4 \]

Then, what is the likelihood of

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

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

\[ \frac{1}{2} \times P(1 \mid \text{Fair}) \times P(\text{Fair} \mid \text{Fair}) \times P(2 \mid \text{Fair}) \times P(\text{Fair} \mid \text{Fair}) \times \ldots \times P(4 \mid \text{Fair}) = \]

\[ \frac{1}{2} \times (\frac{1}{6})^{10} \times (0.95)^{9} = 0.00000000521158647211 = 0.5 \times 10^{-9} \]
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

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

$\frac{1}{2} \times P(1 \mid \text{Loaded}) \times P(\text{Loaded, Loaded}) \times \ldots \times P(4 \mid \text{Loaded}) = \frac{1}{2} \times (1/10)^8 \times (1/2)^2 \times (0.95)^9 = .00000000078781176215 = 7.9 \times 10^{-10}$

Therefore, it is after all 6.59 times more likely that the die is fair all the way, than that it is loaded all the way.
Example: the dishonest casino

Let the sequence of rolls be:

\[ x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6 \]

Now, what is the likelihood \( \pi = F, F, \ldots, F \)?

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

What is the likelihood

\[ \pi = L, L, \ldots, L? \]

\[
\frac{1}{2} \times (1/10)^4 \times (1/2)^6 \times (0.95)^9 = 0.00000049238235134735 = 0.5 \times 10^{-7}
\]

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

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

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

\[ P[ x | M ]: \] The probability that sequence \( x \) was generated by the model

The model is: architecture (#states, etc)
+ parameters \( \theta = a_{ij}, e_i(.) \)

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

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

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

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

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

$\pi^* = \arg\max_{\pi} P[x, \pi]$

We can use dynamic programming!

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

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

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

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

What is $V_k(i+1)$?

From definition,

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

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

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

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

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

**Initialization:**
- \( V_0(0) = 1 \) (0 is the imaginary first position)
- \( V_k(0) = 0 \), for all \( k > 0 \)

**Iteration:**
- \( V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1) \)
- \( \text{Ptr}_j(i) = \arg\max_k a_{kj} V_k(i-1) \)

**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

left: Similar to “aligning” a set of states to a sequence,

Time: $O(K^2N)$; Space: $O(KN)$; right: comparison of valid directions in the alignment and decoding problem.
Viterbi Algorithm – a practical detail

Underflows are a significant problem

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

These numbers become extremely small – underflow

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

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

Let $x$ be a sequence with a portion of $\sim \frac{1}{6}$ 6's, followed by a portion of $\sim \frac{1}{2}$ 6's...

$x = 123456123456...12345\ 6626364656...1626364656$

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

\[
\begin{align*}
&\text{FFF}..........................F \\
&\text{LLL}.............................L
\end{align*}
\]

6 nucleotides “123456” parsed as F, contribute $0.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$

parsed as L, contribute $0.95^6 \times (1/2)^1 \times (1/10)^5 = 0.4 \times 10^{-5}$

“162636” parsed as F, contribute $0.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$

parsed as L, contribute $0.95^6 \times (1/2)^3 \times (1/10)^3 = 9.0 \times 10^{-5}$
Generating a sequence by the model

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

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

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. ... until emitting $x_n$
A couple of questions

Given a sequence $x$,

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

Example: the dishonest casino

Say $x = 123416231\textcolor{red}{62616364616}234161221341$

Most likely path: $\pi = \text{FF} \ldots \text{F}$

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

We will develop algorithms that allow us to compute:

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

\[ P(x_{i...j}) \] Probability of a substring of \( x \) given the model

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

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

We want to calculate

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

Sum over all possible ways of generating $x$:

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

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

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k) \quad \text{(the forward probability)}$$
The Forward Algorithm – derivation

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:**

- $f_0(0) = 1$
- $f_k(0) = 0$, 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:**
- \( V_0(0) = 1 \)
- \( V_k(0) = 0 \), for all \( k > 0 \)

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

**Termination:**
- \( P(x, \pi^*) = \max_k V_k(N) \)

**FORWARD**

**Initialization:**
- \( f_0(0) = 1 \)
- \( f_k(0) = 0 \), for all \( k > 0 \)

**Iteration:**
- \( f_i(i) = e_i(x_i) \sum_k f_k(i-1) a_{kl} \)

**Termination:**
- \( P(x) = \sum_k f_k(N) a_{k0} \)
Motivation for the Backward Algorithm

We want to compute

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

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

We start by computing

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

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

$$= \text{Forward, } f_k(i) \cdot \text{Backward, } b_k(i)$$
The Backward Algorithm – derivation

Define the backward probability:

\[ b_k(i) = P(x_{i+1}...x_N \mid \pi_i = k) \]

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

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

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

\[ = \sum_l e_l(x_{i+1}) a_{kl} b_l(i+1) \]
The Backward Algorithm

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

**Initialization:**

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

**Iteration:**

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

**Termination:**

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

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

Time: $O(K^2N)$
Space: $O(KN)$

Useful implementation technique to avoid underflows

Viterbi: sum of logs
Forward/Backward: rescaling at each position by multiplying by a constant
The GENSCAN Web Server at MIT

Identification of complete gene structures in genomic DNA

For information about Genscan, click here

Server update, November, 2009: We've been recently upgrading the GENSCAN webserver hardware, which resulted in some problems in the output of GENSCAN. We apologize for the inconvenience. These output errors were resolved.

This server provides access to the program Genscan for predicting the locations and exon-intron structures of genes in genomic sequences from a variety of organisms.

This server can accept sequences up to 1 million base pairs (1 Mbp) in length. If you have trouble with the web server or if you have a large number of sequences to process, request a local copy of the program (see instructions at the bottom of this page).

Organism: Vertebrate  Suboptimal exon cutoff (optional): 1.00
Sequence name (optional):
Print options: Predicted peptides only
Upload your DNA sequence file (upper or lower case, spaces/numbers ignored):
Or paste your DNA sequence here (upper or lower case, spaces/numbers ignored):
This server provides access to the program GenomeScan for predicting the locations and exon-intron structures of genes in genomic sequences from a variety of organisms.

GenomeScan incorporates protein homology information when predicting genes. This server allows you to input proteins suspected to be similar to regions of your DNA sequence. You can find such proteins by doing a BLASTX comparison of your sequence to all known proteins, or by running GENSCAN and then comparing the results to known proteins using BLASTP. Please input the proteins in FastA format; the file may contain multiple proteins so long as each is separated by a header on its own line. Files should contain less than one million bases.

If you would like to test the program, feel free to use this DNA testfile and the corresponding protein file.

**More information on GenomeScan:** [GenomeScan homepage](#)

You may also wish to use or read about the GENSCAN server, GenomeScan's predecessor.

---

**Run GenomeScan:**

Organism:  Vertebrate  
Sequence name (optional):  
Print options:  Predicted peptides only  
A eukaryotic gene

- This is the human p53 tumor suppressor gene on chromosome 17.
- Genscan is one of the most popular gene prediction algorithms.
This particular gene lies on the reverse strand.
An Intron

**GT**: signals start of intron
**AG**: signals end of intron

revcomp(CT)=AG

revcomp(AC)=GT
Modeling the 5’ splice site

- Most introns begin with the letters “GT.”
- We can add this signal to the model.
Modeling the 5’ splice site

- Most introns begin with the letters “GT.”
- We can add this signal to the model.
- Indeed, we can model each nucleotide with its own arrow.
Modeling the 5’ splice site

- Like most biological phenomenon, the splice site signal admits exceptions.
- The resulting model of the 5’ splice site is a length-2 PSSM.
Real splice sites

- Real splice sites show some conservation at positions beyond the first two.
- We can add additional arrows to model these states.
Modeling the 5’ splice site
Length distributions of human introns and initial, internal and terminal exons

(a) Introns

(b) Initial exons

(c) Internal exons

(d) Terminal exons
GenScan

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

- Model both strands at once
- Each state may output a string of symbols (according to some probability distribution).
- Explicit intron/exon length modeling
- Advanced splice site modeling
- Complete intron/exon annotation for sequence
- Able to predict multiple genes and partial/whole genes
- Parameters learned from annotated genes
- Separate parameter training for different CpG content groups (< 43%, 43-51%, 51-57%, >57% CG content)

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
Hidden Markov models

How to identify protein structural parts?

Membrane proteins that are important for cell import/export. We would like to predict the position in the amino acids with respect to the membrane. The prediction of gene parts and the membrane protein topology (i.e. which parts are outside, inside and buried in the membrane) will require to train the model with a dataset of experimentally determined genes / transmembrane helices and to validate the model with another dataset. The figure below describes a 7 helix membrane protein forming a sort of a cylinder (porus) across the cell membrane.
Membrane proteins
Cystic fibrosis

The gene affected by CF controls the movement of salt and water in and out of cells. People with cystic fibrosis experience a build-up of thick sticky mucus in the lungs, digestive system and other organs, causing a wide range of challenging symptoms affecting the entire body.
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.
Assessing performance: Sensitivity and Specificity

• Testing of predictions is performed on sequences where the gene structure is known

• **Sensitivity** is the fraction of known genes (or bases or exons) correctly predicted: $Sn = \frac{N_{\text{True Positives}}}{N_{\text{All True}}}$
  – “Am I finding the things that I’m supposed to find?”

• **Specificity** is the fraction of predicted genes (or bases or exons) that correspond to true genes: $Sp = \frac{N_{\text{True Positives}}}{N_{\text{All Positives}}}$
  – “What fraction of my predictions are true?”

• In general, increasing one decreases the other
Validation

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

- Testing of predictions is performed on sequences where the gene structure is known.
- **Sensitivity** is the fraction of known genes (or bases or exons) correctly predicted: \( Sn = \frac{N_{\text{True Positives}}}{N_{\text{All True}}} \)
  - “Am I finding the things that I’m supposed to find?”
- **Specificity** is the fraction of predicted genes (or bases or exons) that correspond to true genes: \( Sp = \frac{N_{\text{True Positives}}}{N_{\text{All Positives}}} \)
  - “What fraction of my predictions are true?”
- In general, increases

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

\[
AN = TN + FP; \ AP = TP + FN; \\
PP = TP + FP; \ PN = TN + FN
\]
Graphic View of Specificity and Sensitivity

Correlation Coefficient

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

\[AN = TN + FP; AP = TP + FN;\]
\[PP = TP + FP; PN = TN + FN\]
Specificity/Sensitivity Tradeoffs

• Ideal Distribution of Scores

• More Realistically...
TMHMM Server v. 2.0
Prediction of transmembrane helices in proteins

SUBMISSION

Submission of a local file in FASTA format (HTML 3.0 or higher)
- Sfoglia... Nessun file selezionato.

OR by pasting sequence(s) in FASTA format:
>AAA39861.1 opsin [Mus musculus]
MNGTEGPNTVFPSNVTGCRPRFEQPYVLAEFWQFSMLAAYMFLLIVLGFINFTLYVTQHKLRT
PLNYILLNLAVIDLFMVFGF TTLTLTLHGYVFVFGTGCDNFVFATLGGE1ALWLSLVLAIEYVVVC
KMSNFRFCEHAIRMVVFTMINRALACAPPILVGC5RYIPCGMQCCCDYLTKEVNVNENPVYIPFV
HFPIMIVIFPCQYQQLVFTKVEAAQQQBESATTQKAKEVTRTHVIMHIVYPFLICWLPEASVAFYIITHQG
SNFGPIFMTLFAAFFKSSISIYNPIYIIMLNQFRCNMLTTLCGGKNPLGDDASATASSTETSVAPA

Output format:
- Extensive, with graphics
- Extensive, no graphics
- One line per protein

Other options:
- Use old model (version 1)

Submit  Clear

Restrictions:
At most 10,000 sequences and 4,000,000 amino acids per submission; each sequence not more than 8,000 amino acids.

Confidentiality:
The sequences are kept confidential and will be deleted after processing.
Model architecture of TMHMM

TMHMM: uses cyclic model with 7 states for
- TM helix core
- TM helix caps on the N- and C-terminal side
- non-membrane region on the cytoplasmic side
- 2 non-membrane regions on the non-cytoplasmic side (for short and long loops to account for different membrane insertion mechanism)
- a globular domain state in the middle of each non-membrane region
# Sequence Length: 274
# Sequence Number of predicted TMHs: 7
# Sequence Exp number of AAs in TMHs: 153.74631
# Sequence Exp number, first 60 AAs: 22.08833
# Sequence Total prob of N-in: 0.04171

# Sequence POSSIBLE N-term signal sequence

<table>
<thead>
<tr>
<th>Sequence</th>
<th>TMHMM2.0</th>
<th>outside</th>
<th>1</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>Tmhelix</td>
<td>27</td>
<td>49</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>inside</td>
<td>50</td>
<td>61</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>Tmhelix</td>
<td>62</td>
<td>84</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>outside</td>
<td>85</td>
<td>103</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>Tmhelix</td>
<td>104</td>
<td>126</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>inside</td>
<td>127</td>
<td>130</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>Tmhelix</td>
<td>131</td>
<td>153</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>outside</td>
<td>154</td>
<td>157</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>Tmhelix</td>
<td>155</td>
<td>150</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>inside</td>
<td>181</td>
<td>200</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>Tmhelix</td>
<td>201</td>
<td>223</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>outside</td>
<td>224</td>
<td>227</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>Tmhelix</td>
<td>228</td>
<td>250</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>inside</td>
<td>251</td>
<td>274</td>
</tr>
</tbody>
</table>

http://www.cbs.dtu.dk/services/TMHMM-2.0/
Adleman's first DNA computation solved a traveling salesman problem of 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. Pros: 1 gram of DNA can hold about $1 \times 10^{14}$ MB of data. A test tube of DNA can contain trillions of strands. Each operation on a test tube of DNA is carried out on all strands in the tube in parallel; Adleman figured his computer was running $2 \times 10^{19}$ operations per joule. Adleman’s process to solve the traveling salesman problem for 200 cities would require an amount of DNA that weighed more than the Earth.

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  GTGACATGTAAATAATCGCA
City 4->5  CGCCGGATACGGCGCTATCA
City 5->6  GATCTAAAGGTATGCATACG
Etc.

L. Adelman, *Scientific American*, pp. 54-61 (Aug 1998);
DNA for computing

figures from Martyn Amos

(a) cities

<table>
<thead>
<tr>
<th>Vertex 1</th>
<th>Vertex 2</th>
<th>Vertex 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertex 4</th>
<th>Vertex 5</th>
<th>Vertex 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertex 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

(b) selection for length and initial/end points

V1 to V2, V1 to V4, V1 to V7
V2 to V3, V2 to V4
V3 to V2, V3 to V4
V4 to V3, V4 to V5, V4 to V7
V5 to V2, V5 to V6
V6 to V2, V6 to V7
‘travelling salesman’ problem

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.

Since the enzymes (enzymes are proteins catalyzing a reaction) work on many DNA molecules at once, the selection process is massively parallel. Specifically, the method based on Adleman’s experiment would be as follows:

• Generate all possible routes.
• Select itineraries that start with the proper city and end with the final city.
• Select itineraries with the correct number of cities.
• Select itineraries that contain each city only once.
• All of the above steps can be accomplished with standard molecular biology techniques.
Discover magazine published an article in comic strip format about Leonard Adleman's DNA computation.
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. The basic principle behind Gel Electrophoresis is to force DNA through a gel matrix by using an electric field. DNA is a negatively charged molecule under most conditions, so if placed in an electric field it will be attracted to the positive potential.
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 between these strands, 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. Since 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.
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 (=bonds) with its complimentary sequence (complementary strands of DNA bind each other).

For example, you can encode the routes between cities by encoding the compliment 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.
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.
**Itineraries Selection: Have a Complete Set of Cities**

DNA containing a specific sequence can be purified from a sample of mixed DNA by a technique called affinity purification, as shown below. This is accomplished by attaching the complement 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 experiment solved a seven city problem, but there are two major shortcomings preventing a large scaling up of his computation.

• The complexity of the traveling salesman problem simply doesn’t disappear when applying a different method of solution - it still increases exponentially.

• For Adleman’s method, what scales exponentially is not the computing time, but rather the amount of DNA. Unfortunately this places some hard restrictions on the number of cities that can be solved; after the Adleman article was published, more than a few people have pointed out that using his method to solve a 200 city problem would take an amount of DNA that weighed more than the earth.
Adleman’s pros & cons

Pros: 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; Adleman figured his computer was running $2 \times 10^{19}$ operations per joule.
Adleman’s process to solve the traveling salesman problem for 200 cities would require an amount of DNA that weighed more than the Earth.
Speed: 500-5000 base pairs a second.
Design of random access primers and coding algorithm. (a, 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.
References


DNA as information storage

The work, carried out by George Church and Sri Kosuri, basically treats DNA as just another digital storage device. Instead of binary data being encoded as magnetic regions on a hard drive platter, strands of DNA 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.

**STORAGE LIMITS**
Estimates based on bacterial genetics suggest that digital DNA could one day rival or exceed today’s storage technology.

<table>
<thead>
<tr>
<th></th>
<th>Hard disk</th>
<th>Flash memory</th>
<th>Bacterial DNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-write speed (μs per bit)</td>
<td>~3,000–5,000</td>
<td>~100</td>
<td>&lt;100</td>
</tr>
<tr>
<td>Data retention (years)</td>
<td>&gt;10</td>
<td>&gt;10</td>
<td>&gt;100</td>
</tr>
<tr>
<td>Power usage (watts per gigabyte)</td>
<td>~0.04</td>
<td>~0.01–0.04</td>
<td>&lt;10⁻¹⁰</td>
</tr>
<tr>
<td>Data density (bits per cm²)</td>
<td>10¹³</td>
<td>~10¹⁶</td>
<td>~10¹⁹</td>
</tr>
</tbody>
</table>

more at the end of the course
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, we encode and store 35 distinct files (over 200 MB of data), in more than 13 million DNA oligonucleotides, and show that we can recover each file individually and with no errors, using a random access approach. We design and validate a large library of primers that enable individual recovery of all files stored within the DNA. We also develop an algorithm that greatly reduces the sequencing read coverage required for error-free decoding by maximizing information from all sequence reads. These advances demonstrate a viable, large-scale system for DNA data storage and retrieval.
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.
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.
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.
Example files encoded within the 200 MB of data.

A comparison to research achievements shows that our coding scheme has similar logical redundancy, but requires lower sequencing coverage to recover files.
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.
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.
1 Bioinformatics (PL)

(a) What are the usage and the limitations of the Bootstrap technique in phylogeny? [6 marks]

\textit{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]

\textit{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. [7 marks]

(b) Discuss with one example how to score a multiple sequence alignment. [5 marks]
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]
1. Bioinformatics (PL)

(a) What are the usage and the limitations of the Bootstrap technique in phylogeny? [6 marks]

(b) We often use Hidden Markov Models (HMM) to predict a pattern (for instance the exons). How can you compute the number of True Positives, True Negatives, False Positives and False Negatives and use them to evaluate your HMM? [6 marks]

(c) How can you evaluate the results obtained (number of clusters and their relative position) using the K means algorithm for clustering? [5 marks]
(b) We often use Hidden Markov Models (HMM) to predict a pattern (for instance the exons). How can you compute the number of True Positives, True Negatives, False Positives and False Negatives and use them to evaluate your HMM?

[6 marks]

Answer:

(i) be predicted to occur: Predicted Positive (PP)
(ii) be predicted not to occur: Predicted Negative (PN)
(iii) actually occur: Actual Positive (AP)
(iv) actually not occur: Actual Negative (AN)
(v) True Positive \( TP = PP \cap AP \)
(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 \( Sp = TP/(TP + FP) \)