Bioinformatics

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Computer Science Tripos Part II

Pietro Lio
BioInformatics  2018-2019

12 lectures -- Pietro Lio’, pl219 (given the interdisciplinarity of the course the lecturer is happy to meet – FC20- the students after the lectures or in the afternoons to clarify lecture content and examinable material)

Bioinformatics is focused on developing algorithms to be used in biological and medical researches. Geneticists, molecular and cell biologists generate massive amounts of information that can only be efficiently analyzed with computers.

Computer science could provide the abstraction needed for consolidating knowledge of biomolecular systems.

Both DNA sequence and protein structure research have adopted good abstractions: ‘DNA-as-string’ (a mathematical string is a finite sequence of symbols) and ‘protein-as-three-dimensional-labelled-graph’, respectively; the interaction DNA-protein produces other graphs.

For sake of space, the sources of many figures are acknowledged during the lectures
BioInformatics algorithms

- Multidisciplinarity
- Computer scientists could help biologists
- Biology could inspire computer science
- Various algorithms covering different topics
- No biology in the exam questions
- You need to know only the biology in the slides to understand the reason of the algorithms
- Partly based on book: Compeau and Pevzner Bioinformatics algorithms (chapter 3,5,7-10 chapter); also Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison
- Color slides from the course website
What is Bioinformatics

- Biology
- Machine Learning
- Algorithms

Great Data
Cool Questions
Powerful tools

Drug Discovery Pipeline

Accelerated by Bioinformatics
In the development of new field, techniques, technologies have impact!

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

— Sydney Brenner —

Sydney Brenner, 2002 Nobel Prize in Physiology or Medicine
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC139404/
Bioinformatics: what is for?

- helping doctors and understanding biology:
- Computing with DNA and other biological molecules
- Using DNA to storage information

most of Bioinformatics focuses on genes, mRNA (i.e. gene activity) and proteins.

**Central Dogma**

- Transcription
- Translation

**Gene**

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

Batch 1 now sold out
Please contact sales for batch 2 waiting list

Bento Lab is a DNA laboratory, suitable for beginners to professionals. It combines the essential tools for molecular biology. And with an A4 footprint that fits into any laptop-sized bag, Bento Lab can travel wherever your science goes.
DNA SEQUENCES AS STRINGS

DNA: 4-letter alphabet, A (adenine), T (thymine), C (cytosine) and G (guanine). In the double helix A pairs with T, C with G ; RNA: same as DNA but T -> U (uracil)
3 letters (triplet – a codon) code for one amino acid in a protein.

5′-CCTGAGCCAACACTATTGATGAA-3′
3′-GGACTCGGTTGATAAUCTACTTT-5′

RNA

CCUGAGCCAACUAAUGAUGA

Genome: an organism’s genetic material; human genome= 46 strings (chromosomes) with overall length 3 x 10⁹ base.

If you stretch the DNA in one cell all the way out, it would be about 2m long and all the DNA in all your cells put together would be about twice the diameter of the Solar System.
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>Variant calling, ~2 trillion central processing unit (CPU) hours</td>
</tr>
<tr>
<td></td>
<td>Massive volumes</td>
<td></td>
<td></td>
<td>All-pairs genome alignments, ~10,000 trillion CPU hours</td>
</tr>
<tr>
<td>Distribution</td>
<td>Dedicated lines from antennae to server (600 TB/s)</td>
<td>Small units of distribution</td>
<td>Major component of modern user’s bandwidth (10 MB/s)</td>
<td>Many small (10 MB/s) and fewer massive (10 TB/s) data movement</td>
</tr>
</tbody>
</table>

doi:10.1371/journal.pbio.1002195.t001
Proteins as 3D labelled graphs

units are the 20 amino acids A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y.

5′-CCTGAGCCAACTATTTGATGAA-3′
3′-GGACTCGGTTGATAACTACTT-5′

CCUGAGCCACUACUUUGAUUGAA

PEPTIDE

data repository  https://www.rcsb.org/
DNA: 4-letter alphabet, A (adenosine), T (thymine), C (cytosine) and G (guanine). In the double helix A pairs with T, C with G.
Gene: hereditary information located on the chromosomes and consisting of DNA.
RNA: same as DNA but T -> U (uracil)
3 letters (triplet – a codon) code for one amino acid in a protein.
Genome: an organism’s genetic material
Healthy Individual

sequences in Fasta format

>gi|28302128|ref|NM_000518.4| Homo sapiens hemoglobin, beta (HBB), mRNA
ACATTTGCTTCTGACAACAACCTGTGTTCACTAGCAACCTCAACAGACACCATGGTGCATCTGACTCCTGA
GGAGAAGTCTGCGCTCCTGCTGGGGCAAGGAGTGAATGTTGTGGTGGAGGCCCTGGGC
AGGCTGCTGGTGTCTACCTTGGACACCAGGGTTCTTTTGAGTCTCTTGAGGTGTCTCCACTCTGATG
CTGGTATGGGCAACCTAAAGGTGAAGGCTGACATGGGCAAGAAGTGTGCTCAGGTGCCCTTATGAGTGGCTG
TCACCTGGACACCTCAAACGTCGCTTGGCCACACTGAGTGCACAGTGACAGTGTGACCAGCAGTGGAT
CTGAGAACTTCAGGCTCTCTGGGCAACGTGCTGGTCTGTGCTGTGCTGGCCATCACTTTTGGAAGAATTCA
CCCCACCATGCGAGGCTGCTTACTCATGAAAGTGGTTGCTGGGCTGGTGGCTAATGCCCTGGCCCACAAGATATCA
CTAAGCTCGCTTTTCCTTGCGCTTCAATTCTATATTTAAGGTGTCTTCTTTCCTCCTAAGTCCAACACTAAGACT
GGGGATATTATGAGGCGTGTGGCTCTGCATCTGATTCTGCCTAATAAAACATTTATTTTCATTGC

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

MVHLTPEKSAVTALWKGKVNDEVGGGALGRLLVVYPWTQRFSESFGDLSTPDAMGPKVKAHGKKVLG
AFSDGLAHLDNLKGTFAFLSHELCDKLHVDPNFLLLGNVLVCLAHFFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH
Individual with Sickle Cell Anemia

>gi|28302128|ref|NM_000518.4| Homo sapiens hemoglobin, beta (HBB), mRNA
ACATTTGCTTTCTGACACAAACTGTGTTCACTAGCAACCTCAAACAGACACCATG
GTGCATCTGACTCCTGA
GGAAGTCTGCCGTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGC
AGGCTGCTGGTGTCTACCCTTGACACCAGAGGTTCTTGTAGTCCTCTTTGGGGATCTGTCCACTCCTGATG
CTGTATGGGCAACCCCTAAGGGTGAAGGTCTCATGGCAAGAAAAGTGTCCTGCGCTTTTAGTGATGGCCCTGGGC
TCACCTGGACAACCTCAAGGGCACCTTTTGCCACACTGAGTGAGCTGACCGCTGACAGCTGAGAT
CCTGAGAAGCCTCTGCGCCACCTGGTCGCTGTCCTCTGTGCTGGCCTCCACTCTTTGGGCAAGAAATTCA
CCCCACCAGTGCAGGCTGCTATCACAGAAATGGGTGGGCTGCTGCTGCTGGCTATGCTGCCCCACAAAGTATCA
CTAAGCTCGCTTTCTTGCTGCTCAAATATCTATTAAGGTTCCTCTTTGGCTCCCTGCTAAGTGCCACTACTAACT
GGGGGATATTATGAAAGGGCCTTGAGCATCTGGATTCTGCCTAAATAAAAACATTTATTTTCATTGC

>gi|4504349|ref|NP_000509.1| beta globin [Homo sapiens]
MVHLTPVEKSAVTAIWLKGKPNVDVGEAGRLLLVVPWTQRFFESFGDLSTPDAVMGNPKVAHGKKVLG
AFSDGIAHLNLKGTATLSELHCDKLHVDPENFRLONGVLVCVALHFGKEFTPPVQAAYQKVAGVAN
ALAHKYH
A) Each triplet of bases codes for one amino acid.
B) Genes differ for the amount of messenger RNA and protein molecules they produce (variable among cells type, position and time regulation).
C) Potentially the DNA strands could code for 6 proteins.
Gene interactions as graphs

Genes are activated or repressed by regulatory proteins which bind to gene flanking sequences (promoter) and are coded by the same or other genes.
The Cell is a Computer in Soup

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).
Scaling electro and bio devices

**Bacterium**

\[ \lambda = 0.25 \text{ micron in Pentium II} \]

**Human chromosome.**
Network level
Cells versus Computers (simple)

- 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
<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>
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.
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 = 1, G = 0, 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.

more at the end of the course
DNA for computing:
Represent Each City By A DNA Strand of 20 Bases

City1  ATGCTCAGCCTACTATAGCGA
City2  TGCAGATGCTACTAGCTATAT
City3  GCATAGTGTACACTGTACAA
City4  TTATTAGCTGCGGCGCTATG
City5  CCGCGATAGTCTAGATTTCC
Etc.

Represent Each Air Route By Mixed Complementary Strands

City 1→2  TGATATCGCTACGCTACATG
City 2→3  ATCGTATATACGCTATACCAT
City 3→4  GTGACATGTATTAAATCGCA
City 4→5  CGCCGGATACGGCGCTATCA
City 5→6  GATCTAAAGGTATGCATACG
Etc.

L. Adelman, *Scientific American*, pp. 54-61 (Aug 1998);
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 (more at the end of the course)

Adleman’s ingredients encoding paths - schematic representation of DNA strings

figures from Martyn Amos DNA Computing
BioPerl

From Wikipedia, the free encyclopedia

BioPerl [1] is a collection of Perl modules that facilitate the development of Perl scripts for bioinformatics applications. It has played an integral role in the bioinformatics community over the years.

Bioconductor

Introduction

Bioconductor is an open-source project that provides a platform for the development and distribution of bioinformatics software. It is designed to be used with the R programming language, and it includes a wide range of tools for analyzing and visualizing genomic and proteomic data.

BioPython

Introduction

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

SCABIO (Scala algorithms for bioinformatics)

BioPHP PHP for Bioinformatics

The aim of this site is to share knowledge by using a Wiki-like service: classes, functions, and minitools can be edited by registered users. More info here.

GenePHP - Download

- Sequence Class
- Sequence Database Class
- Sequence Aligner Class
- Restriction Enzyme Class
- Miscellaneous Class

Minitools - Download all minitools (.zip)

- Sequence manipulation and data loads
- PCR Amplifier
- microarray

Bip: Biomedical Logic Programming

Bip is a collection of logic programming modules intended primarily for bioinformatics and modules which may be of more general interest. Bip is intended to be both an application written in SWI-Prolog, a fast, robust and scalable implementation of ISO Prolog.

Bip is Free Software, available, licensed under the LGPL.
General references for chapter 1

Molecular biology for Computer Scientists
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/
General references for lecture notes
Structure of the course

Alignment 1

Hidden Markov Models 6

Phylogeny 2

Genome sequencing 3

Clustering 4

Genome Assembly 5

Phylogeny to improve multi sequence alignment

Hierarchical clustering

identify all functions

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 trees
Aligning DNA and Protein Sequences

• 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
- Penalizing Insertions and Deletions in Sequence Alignment
- Space-Efficient Sequence Alignment
- Nussinov folding algorithm
Alignment

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?
What Is the Sequence Alignment?

**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 \( w \):

\[
\begin{align*}
    v & : \quad \text{A T G T T A T} \\
    w & : \quad \text{A T C G T A C}
\end{align*}
\]

\( m = 7 \)

\( n = 7 \)

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

Letters of \( v \)

\[
\begin{array}{cccccccc}
    & A & T & -- & G & T & T & A & T & -- \\
\end{array}
\]

Letters of \( w \)

\[
\begin{array}{cccccccc}
    A & T & C & G & T & -- & A & -- & C \\
\end{array}
\]

4 matches 2 insertions 2 deletions
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

\[ v = v_1 v_2 \ldots v_m \quad \text{and} \quad w = w_1 w_2 \ldots w_n \]

• The LCS of \( v \) and \( w \) is a sequence of positions in

\[ v: 1 \leq i_1 < i_2 < \ldots < i_t \leq m \]

and a sequence of positions in

\[ w: 1 \leq j_1 < j_2 < \ldots < j_t \leq n \]

such that \( i_t \)-th letter of \( v \) equals to \( j_t \)-th letter of \( w \) and \( t \) is maximal
Every common subsequence is a path in 2-D grid
The Edit distance between two strings is the minimum number of operations (insertions, deletions, and substitutions) to transform one string into the other.

Hamming distance always compares the \( i \)-th letter of \( v \) with the \( i \)-th letter of \( w \):

\[
\begin{align*}
v &= \text{ATATATA} \\
w &= \text{TATATATA}
\end{align*}
\]

Hamming distance: \( d(v, w) = 8 \)

Computing Hamming distance is a trivial task.

Edit distance may compare the \( i \)-th letter of \( v \) with the \( j \)-th letter of \( w \):

\[
\begin{align*}
v &= \text{-ATATATA} \\
w &= \text{TATATATA-}
\end{align*}
\]

Edit distance: \( d(v, w) = 2 \)

Computing edit distance is a non-trivial task.
TGCATAT $\rightarrow$ ATCCGAT in 4 steps

TGCATAT $\rightarrow$ (insert A at front)
ATGCATA$^{\text{T}}$ $\rightarrow$ (delete 6$^{\text{th}}$ T)
ATGCA$^{\text{TA}}$ $\rightarrow$ (substitute G for 5$^{\text{th}}$ A)
ATGCG$^{\text{TA}}$ $\rightarrow$ (substitute C for 3$^{\text{rd}}$ G)
ATCCGAT (Done)
Alignment as a Path in the Edit Graph

Old Alignment

0122345677

v = AT_GTTAT_

w = ATCGT_A_C

0123455667

New Alignment

0122345677

v = AT_GTTAT_

w = ATCG_TA_C

0123445667

Two similar alignments; the score is 5 for both the alignment paths.
LCS Problem as - Edit Graph

Every path is a common subsequence.

Every diagonal edge adds an extra element to common subsequence

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

Let $v_i = \text{prefix of } v \text{ of length } i: \ v_1 \ldots v_i$

and $w_j = \text{prefix of } w \text{ of length } j: \ w_1 \ldots w_j$

The length of $LCS(v_i, w_j)$ is computed by:

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

Every Path in the Grid Corresponds to an Alignment
LCS Algorithm

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

\[
s_{i, j} = \max \begin{cases} 
  s_{i-1, j} - \sigma \\
  s_{i, j-1} - \sigma \\
  s_{i-1, j-1} + 1, \text{ if } v_i = w_j \\
  s_{i-1, j-1} - \mu, \text{ if } v_i \neq w_j 
\end{cases}
\]
The 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 very 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.
Notice three possible cases:

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

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

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

Towards an algorithm to align biological sequences

\[ F(i,j) = \begin{cases} 
  F(i-1, j-1) + m, & \text{if } x_i = y_j \\
  F(i-1, j) - d, & \text{if not}
\end{cases} \]
• 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,

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

Where

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

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

**Global Alignment**

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

**Local Alignment**—better alignment to find conserved segment

```
tccCAGTTATGTCAGgggacacgagcatgcagagac
| | | | | | | | | | | | | | | | | | | | | | | | |
|aattgccccgctcgtttttcagCAGTTATGTCAGatc|
```
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**
   a. For each $j = 1 \ldots M$
      For each $i = 1 \ldots N$
      $F(i, j) = \max$
      \[
      \begin{cases}
      F(i-1, j) - d \quad &\text{[case 1]} \\
      F(i, j-1) - d \quad &\text{[case 2]} \\
      F(i-1, j-1) + s(x_i, y_j) \quad &\text{[case 3]}
      \end{cases}
      \]
      $\text{Ptr}(i, j) =$
      \[
      \begin{cases}
      \text{UP, if [case 1]} \\
      \text{LEFT if [case 2]} \\
      \text{DIAG if [case 3]}
      \end{cases}
      \]
   d is a penalty

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:

```
CTATCACCTGACCTCCAGGCCTGCCCCTTCCGGC
GCGAGTTCTATCTATGCACGACC--GGTGC
```

Changes:

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

2. **Termination**

   $F_{OPT} = \max_{i} F(i, N) \max_{j} F(M, j)$
Can we use a similar algorithm to align entire genomes?
Local Alignment = Global Alignment in a Subrectangle
Local Alignment Problem

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

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

- **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 $w$. 
The local alignment: Smith-Waterman algorithm

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

\[ x = \text{aaaacc} \text{cccgggg} \]
\[ y = \text{cccg} \text{ggg} \text{aaccaacc} \]

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

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

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

The local alignment: Smith-Waterman algorithm
Which Alignment is Better?

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

GCC-C-AGT--TATGT-CAGGGGCGACG--A-GCATGCAGA-
GCCGCC-GTCTG-T-TTCAG-----CA-GTTATG--T-CAGAT

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

----G-----C------C----CAGTTATGTCAGGGGCA
GCCGCCGTCGTTTTTCAGCAGTTATGTCAG-----A-----T-----
Which Alignment is Better?

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

GCC-C-AGT---TATGT-CAGGGGGCAGC---AGCATGCAGA-
GCCGCC-GTCTTGT-T-TTCAG-----CA-GTTATG--T-CAGAT

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

---G-----C------C---CAGTTATGTTCAGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTTCAG-----A------T------
local alignment
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:
Mismatches and Indel Penalties

\[ \#\text{matches} - \mu \cdot \#\text{mismatches} - \sigma \cdot \#\text{indels} \]

\[
\begin{align*}
\text{A} & - T & - G & T & T & A & T & A \\
\text{A} & - T & C & G & T & - & C & - & C
\end{align*}
\]

\[+1+1-2+1+1-2-3-2-3=-7\]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>+1</td>
<td>-\mu</td>
<td>-\mu</td>
<td>-\mu</td>
<td>-\sigma</td>
</tr>
<tr>
<td>C</td>
<td>-\mu</td>
<td>+1</td>
<td>-\mu</td>
<td>-\mu</td>
<td>-\sigma</td>
</tr>
<tr>
<td>G</td>
<td>-\mu</td>
<td>-\mu</td>
<td>+1</td>
<td>-\mu</td>
<td>-\sigma</td>
</tr>
<tr>
<td>T</td>
<td>-\mu</td>
<td>-\mu</td>
<td>-\mu</td>
<td>+1</td>
<td>-\sigma</td>
</tr>
<tr>
<td>-</td>
<td>-\sigma</td>
<td>-\sigma</td>
<td>-\sigma</td>
<td>-\sigma</td>
<td>-\sigma</td>
</tr>
</tbody>
</table>

Scoring matrix

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

Even more general scoring matrix
### Scoring Matrices for Amino Acid Sequences

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

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?

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

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

$$upper_{i,j} = \max \{ upper_{i,j-1} - \varepsilon, middle_{i,j-1} - \sigma \}$$
Modelling Affine Gap Penalties by Long Edges
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:**

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

**Termination**: same

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

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

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

\begin{align*}
| & | \\
\text{gap} & \text{gap} \\
\text{open} & \text{extend}
\end{align*}

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

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

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

**Initialization:**

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

**Iteration:**

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

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

**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:** $\# \text{gaps}(x, y) < k(N)$ \hspace{1cm} (say $N>M$)

$x_i$ implies $|i-j| < k(N)$

$y_j$

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

\[
\begin{array}{|c|c|}
\hline
F[i, i+k/2] & \text{Out of range} \\
\hline
F[i+1, i+k/2] & F[i+1, i+k/2 +1] \\
\hline
\end{array}
\]

Note that for diagonals, $i-j =$ constant.
Banded Dynamic Programming

**Initialization:**
F(i,0), F(0,j) undefined for i, j > k

**Iteration:**
For i = 1...M  
For j = max(1, i − k)...min(N, i+k)

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

**Termination:**
same

Easy to extend to the affine gap case
<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>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Find match score, mismatch, gap penalty

Scores:
- A: 0
- C: -1

Penalties:
- Find match score, mismatch, gap penalty
<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>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>T</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

ACGCTG

---

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

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

ACG

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

The sequence ACG - CA is highlighted.
<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>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>2</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>3</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>-4</td>
<td>-1</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>-5</td>
<td>-2</td>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

The table represents a scoring matrix for aligning DNA sequences, with arrows indicating the cost of mutating from one nucleotide to another.
<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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>1</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>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</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>
<tr>
<td>C</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td></td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The sequence is: "ACGCTG CATG−T−"
Local Alignment Example

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

Find match score, mismatch, gap penalty
Local Alignment Example

$y = TAATA$

$x = TACTAA$

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

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

\[
\begin{array}{cccccccc}
   & \text{T} & \text{A} & \text{C} & \text{T} & \text{A} & \text{A} \\
\hline
   y & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
   0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
   T & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
   1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
   A & 2 & 0 & 0 & 2 & 0 & 2 & 1 \\
   2 & 0 & 0 & 2 & 0 & 2 & 1 & 1 \\
   A & 3 & 0 & 0 & 1 & 1 & 1 & 3 \\
   3 & 0 & 0 & 1 & 1 & 1 & 3 & 3 \\
   T & 4 & 0 & 0 & 0 & 0 & 2 & 1 \\
   4 & 0 & 0 & 0 & 0 & 2 & 1 & 1 \\
   A & 5 & 0 & 0 & 1 & 0 & 0 & 3 \\
   5 & 0 & 0 & 1 & 0 & 0 & 3 & 3 \\
\end{array}
\]
Local Alignment Example

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

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<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>T1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>T3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>T5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
Computing Alignment Score with Linear Memory

Alignment Score

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

- $\text{prefix}(i)$ 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

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

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

store \textit{suffix}(i) column

\begin{align*}
0 & \quad m/2 & \quad m
\end{align*}
Length(i) = Prefix(i) + Suffix(i)

- Add $\text{prefix}(i)$ and $\text{suffix}(i)$ to compute $\text{length}(i)$:
  - $\text{length}(i) = \text{prefix}(i) + \text{suffix}(i)$
- You now have a middle vertex of the maximum path $(i, m/2)$ as maximum of $\text{length}(i)$
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 $(\text{mid}, m/2)$ 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)
Middle Node of the Alignment

Middle node
(a node where an optimal alignment path crosses the middle column)
Divide and Conquer Approach to Sequence Alignment

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

\textbf{AlignmentPath}(source, sink)

find \textit{MiddleNode}

\textbf{AlignmentPath}(source, \textit{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?

\[ \text{4-path that visits the node (4,middle)} \]
\[ \text{In the middle column} \]

\[ \text{i-path – a longest path among paths that visit the i-th node in the middle column} \]
Can We Find The Lengths of All $i$-paths?

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

$$\text{length}(0)=2$$

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

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

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

*fromSource*(i)

*toSink*(i)

105
How Much Time Did It Take to Find the Middle Node?

area/2 + area/2 = area

colored nodes represent fromSource(i)

colored nodes represent toSink(i)
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:

\[
\frac{\text{area}}{8} + \frac{\text{area}}{8} + \frac{\text{area}}{8} + \frac{\text{area}}{8} = \frac{\text{area}}{4}
\]

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?

\[
\text{area} + \frac{\text{area}}{2} + \frac{\text{area}}{4} + \frac{\text{area}}{8} + \frac{\text{area}}{16} + \ldots + < 2 \cdot \text{area}
\]
The Middle Edge

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

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

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

**LinearSpaceAlignment**(*top*,*bottom*,*left*,*right*)

  if *left* = *right*
    return alignment formed by *bottom-top* edges “↓”

  middle ← ⌊(*left*+*right*)/2⌋

  *midNode* ← **MiddleNode**(*top*,*bottom*,*left*,*right*)

  *midEdge* ← **MiddleEdge**(*top*,*bottom*,*left*,*right*)

  **LinearSpaceAlignment**(*top*,*midNode*,*left*,*middle*)

  output *midEdge*

  if *midEdge* = “→” or *midEdge* = “↘”
    middle ← middle+1

  if *midEdge* = “↓” or *midEdge* = “↘”
    *midNode* ← midNode+1

  **LinearSpaceAlignment**(*midNode*,*bottom*,*middle*,*right*)
A: space complexity

B: time complexity

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

- yes: The Four Russians Technique
- Arlazarov, V.; Dinic, E.; Kronrod, M.; Faradžev, I.
- 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

NOT EXAMINABLE
Self Alignment

https://www.sciencedirect.com/science/article/pii/S0958166916301082#fig0020
RNA Secondary Structure: The Nussinov Folding Algorithm

ALGORITHMS FOR LOOP MATCHINGS*

RUTH NUSSINOV,† GEORGE PIECZENIK,‡ JERROLD R. GRIGGS§
AND DANIEL J. KLEITMAN§

Abstract. A simplified (two-base) version of the problem of planar folding of long chains (e.g., RNA and DNA biomolecules) is formulated as a matching problem. The chain is prescribed as a loop or circular sequence of letters A and B, n units long. A matching here means a set of A-B base pairings or matches obeying a planarity condition: no two matches may cross each other if drawn on the interior of the loop. Also, no two adjacent letters may be matched. We present a dynamic programming algorithm requiring $O(n^3)$ steps and $O(n^2)$ storage which computes the size of the maximum for the given A-B base sequence and which also allows reconstructing a particular folded form of the original string which realizes the maximum matching size. The algorithm can be adapted to deal with sequences with larger alphabets and with weighted matchings.

An algorithm is also presented for a modified problem closer to the biochemical problem of interest: We demand that every match must be adjacent to another match, forcing groups of two or more parallel matches.

Some results on the expected maximum matching size are presented. As $n \to \infty$, at least 80% of the vertices can be matched on the average on an A-B string of size $n$.

We briefly discuss the practical application of the algorithm by using contracted versions of very long molecules with a preliminary block construction. A maximum matching is presented for the J-gene of the φX174 DNA virus. We conclude by stating some problems requiring further study.

* Supported by the National Science Foundation.
† Arizona State University, Tempe, Arizona 85287.
‡ The Pennsylvania State University, University Park, Pennsylvania 16802.
§ New York University, New York, New York 10003.

Fig. 1. An example of a maximum matching on a small chain (n = 17). Here M.M.S. = 7.
RNA Secondary Structure
secondary structure= topology of local segments

- 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 \(j-i+1\).

119
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 $L$:

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

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

RNA Secondary Structure: The Nussinov Folding Algorithm

Nussinov Folding Algorithm:

After scores for subsequences of length 2

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

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Nussinov Folding Algorithm:
After scores for subsequences of length 3

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

<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>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>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>
Nussinov Folding Algorithm
After scores for subsequences of length 4

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>U</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>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>0</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>1</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>
<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
After scores for subsequences of length 6

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

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Nussinov Folding Algorithm

After scores for subsequences of length 7

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

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

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

After scores for subsequences of length 9

$$\gamma(i, j) =$$

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

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>U</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>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></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></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></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></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></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></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></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></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Nussinov Folding Algorithm
Traceback
Nussinov algorithm (a different example): fill-stage

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

Scoring system:
\( \delta(i, j) = 1 \) 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.

Nussinov algorithm: traceback

<table>
<thead>
<tr>
<th>G</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
</tr>
<tr>
<td>U</td>
<td>7</td>
</tr>
<tr>
<td>U</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
</tr>
</tbody>
</table>

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

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

current record stack
1 9
1 9
1 8

G • C
G • U
G • C
A • U
G • C
C

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

131
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.
**Initialisation** $\gamma(i, i-1) = 0 \& \gamma(i, i) = 0$

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

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.
Phylogeny

Ancestral Node or ROOT of the Tree

Branches or Lineages

Internal Nodes

Terminal Nodes

A

B

C

D

E

unrooted

rooted

time

((A,(B,C)),(D,E)) = The above phylogeny as nested parentheses
Generating Pylogenetict Tree of Homogeneous Source Code in a Plagiarism Detection System

Jeong-Hoon Ji, Su-Hyun Park, Gyun Woo*, and Hwan-Gue Cho

Abstract: Program plagiarism is widespread due to intelligent software and the global Internet environment. Consequently, the detection of plagiarized source code and software is becoming important especially in academic field. Though numerous studies have been reported for detecting plagiarized pairs of codes, we cannot find any profound work on understanding the underlying mechanisms of plagiarism. In this paper, we study the evolutionary process of source codes regarding that the plagiarism procedure can be considered as evolutionary steps of source codes. The final goal of our paper is to reconstruct a tree depicting the evolution process in the source code. To this end, we extend the well-known bioinformatics approach, a local alignment approach, to detect a region of similar code with an adaptive scoring matrix. The asymmetric code similarity based on the local alignment can be considered as one of the main contributions of this paper. The phylogenetic tree or evolution tree of source codes can be reconstructed using this asymmetric measure. To show the effectiveness and efficiency of the phylogeny construction algorithm, we conducted experiments with more than 100 real source codes which were obtained from East-Asia ICPC (International Collegiate Programming Contest). Our experiments showed that the proposed algorithm is quite successful in reconstructing the evolutionary direction, which enables us to identify plagiarized codes more accurately and reliably. Also, the phylogeny construction algorithm is successfully implemented on top of the plagiarism detection system of an automatic program evaluation system.
Comparative study of recent MEA malware phylogeny

Joanna Moubarak, Maroun Chamoun, Eric Filiol
Published 2017 in 2017 2nd International Conference on Computer and...

Governments in the MEA did not take cyberwarfare seriously a few years ago. Nowadays, there is a shift to a more concerned posture on the subject of cyber security after a series of public revelations of networks being penetrated around the region. The struggle unpacked by the Stuxnet malware in 2009 and then pursued through Duqu, Flame, Shamoon, Gauss, Duqu2.0, Shamoon 2.0 and Stonedrill malware. This paper is a technical survey and a proof of concept of the operating vectors utilized by these malware. It takes this very complex approach, and shows how common stealth and evasion functions and similar stealth methodologies have greatly abridged this undertaking. It provides the understanding needed to analyze and go through the history and development of the most remarkable attacks in the Middle East, their objectives and describes the similarities involved in that process. However, it focuses this around the actual downsides of each malware analyzed and what make it vulnerable or detected. The main purpose of this paper is to highlight the phylogenetic aspects infused in cyberattacks.

LESS
Additivity
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)
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>ACGTAGGCTT</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

$D_{i,j} =$ 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>ACGTGGCCT</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>TCGAGGCAC</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>

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

- **Bacteria**
- **Archaea**
- **Eukaryotes**
  - **Plants**
    - **Seed plants**
      - **Flowering**
      - **Non-flowering**
        - **Ferns**
        - **Mosses**
        - **Sponges**
        - **Cnidarians**
        - **Green algae**
        - **Fungi**
  - **Animals**
    - **Flatworms**
    - **Rotifers**
    - **Roundworms**
    - **Ctenophores**
    - **Arthropods**
      - **Echinoderms**
      - **Cartilaginous fish**
      - **Bony fish**
      - **Segmented worms**
      - **Crustaceans**
      - **Insects**
    - **Vertebrates**
      - **Cartilaginous fish**
      - **Bony fish**
      - **Segmented worms**
      - **Amphibians**
      - **Mammals**
      - **Turtles**
      - **Snakes & lizards & birds**
    - **Tetrapods**
      - **Amniotes**
      - **Mammals**
      - **Amphibians**
      - **Reptiles**
      - **Birds**
      - **Turtles**
      - **Crocodiles**
      - **Snakes & lizards & birds**
**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.
## 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>

![Tree Diagram]

[Tree Diagram showing Chimp and Whale as children of a common node, with distances labeled as 1, 2, 3, and 0.]
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.

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

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

<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>

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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
More Than One Tree Fits a Matrix

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimp</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Human</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Seal</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Whale</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Which Tree is “Better”? 

Simple tree: tree with no nodes of degree 2.

Theorem: There is a unique simple tree fitting an additive matrix.
Distance-Based Phylogeny Problem: Construct an evolutionary tree from a distance matrix.

- **Input:** A distance matrix.
- **Output:** The simple tree fitting this distance matrix (if this matrix is additive).
## An Idea for Distance-Based Phylogeny

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

![Phylogenetic tree](image)
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

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

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>Chimp</th>
<th>Human</th>
<th>Seal</th>
<th>Whale</th>
<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:

- Chimp (0, 3, 6, 4, 4)
- Human (3, 0, 7, 5, 5)
- Seal (6, 7, 0, 2, 2)
- Whale (4, 5, 2, 0, 0)
- m (4, 5, 2, 0, 0)
An Idea for Distance-Based Phylogeny

<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 of phylogeny](image)
An Idea for Distance-Based Phylogeny

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

Diagram:

- Chimp
- Human
- Seal
- Whale

Edges:
- Chimp to Human: ?
- Human to Seal: ?
- Seal to Whale: 2
- Whale to Seal: 0

Table:

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

$$d_{	ext{Chimp},a} = (D_{	ext{Chimp},m} + D_{	ext{Chimp,Human}} - D_{	ext{Human},m}) / 2$$
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
- Seal
- Whale

Edges:
- Chimp to Human: 1
- Human to a: 2
- a to Seal: 2
- Seal to m: 0
- m 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>

Diagram: 
- Chimp to Seal with distance 1
- Human to Seal with distance 2
- Human to Whale with distance 0
- Chimp to Whale with distance 2
- Chimp to Human with distance 3
- Seal to Whale with distance m
- Whale to Seal with distance m
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.

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

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

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

The 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

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

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

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

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

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

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

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

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

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

**Limb Length Theorem:** \( \text{LimbLength}(\text{chimp}) \) is equal to the minimum value of \( \frac{(D_{\text{chimp},k} + D_{\text{chimp},j} - D_{j,k})}{2} \) 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>

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

**Limb Length Theorem:** \( \text{LimbLength}(\text{chimp}) \) is equal to the minimum value of \( \frac{D_{\text{chimp},k} + D_{\text{chimp},j} - D_{j,k}}{2} \) 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

\[ D \]

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

1. Pick an arbitrary leaf $j$. 

$$
D = \begin{bmatrix}
i & j & k & l \\
i & 0 & 13 & 21 & 22 \\
j & 13 & 0 & 12 & 13 \\
k & 21 & 12 & 0 & 13 \\
l & 22 & 13 & 13 & 0 \\
\end{bmatrix}
$$
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

<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>

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

\[ D_{\text{trim}} \]

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

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 $\text{Tree}(D_{\text{trim}})$. 

Additive Phylogeny In Action
6. Identify the point in $Tree(D_{\text{trim}})$ where leaf $j$ should be attached.
AdditivePhylogeny In Action

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

\[LimbLength(j) = 2\]

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

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

\[
D_{\text{bald}} = \begin{pmatrix}
i & j & k & l \\
i & 0 & 11 & 21 & 22 \\
j & 11 & 0 & 10 & 11 \\
k & 21 & 10 & 0 & 13 \\
l & 22 & 11 & 13 & 0
\end{pmatrix}
\]

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

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

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

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

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

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

Sum of Squared Errors

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

\[ = 1^2 + 1^2 = 2 \]
Sum of Squared Errors

**Exercise Break:** Assign lengths to edges in $T$ in order to 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 Phylogeny

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

- **Input:** An $n \times n$ distance matrix $D$.
- **Output:** A weighted tree $T$ with $n$ leaves minimizing $\text{Discrepancy}(T, D)$ over all weighted trees with $n$ leaves.

Unfortunately, this problem is $NP$-Complete...
Ultrametric Trees

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

**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 Monkey
- Baboon
- Orangutan
- Gorilla
- Chimpanzee
- Bonobo
- Human

- Edge weights from Squirrel Monkey to Baboon: 33
- Edge weights from Baboon to Orangutan: 23
- Edge weights from Orangutan to Gorilla: 10
- Edge weights from Gorilla to Chimpanzee: 6
- Edge weights from Chimpanzee to Bonobo: 1
- Edge weights from Bonobo to Human: 2
- Edge weights from Human: 6
Ultrametric tree: distance from root to any leaf is the same (i.e., age of root).
UPGMA: A Clustering Heuristic

1. Form a cluster for each present-day species, each containing a single leaf.
2. Find the two closest clusters $C_1$ and $C_2$ according to the average distance

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

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

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

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

{ $k$, $l$ }

\begin{center}
\begin{figure}[h]
\centering
\begin{tabular}{cccc}
\hline
$i$ & $0$ & $j$ & $0$ & $k$ & $0$ & $l$ & $0$ \\
\hline
\end{tabular}
\end{figure}
\end{center}
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**

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

{ $k$, $l$ }
UPGMA: A Clustering Heuristic

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

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

6. Iterate until a single cluster contains all species.

\[
\begin{array}{ccc}
  i & j & \{k, l\} \\
  i & 0 & 3 & 3.5 \\
  j & 3 & 0 & 4.5 \\
  \{k, l\} & 3.5 & 4.5 & 0 \\
\end{array}
\]
6. Iterate until a single cluster contains all species.
UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.
UPGMA: A Clustering Heuristic

6. Iterate until a single cluster contains all species.
UPGMA: A Clustering Heuristic

**UPGMA(D):**

1. Form a cluster for each present-day species, each containing a single leaf.
2. Find the two closest clusters $C_1$ and $C_2$ according to the average distance
   \[ D_{\text{avg}}(C_1, C_2) = \sum_{i \in C_1, j \in C_2} D_{i,j} / |C_1| \cdot |C_2| \]
   where $|C|$ denotes the number of elements in $C$
3. Merge $C_1$ and $C_2$ into a single cluster $C$.
4. Form a new node for $C$ and connect to $C_1$ and $C_2$ by an edge. Set age of $C$ as $D_{\text{avg}}(C_1, C_2)/2$.
5. Update the distance matrix by computing the average distance between each pair of clusters.
6. Iterate steps 2-5 until a single cluster contains all species.
UPGMA Doesn’t “Fit” a Tree to a Matrix

<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 Doesn’t “Fit” a Tree to a Matrix

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

![Tree Diagram](image_url)
In Summary...

- **AdditivePhylogeny:**
  - good: produces the tree fitting an *additive* matrix
  - bad: fails completely on a *non-additive* matrix

- **UPGMA:**
  - good: produces a tree for any matrix
  - bad: tree doesn’t necessarily fit an additive matrix

- **???????:**
  - good: produces the tree fitting an additive matrix
  - good: provides heuristic for a non-additive matrix
Neighbor-Joining Theorem

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

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

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

**Example:**

$$D = \begin{pmatrix}
0 & 13 & 21 & 22 \\
13 & 0 & 12 & 13 \\
21 & 12 & 0 & 13 \\
22 & 13 & 13 & 0
\end{pmatrix}$$

$$D^{*} = \begin{pmatrix}
0 & -68 & -60 & -60 \\
-68 & 0 & -60 & -60 \\
-60 & -60 & 0 & -68 \\
-60 & -60 & -68 & 0
\end{pmatrix}$$
Neighbor-Joining Theorem: If $D$ is additive, then the smallest element of $D^*$ corresponds to neighboring leaves in $\text{Tree}(D)$.

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

<table>
<thead>
<tr>
<th></th>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
<th>$l$</th>
<th>$D^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>0</td>
<td>-68</td>
<td>-60</td>
<td>-60</td>
<td></td>
</tr>
<tr>
<td>$j$</td>
<td>-68</td>
<td>0</td>
<td>-60</td>
<td>-60</td>
<td></td>
</tr>
<tr>
<td>$k$</td>
<td>-60</td>
<td>-60</td>
<td>0</td>
<td>-68</td>
<td></td>
</tr>
<tr>
<td>$l$</td>
<td>-60</td>
<td>-60</td>
<td>-68</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
1. Construct neighbor-joining matrix $D^*$ from $D$. 

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

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

2. Find a minimum element $D^*_{i,j}$ of $D^*$. 
2. Find a minimum element $D^*_{i,j}$ of $D^*$.
3. Compute $\Delta_{i,j} = (\text{TotalDistance}_D(i) - \text{TotalDistance}_D(j)) / (n - 2)$. 
# 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>13</td>
<td>21</td>
<td>22</td>
<td>56</td>
</tr>
<tr>
<td>$j$</td>
<td>13</td>
<td>0</td>
<td>12</td>
<td>13</td>
<td>38</td>
</tr>
<tr>
<td>$k$</td>
<td>21</td>
<td>12</td>
<td>0</td>
<td>13</td>
<td>46</td>
</tr>
<tr>
<td>$l$</td>
<td>22</td>
<td>13</td>
<td>13</td>
<td>0</td>
<td>48</td>
</tr>
</tbody>
</table>

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

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

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

4. Set \(\text{LimbLength}(i)\) equal to \(\frac{1}{2}(D_{i,j} + \Delta_{i,j})\) and \(\text{LimbLength}(j)\) equal to \(\frac{1}{2}(D_{i,j} - \Delta_{i,j})\).
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 $k$, $D_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$. 

\[
\begin{array}{ccc|c}
 m & k & l & \text{TotalDistance}_D \\
 \hline
 m & 0 & 10 & 11 & 21 \\
 D' & k & 10 & 0 & 13 & 23 \\
 l & 11 & 13 & 0 & 24 \\
\end{array}
\]
Flashback: Computation of $d_{k,m}$

\[
\begin{align*}
    d_{k,m} &= \frac{[(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})]}{2} \\
    d_{k,m} &= \frac{(d_{i,k} + d_{j,k} - d_{i,j})}{2} \\
    d_{k,m} &= \frac{(D_{i,k} + D_{j,k} - D_{i,j})}{2}
\end{align*}
\]
Neighborhood-Joining in Action

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

$\begin{array}{ccc}
m & k & l \\
m & 0 & 10 & 11 \\
k & 10 & 0 & 13 \\
l & 11 & 13 & 0 \\
\end{array}$
Neighbor-Joining in Action

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

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

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

7. Reattach limbs of \(i\) and \(j\) to obtain \(\text{Tree}(D)\).
7. Reattach limbs of $i$ and $j$ to obtain $\text{Tree}(D)$.
Neighbor-Joining

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

**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 $O(r^3)$. 
**Neighbor-Joining**

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

<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 diagram:

```
    2
   /\ 1
  /  \ 1
 i  j  k  l
```

Note: The UPGMA tree diagram is simplified for illustrative purposes. The actual tree structure and distances are shown in the text.
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>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>TCGAAAGCAGCAT</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Example with a 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>

<table>
<thead>
<tr>
<th></th>
<th>U₁</th>
<th>U₂</th>
<th>U₃</th>
<th>U₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>U₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 1

S calculations

\(S_A = \frac{(5+4+7+6+8)}{4} = 7.5\)
\(S_B = \frac{(5+7+10+9+11)}{4} = 10.5\)
\(S_C = \frac{(4+7+6+8+3)}{4} = 8\)
\(S_D = \frac{(7+10+7+5+9)}{4} = 9.5\)
\(S_E = \frac{(6+9+6+5+3)}{4} = 8.5\)
\(S_F = \frac{(8+11+8+9+8)}{4} = 11\)

Step 2

Smallest are

\(M_{BD} = 5 - 7.5 = -10.5\)
\(M_{BC} = 5 - 9.5 = -4\)

Choose one of these (AB here).

Smallest is

\(M_{CU₁} = 3 - 7.5 = -4.5\)
\(M_{CD} = 5 - 9 = -4\)

Choose one of these (DE here).

Step 3

Create a node (U) that joins pair with lowest \(M_u\) such that

\(S_{UA} = D_{AB} + (S_A - S_B) = 2\)
\(S_{UB} = D_{AB} + (S_B - S_A) = 4\)

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

\(D_{UJ} = D_{ij} + D_{ii} - D_{ii}\),

where i and j are those selected from above.

Comments

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

\[
\{ \text{n species} \}
\]

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

- Chimp: ACGTAGGCCT
- Human: ATGTAAGACT
- Seal: TCGAGAGCAC
- Whale: 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**
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.
Toward a Computational Problem

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

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

ACG
T
AGGCCT

ACG
A
AAGACT

TCG
A
GAGCAC

ACG
A
AAGCCT

Chimp

ACG
T
AGGCCT

ATG
T
AAGACT

TCG
A
GAGCAC

Seal

TCG
A
AAGCCT

Whale
A Dynamic Programming Algorithm

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

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

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

For symbols $i$ and $j$, define

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

Exercise Break: Prove the following recurrence relation:

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

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

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

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

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

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

Code Challenge: Solve the Small Parsimony Problem.
Recreating a Functional Ancestral Archosaur Visual Pigment

Belinda S. W. Chang, Karolina Jönsson, Manija A. Kazmi, Michael J. Donoghue, Thomas P. Sakmar

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
Small Parsimony for Unrooted Trees

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

Parsimony Score: 8
Finding the Most Parsimonious Tree

Parsimony Score: 11
Finding the Most Parsimonious Tree

Parsimony Score: 14
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.
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.

Unfortunately, this problem is $NP$-Complete...
A Greedy Heuristic for Large Parsimony

Note that removing an **internal edge**, an edge connecting two internal nodes (along with the nodes), produces four subtrees \((W, X, Y, Z)\).
A Greedy Heuristic for Large Parsimony

Note that removing an **internal edge**, an edge connecting two internal nodes (along with the nodes), produces four subtrees ($W$, $X$, $Y$, $Z$).
A Greedy Heuristic for Large Parsimony

Note that removing an internal edge, an edge connecting two internal nodes (along with the nodes), produces four subtrees (W, X, Y, Z).
A Greedy Heuristic for Large Parsimony

Rearranging these subtrees is called a nearest neighbor interchange.
A Greedy Heuristic for Large Parsimony

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

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

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

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

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

- 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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>A</td>
<td>T</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>G</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>C</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>T</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>G</td>
<td>C</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>T</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

Inferred tree

(b) Pseudosample 1

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

Pseudosample 2

Pseudosample $n$

Bootstrap trees

(b) Subhypothesis 1

95% is significantly positive
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
Distance algorithm in computer science
A) A sequence logo for the FakeAV-DO function “F1”. Positions with large characters indicate invariant parts of the function; positions with small characters vary due to code metamorphism

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

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

(W.M. Khoo Unity in diversity: Phylogenetic-inspired techniques for reverse engineering and detection of malware families)
More species increases power to detect conserved elements

Data from Eric Green at NGHRI, alignments by Webb Miller
Generalizing Pairwise to Multiple Alignment

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

\[
\begin{align*}
\text{A} & \quad \text{T} & \quad \text{G} & \quad \text{C} & \quad \text{G} & \quad - \\
\text{A} & \quad - & \quad \text{C} & \quad \text{G} & \quad \text{T} & \quad - & \quad \text{A} \\
\text{A} & \quad \text{T} & \quad \text{C} & \quad \text{A} & \quad \text{C} & \quad - & \quad \text{A}
\end{align*}
\]

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

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

- \( \delta(x, y, z) \) 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:

\[
\begin{align*}
\text{AC} & \rightarrow \text{GC} \rightarrow \text{GG} \rightarrow \text{C} \\
\text{AC} & \rightarrow \text{GC} \rightarrow \text{GAG} \\
\text{G} & \rightarrow \text{CC} \rightarrow \text{CGC} \rightarrow \text{GAG} \\
\end{align*}
\]

\[
\begin{align*}
\text{ACGCGG-C} & \rightarrow \text{AC-GCGG-C} \rightarrow \text{AC-GCGAG} \\
\text{ACGC-GAC} & \rightarrow \text{GCCGC-GAG} \rightarrow \text{GCCGCGAG} \\
\end{align*}
\]
Idea: Construct Multiple from Pairwise Alignments

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

```
AAAATTTT----
----TTTTGGGG
GGGGAAAA----
TTTTGGGG----
```
Progressive alignment methods are heuristic in nature. They produce multiple alignments from a number of pairwise alignments. Perhaps the most widely used algorithm of this type is CLUSTALW.
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)
Progressive Alignment

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?

Jotun Hein has developed methods to build alignment and tree

Simultaneous Bayesian Estimation of Alignment and Phylogeny under a Joint Model of Protein Sequence and Structure

Joseph L. Herman, Joe T. Challis, Ádám Novák, Jotun Hein, and Scott C. Schmidler

Abstract

For sequences that are highly divergent, there is often insufficient information to infer accurate alignments, and phylogenetic uncertainty may be high. One way to address this issue is to make use of protein structural information, since structures generally diverge more slowly than sequences. In this work, we extend a recently developed stochastic model of pairwise structural evolution to multiple structures on a tree, analytically integrating over ancestral structures to permit efficient likelihood computations under the resulting joint sequence-structure model. We observe that the inclusion of structural information significantly reduces alignment and topology uncertainty, and reduces the number of topology and alignment errors in cases where the true trees and alignments are known. In some cases, the inclusion of structure results in changes to the consensus topology, indicating that structure may contain additional information beyond that which can be obtained from sequences. We use the model to investigate the order of divergence of cytochrome c, myoglobin, and hemoglobin and observe a stabilization of phylogenetic inference: although a sequence-based inference assigns significant posterior probability to several different topologies, the structural model strongly favors one of these over the others and is more robust to the choice of data set.
Genome alignment

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
- The String Reconstruction Problem
- String Reconstruction as a Hamiltonian Path Problem
- String Reconstruction as an Eulerian Path Problem
- Similar Problems with Different Fates
- De Bruijn Graphs
- Euler’s Theorem
- Assembling Read-Pairs
- De Bruijn Graphs Face Harsh Realities of Assembly
Next Generation Sequencing Technologies

• **Late 2000s**: The market for new sequencing machines takes off.
  – Illumina reduces the cost of sequencing a human genome from $3 billion to $10,000.
  – Complete Genomics builds a genomic factory in Silicon Valley that sequences hundreds of genomes per month.
  – Beijing Genome Institute orders hundreds of sequencing machines, becoming the world’s largest sequencing center.
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”

“Burning” Some Reads
No Idea What Position Every Read Comes From
From *Experimental* to *Computational* Challenges

Multiple (unsequenced) genome copies

Reads

Assembled genome

...GGCATCGTCAGAAACTATCATAGCTAGATCGTACG{TAGCC...
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?

$Composition_3(TAATGCCATGGGATGTT) =$

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

\[ Composition_3(TAATGCCATGGATGTT) = \]
\[ TAA\ AAT\ ATG\ TGC\ GCC\ CCA\ CAT\ ATG\ TGG\ GGG\ GGA\ GAT\ ATG\ TGT\ GTT \]
\[ = \]
\[ AAT\ ATG\ ATG\ ATG\ CAT\ CCA\ GAT\ GCC\ GGA\ GGG\ GTT\ TAA\ TGC\ TGG\ TGT \]

e.g., lexicographic order (like in a dictionary)
Reconstructing a String from its Composition

String Reconstruction Problem. Reconstruct a string from its k-mer composition.

• **Input.** A collection of k-mers.

• **Output.** A Genome such that Composition\(_k\)(Genome) is equal to the collection of k-mers.
A Naive String Reconstruction Approach
Representing a Genome as a Path

Composition<sub>3</sub>(TAATGCCATGGGATGTT) =

Can we construct this genome path without knowing the genome TAATGCCATGGGATGTT, only from its composition?

Yes. We simply need to connect k-mer<sub>1</sub> with k-mer<sub>2</sub> if suffix(k-mer<sub>1</sub>) = prefix(k-mer<sub>2</sub>). E.g. TAA → AAT.

[Diagram of genome path with arrows connecting TAA, AAT, ATG, TGC, GCC, CCA, CAT, ATG, TGG, GGG, GGA, GAT, ATG, TGT, GTT.]
A Path Turns into a Graph

Yes. We simply need to connect \( k\text{-mer}_1 \) with \( k\text{-mer}_2 \) if \( \text{suffix}(k\text{-mer}_1) = \text{prefix}(k\text{-mer}_2) \).

E.g. TAA → AAT
A Path Turns into a Graph

Can we still find the genome path in this graph?
A **Hamiltonian path**: a path that visits each node in a graph exactly once.

What are we trying to find in this graph?
Does This Graph Have a Hamiltonian Path?

Hamiltonian Path Problem. Find a Hamiltonian path in a graph.
Input. A graph.
Output. A path visiting every node in the graph exactly once.

Icosian game (1857)

Undirected graph
TA TGGGATGCCATGTT

TA TGCCATGGGATGTT
A Slightly Different Path

`TAATGCCATGGGATGTT`

3-mers as nodes

3-mers as edges

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

prefix of TAA

suffix of TAA
Labeling Nodes in the New Path

TAATGCCATGGGATGTT

3-mers as nodes

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

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

TAATGCCATGGGATGTT
Gluing Identically Labeled Nodes

TA ATG CC AT GG GAT GTT

TA A A

AT

TT

TA

AA

ATG

TG

GG

GG

GGA

GG

ATG

TG

GAT

GA

CAT

CA

CCA

CC

GCC

TGC

TA

320
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

TAATGC CATTGG GTT

\[
\begin{align*}
\text{TA} & \rightarrow \text{AA} \\
\text{AT} & \rightarrow \text{CC} \\
\text{G} & \rightarrow \text{CC} \\
\text{T} & \rightarrow \text{G} \\
\text{G} & \rightarrow \text{T} \\
\end{align*}
\]
What We Want: From Reads (k-mers) to Genome

TAATGCCATGGGATGTT

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT
What We will Show: From Reads to de Bruijn Graph to Genome
Constructing de Bruijn Graph when Genome Is Known

TAATGCCATGGGATGTT
Constructing de Bruijn when Genome Is Unknown

Composition_{3}(TAATGCCATGGGATGTT)
Representing Composition as a Graph Consisting of Isolated Edges

Composition$_3$(TAATGCCATGGGATGTT)
Constructing de Bruijn Graph from k-mer Composition

Composition$_3$(TAATGCCCATGGGATGTT)
Gluing Identically Labeled Nodes

335
We Are Not Done with Gluing Yet
Gluing Identically Labeled Nodes

TA → AA → AT → TG → GC → CC → CA → AT → TG → GG → GG → GA → AT → TG → GT → TT
Gluing Identically Labeled Nodes
Gluing Identically Labeled Nodes
The Same de Bruijn Graph: 
DeBruin(Genome) = DeBruin(Genome Composition)
Constructing de Bruijn Graph

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

- Represent every $k$-mer as an edge between its prefix and suffix
- Glue **ALL** nodes with identical labels.

DeBruijn($k$-mers)
form a node for each ($k-1$)-mer from $k$-mers
for each $k$-mer in $k$-mers
  connect its prefix node with its suffix node by an edge
Universal String Problem (Nicolaas de Bruijn, 1946). Find a circular string containing each binary k-mer exactly once.

000 001 010 011 100 101 110 111
Universal String Problem (Nicolaas de Bruijn, 1946). Find a circular string containing each binary k-mer exactly once.

000  001  010  011  100  101  110  111
From Hamilton to Euler to de Bruijn
De Bruijn Graph for 4-Universal String

Does it have an Eulerian cycle? If yes, how can we find it?
Eulerian CYCLE Problem

Eulerian CYCLE Problem. Find an Eulerian cycle in a graph.

- Input. A graph.

- Output. A cycle visiting every edge in the graph exactly once.
A Graph is **Eulerian** if it contains an Eulerian Cycle.

Is this graph Eulerian?
A Graph is **Eulerian** if it contains an Eulerian Cycle.

Is this graph Eulerian?

A graph is balanced if $\text{indegree} = \text{outdegree}$ 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

\[ TAATGCCATGGGATGTT \]

\[ \text{AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT} \]
Multiple Eulerian Paths

TA TGCCATGGGATGTT
A

TA TGGGATGCCATGTT
A
Breaking Genome into Contigs
DNA Sequencing with Read-pairs

Multiple identical copies of genome

Randomly cut genomes into large equally sized fragments of size InsertLength

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

A paired *k*-mer is a pair of *k*-mers at a fixed distance *d* apart in Genome.
E.g. TCA and TCC are at distance *d*=11 apart.

Disclaimers:
1. In reality, **Read1** and **Read2** are typically sampled from different strands:
   (→ ....... ← rather than → ....... →)
2. In reality, the distance *d* between reads is measured with errors.
What is \text{PairedComposition}(TAATGCCATGGGATGTT)\? 

Representing a paired 3-mer TAATGCC as a 2-line expression:
PairedComposition(\texttt{TAATGC\textcolor{red}{CC}ATGG\textcolor{olive}{ATG}TT})

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 $\text{TAAATGCCATGGGATGTT}$ as a Path

Paired prefix of $\text{TAAATGCCATGGGATGTT}$
Labeling Nodes by Paired Prefixes and Suffixes
Glue nodes with identical labels
Glue nodes with identical labels

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

![Paired de Bruijn Graph Diagram]

Paired prefix of ▶️ CCA GGG ← Paired suffix of
A 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

TAATGCCATGGGATGTT

TAATGGGATGCCATGTT

Paired de Bruijn Graph

De Bruijn Graph
Some Ridiculously Unrealistic Assumptions

- Perfect coverage of genome by reads (every $k$-mer from the genome is represented by a read)
- Reads are error-free.
- Multiplicities of $k$-mers are known
- Distances between reads within read-pairs are exact.
Some Ridiculously Unrealistic Assumptions

- **Imperfect** coverage of genome by reads (every $k$-mer from the genome is represented by a read)

- Reads are **error-prone**.

- Multiplicities of $k$-mers are **unknown**.

- Distances between reads within read-pairs are **inexact**.

- Etc., etc., etc.
1st Unrealistic Assumption: Perfect Coverage

250-nucleotide reads generated by Illumina technology capture only a small fraction of 250-mers from the genome, thus violating the key assumption of the de Bruijn graphs.
Breaking Reads into Shorter $k$-mers

atgccgtatggacacgact
atgccgtatg
gccgtatggga
gtatggacaa
gacaacgact

atgccgtatggacacgact
atgcc
tgccg
gccgt
cgta
cgtat
gtatg	atatg
tatgg
atgga
tggac
ggaca
gacaa
acaac
caacg
aacga
acgac
cgact
2\textsuperscript{nd} Unrealistic Assumption: Error-free Reads

atgccgtatggacaaacgact
atgccgtatg
gccgtatggga
gtatggacaa
gacaacgact
cgtaCggaca

atgccgtatggacaaacgact
atgcc
tgccg
gccgt
ccgta
cgtat
gtatg	tatgg
atgga	tggac
ggaca
gacaa
acaac
caacg
aacga
acgac
cgact
cgtaC
gtaCg
taCgg
aCgga
Cgga
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
RECAP

Input: GGC\textsuperscript{CTCTATATCTCGGCTCTAGGCC\textsubscript{CTCATTTTTTT}}

Copy: GGC\textsuperscript{CTCTATATCTCGGCTCTAGGCC\textsubscript{CTCATTTTTTT}}
GGC\textsuperscript{CTCTATATCTCGGCTCTAGGCC\textsubscript{CTCATTTTTTT}}
GGC\textsuperscript{CTCTATATCTCGGCTCTAGGCC\textsubscript{CTCATTTTTTT}}
GGC\textsuperscript{CTCTATATCTCGGCTCTAGGCC\textsubscript{CTCATTTTTTT}}

Fragment: GGC\textsuperscript{GTCTA} TAT\textsuperscript{CTCGG} CTCTAGGCC\textsubscript{CTCATTTTTTT}
GGC\textsuperscript{GTCTATAT} CT\textsuperscript{CGGCTCTAGGCC\textsubscript{CTCATTTTTTT}}
GGC\textsuperscript{GTCTATAT} CT\textsuperscript{ATCTCGGCTCTAGGCC\textsubscript{CTCATTTTTTT}}
GGC\textsuperscript{GTCTATAT} AT\textsuperscript{CTCGGCTCTAGGCC\textsubscript{CTCATTTTTTT}}

\begin{align*}
\text{CTAGGCC\textsubscript{CTCATTTTTTT}} \\
\text{CTCTAGGCC\textsubscript{CTCATTTTTTT}} \\
\text{G\textsuperscript{GCTCTAGGCC\textsubscript{CTCATTTTTTT}}} \\
\text{CTCGGCTCTAGGCC\textsubscript{CTCATTTTTTT}} \\
\text{T\textsuperscript{ATCTCGA}CT\textsuperscript{TCTAGGCC\textsubscript{CTCATTTTTTT}}} \\
\text{T\textsuperscript{ATCTCGA}CT\textsuperscript{TCTAGGCC\textsubscript{CTCATTTTTTT}}} \\
\text{T\textsuperscript{CTATATCTCGGCTCTAGG}C\textsubscript{CTCATTTTTTT}} \\
\text{GGC\textsuperscript{GTCTATATCTCG}G} \\
\text{GGC\textsuperscript{GTCTATATCTCG}G} \\
\text{GGC\textsuperscript{GTCTATATCTCG}G} \\
\text{GGC\textsuperscript{GTCTATATCTCG}G} \\
\text{GGC\textsuperscript{GTCTATATCTCG}G} \\
\text{GGC\textsuperscript{GTCTATATCTCG}G} \\
\text{GGC\textsuperscript{GTCTATATCTCG}G} \\
\text{GGC\textsuperscript{GTCTATATCTCG}G} \\
\text{GGC\textsuperscript{GTCTATATCTCG}G}
\end{align*}

177 nucleotides

35 nucleotides

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

$S$: GCGATTCA$\text{T}CG$

A 4-mer of $S$: ATTC

All 3-mers of $S$: GGC, GCG, CGA, GAT, ATT, TTC, TCA, CAT, ATC, TCG

I’ll use “$k$-1-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.
RECAP

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 0000110010111101.
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 $k$-mer from the input.

If we add one more B to our input string: \textbf{AAABBBBA}, and rebuild the De Bruijn graph accordingly, we get a \textit{multiedge}.
RECAP

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.)

Is it Eulerian? Yes

Argument 1: \(AA \rightarrow AA \rightarrow AB \rightarrow BB \rightarrow BB \rightarrow BA\)

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: \( \text{a\_long\_long\_long\_long\_time} \)

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_long_time
RECAP

Last 5 k-mer additions, \( k = 5 \)

a_long_long_long_time

Finished graph
De Bruijn graph

**RECAP**

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

RECAP

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?
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
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

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

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

- Errors in middle of read
  - Pop Bubbles

- Chimeric Edges
  - Clip short, low coverage nodes
“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, . . . .”


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

Reads:  theageofw  sthebestof  astheage  wors  umesitwast

Kmers:  the  sth  ast  wor  ime
(k=3)  hea  the  sth  ors  mes
   eae  heb  the  rst  es i
   age  ebe  hea  sto  sit
   geo  bes  eae  oft  twa
   eof  est  age  oft  tti
   ofw  sto  geo  fti  was

Step 2:
Build a De-Bruijn graph from the kmers
Step 3: Simplify the graph as much as possible:

A De Bruijn Graph

De Bruijn assemblies ‘broken’ by repeats longer than kmer

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

The final assembly \((k=3)\)

- wor
- times
- it wasthe
- foolishness
- st
- wisdom
- incredulity
- age
- epoch
- be
- of
- belief

Repeat with a longer "kmer" length

A better assembly \((k=20)\)

- it was 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:

- \(k=3\)
  - st, the
  - heb
  - ebe
  - ben
  - nto

- \(k=10\)
  - st, the
  - heb
  - ebe
  - ben
  - nto

Mostly unaffected kmers

100% wrong kmer

Slides from Presentation by Alicia Clum genomebiology.jgi-psf.org/Content/MGM-13.Sep2012/.../3.clum.ppt
most of Bioinformatics focuses on genes, mRNA (i.e. gene activity) and proteins.
gene expression

Clustering

gene network

gene interaction

Page Rank analysis for genes for normal and cancer
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:

- **YLR258W**: 1.1 1.4 1.4 3.7 4.0 10.0 5.9
- **YPL012W**: 1.1 0.8 0.9 0.4 0.3 0.1 0.1
- **YPR055W**: 1.1 1.1 1.1 1.1 1.1 1.1 1.1

\[ e_{ij} = \text{expression level of gene } i \text{ at checkpoint } j \]
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
https://wiki.yeastgenome.org/index.php/YLR258W
20 gen 2012 - Description of YLR258W. glycogen synthase, similar to Gsy2; 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 isofrom 2 - Saccharomyces ... https://www.uniprot.org/uniprot/P27472
Saccharomyces cerevisiae (strain ATCC 244508 / S288c) [Baker's yeast], Status ... BioCyc; YEAST:YLR258W-MONOMER ... Ordered Locus Names:YLR258W.

Gene Expression Omnibus
GEO is a public functional genomics data repository supporting MIAME-compliant data submissions. Array- and sequence-based data are accepted. Tools are provided to help users query and download experiments and curated gene expression profiles.

Getting Started
Overview
FAQ
Search for Studies at GEO DataSets
Search for Gene Expression at GEO Profiles
About GEO DataSets
About GEO Profiles
About GEO2R Analysis
How to Construct a Query
How to Download Data
Tools
Search GEO Documentation
Analyze a Study with GEO2R
Studies with Genome Data Viewer Tracks
Programmatic Access
FTP Site
Switching to Logarithms of Expression Levels

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>0.2</th>
<th>0.4</th>
<th>0.5</th>
<th>1.1</th>
<th>1.1</th>
<th>1.1</th>
<th>1.1</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>
<td></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>
<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)
```
CHAPTER 1

Gene Expression Vector

YLR361C  0.14  0.03 -0.06  0.07 -0.01 -0.06 -0.01
YMR290C  0.12 -0.23 -0.24 -1.16 -1.40 -2.67 -3.00
YNR065C  -0.10 -0.14 -0.03 -0.06 -0.07 -0.14 -0.04
YGR043C  -0.43 -0.73 -0.06 -0.11 -0.16  3.47  2.64
YLR258W  0.11  0.43  0.45  1.89  2.00  3.42  2.56
YPL012W  0.09 -0.28 -0.15 -1.18 -1.59 -2.96 -3.08
YNL141W  -0.16 -0.04 -0.07 -1.26 -1.20 -2.82 -3.13
YJL028W  -0.28 -0.23 -0.19 -0.19 -0.32 -0.18 -0.18
YKL026C  -0.19 -0.15  0.03  0.27  0.54  3.64  2.74
YPR055W  0.15  0.15  0.17  0.09  0.07  0.09  0.07

The rows of the matrix from Figure 1.3 partitioned into three clusters. Green genes exhibit increased expression, red genes exhibit decreased expression, and 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 $\sigma^2$ 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 Name</th>
<th>Expression Values</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>

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
**Genes as Points in Multidimensional Space**

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

**n x m**

gene expression matrix

**n points in**

**m-dimensional space**

---

**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, and 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 and Cancer Diagnostics

**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:

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

\begin{align*}
\text{MaxDistance}(\text{Data, Centers}) &= \\
\max_{\text{all points DataPoint from Data}} d(\text{DataPoint, Centers})
\end{align*}
**k-Center Clustering Problem**

**k-Center Clustering Problem.** Given a set of points \( Data \), find \( k \) centers minimizing \( \text{MaxDistance}(Data, Centers) \).

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

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

*Intractable*
\textbf{$k$-Center Clustering Heuristic}

\begin{algorithm}
\textbf{FarthestFirstTraversal}(Data, $k$)

\hspace{1em} Centers $\leftarrow$ the set consisting of a single DataPoint from Data
\hspace{1em} \textbf{while} Centers have fewer than $k$ points
\hspace{2em} DataPoint $\leftarrow$ a point in Data maximizing $d$(DataPoint, Centers)
\hspace{2em} among all data points
\hspace{1em} add DataPoint to Centers
\end{algorithm}
**$k$-Center Clustering Heuristic**

*FarthestFirstTraversal*(Data, $k$)

Centers ← the set consisting of a single DataPoint from Data

while Centers have fewer than $k$ points

DataPoint ← a point in Data maximizing $d$(DataPoint, Centers) among all data points

add DataPoint to Centers
What Is Wrong with **FarthestFirstTraversal**?

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

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

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*:

\[
\text{Distortion}(\text{Data, Centers}) = \sum \; \text{DataPoint from Data} \; d(\text{DataPoint, Centers})^2/n
\]

A single data point contributes to **MaxDistance**

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

<table>
<thead>
<tr>
<th><strong>k-Center Clustering Problem:</strong></th>
<th><strong>k-Means Clustering Problem:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A set of points ( \text{Data} ) and an integer ( k ).</td>
<td><strong>Input:</strong> A set of points ( \text{Data} ) and an integer ( k ).</td>
</tr>
<tr>
<td><strong>Output:</strong> A set of ( k ) points ( \text{Centers} ) that minimizes ( \text{MaxDistance}(\text{DataPoints}, \text{Centers}) ) over all choices of ( \text{Centers} ).</td>
<td><strong>Output:</strong> A set of ( k ) points ( \text{Centers} ) that minimizes ( \text{Distortion}(\text{Data}, \text{Centers}) ) over all choices of ( \text{Centers} ).</td>
</tr>
</tbody>
</table>

\( \text{NP-Hard for } k > 1 \)
**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:

$$((2+4+6)/3, (3+1+5)/3) = (4, 3)$$
Select $k$ arbitrary data points as Centers
The Lloyd Algorithm in Action

assign each data point to its nearest center
The Lloyd Algorithm in Action

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

assign each data point to its nearest center
The Lloyd Algorithm in Action

Clusters

Centers

again!

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

Clusters

Centers

assign each data point to its nearest center

again!
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
*k*-means Clustering vs. the Human Eye

How would the *Lloyd algorithm* cluster these sets of points?
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.

- **Midpoint:** A point approximately halfway between two clusters.
Soft vs. Hard Clustering

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

- Can we color the midpoint half-red and half-blue?
  - “soft” assignment of data points to clusters.
**Soft vs. Hard Clustering**

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

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

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

Probability of generating the given sequence of flips is

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

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>HTTTHTTHTH</td>
<td>0.4</td>
</tr>
<tr>
<td>HHHHTHHHHH</td>
<td>0.9</td>
</tr>
<tr>
<td>HTTHHHHTHH</td>
<td>0.8</td>
</tr>
<tr>
<td>HTTTTTHHTT</td>
<td>0.3</td>
</tr>
<tr>
<td>THHHTHHHTH</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>H T T T H T T H T H</td>
<td>0.4</td>
</tr>
<tr>
<td>H H H H T H H H H H</td>
<td>0.9</td>
</tr>
<tr>
<td>H T H H H H H H T H H</td>
<td>0.8</td>
</tr>
<tr>
<td>H T T T T T H H T T T T</td>
<td>0.3</td>
</tr>
<tr>
<td>T H H H T H H H H H H T H</td>
<td>0.7</td>
</tr>
</tbody>
</table>

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

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

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

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

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHTTHHTTHH</td>
<td>0.4 * 1</td>
<td>(0.35, 0.80)</td>
</tr>
<tr>
<td>HHHHTHHHHH</td>
<td>0.9 * 0</td>
<td></td>
</tr>
<tr>
<td>HTHHHHHTTHH</td>
<td>0.8 * 0</td>
<td></td>
</tr>
<tr>
<td>HHTTTTTTHHTT</td>
<td>0.3 * 1</td>
<td></td>
</tr>
<tr>
<td>TTHHTHHHTTH</td>
<td>0.7 * 0</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

Data $\times$ HiddenVector $= (1,1,\ldots,1) \times HiddenVector = 0.35$

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>HHTTHHTTHH</td>
<td>0.4 *</td>
<td>1</td>
</tr>
<tr>
<td>HHHHHTHHHHH</td>
<td>0.9 *</td>
<td>0</td>
</tr>
<tr>
<td>HTHHHHHTHHH</td>
<td>0.8 *</td>
<td>0</td>
</tr>
<tr>
<td>HHTTTTTHHTTT</td>
<td>0.3 *</td>
<td>1</td>
</tr>
<tr>
<td>TTHHTHHHHTTH</td>
<td>0.7 *</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

\[
= \frac{(0.5\times0+0.9\times1+0.8\times1+0.4\times0+0.7\times1)}{(0+1+1+0+1)} = 0.80
\]

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

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

\[
\begin{align*}
\text{Data} & \quad \text{HiddenVector} & \quad \text{Parameters} = (\theta_A, \theta_B) \\
\text{HHTHHTHHTH} & \quad 0.4 & \quad * & \quad 1 \\
\text{HHHHHHTHTHT} & \quad 0.9 & \quad * & \quad 0 \\
\text{HTHHHHHTHTH} & \quad 0.8 & \quad * & \quad 0 & \quad \text{(0.35, 0.80)} \\
\text{HTTTTTTHHTT} & \quad 0.3 & \quad * & \quad 1 \\
\text{THHHTHHTHTH} & \quad 0.7 & \quad * & \quad 0
\end{align*}
\]

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

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

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

HiddenVector → Parameters
Data, HiddenVector, Parameters

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

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

Which coin is more likely to generate the 1\(^{\text{st}}\) sequence (with 4 H)?

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

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

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

\[
\begin{align*}
\Pr(\text{1st sequence}|\theta_A) &= \theta_A^4 (1-\theta_A)^6 = 0.35^4 \cdot 0.65^6 \approx 0.00113 \\
\Pr(\text{1st sequence}|\theta_B) &= \theta_B^4 (1-\theta_B)^6 = 0.80^4 \cdot 0.20^6 \approx 0.00003
\end{align*}
\]

\[\text{Pr}(\text{1st sequence}|\theta_A) > \text{Pr}(\text{1st sequence}|\theta_B)\]
From *Data & Parameters to HiddenVector*

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters=(θ_A, θ_B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>1</td>
<td>(0.35, 0.80)</td>
</tr>
<tr>
<td>0.9</td>
<td>?</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\textsuperscript{nd} sequence (with 9 H)?

Pr(2\textsuperscript{nd} sequence|θ_A)= θ_A^9 (1-θ_A)^1 =0.35^9 \cdot 0.65^1 \approx 0.00005 < Pr(2\textsuperscript{nd} sequence|θ_B)= θ_B^9 (1-θ_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 = (θ_A, θ_B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>1</td>
<td>(0.35, 0.80)</td>
</tr>
<tr>
<td>0.9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>?</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\textsuperscript{nd} sequence (with 9 H)?

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

Data

*HiddenVector’*

*Parameters’*

Iterate!
From Coin Flipping to k-means Clustering: Where Are \textit{Data, HiddenVector, and Parameters}?

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

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

\textit{HiddenVector}: assignments of data points to \( k \) centers (\( n \)-dimensional vector with coordinates varying from 1 to \( k \)).
Coin Flipping and Soft Clustering

- **Coin flipping**: how would you select between coins $A$ and $B$ if $\Pr(\text{sequence}|\theta_A) = \Pr(\text{sequence}|\theta_B)$?

- **$k$-means clustering**: what cluster would you assign a data point it to if it is a midpoint of centers $C_1$ and $C_2$?

**Soft assignments**: assigning $C_1$ and $C_2$ “responsibility” $\approx 0.5$ for a midpoint.
From *Data & Parameters to HiddenVector*

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

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

$$\Pr(1^{st\ sequence}|\theta_A) = \theta_A^5(1-\theta_A)^5 = 0.60^4 \cdot 0.40^6 \approx 0.000531 >$$

$$\Pr(1^{st\ sequence}|\theta_B) = \theta_B^5(1-\theta_B)^5 = 0.82^4 \cdot 0.18^6 \approx 0.000015$$
Memory Flash:
From *Data & Parameters to HiddenVector*

\[
\begin{array}{ccc}
\text{Data} & \text{HiddenVector} & \text{Parameters} = (\theta_A, \theta_B) \\
0.4 & 1 & \\
0.9 & ? & \\
0.8 & ? & (0.60, 0.82) \\
0.3 & ? & \\
0.7 & ? & \\
\end{array}
\]

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

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

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

What are the responsibilities of coins for this sequence?

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

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

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

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

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

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

(0.60, 0.82)
Expectation Maximization Algorithm

Data

Hidden Matrix

Parameters
E-step

Data

HiddenMatrix

Parameters
M-step

Data

HiddenVector

Parameters'
### Memory Flash: Dot Product

Data | HiddenVector | Parameters = ($\theta_A$, $\theta_B$)
--- | --- | ---
HTTTHTTTHTH | 0.4 * 1 | 
HHHHHTHHHHH | 0.9 * 0 | 
HTHHHHHTHHH | 0.8 * 0 | 
HTTTTTTHHTT | 0.3 * 1 | 
TTHHHTHHHTH | 0.7 * 0 |

\[ \theta_A = Data \times HiddenVector \div 1 \times HiddenVector \]

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

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

<table>
<thead>
<tr>
<th>HiddenMatrix</th>
<th>Data</th>
<th>HiddenVector</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHTTHTTTTHH</td>
<td>0.4</td>
<td>1</td>
<td>( \theta_A ) = Data * HiddenVector / 1 * HiddenVector</td>
</tr>
<tr>
<td>HHHHTHHNHHH</td>
<td>0.9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>HTHHNHHHTHH</td>
<td>0.8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>HHTTTTHHTTTT</td>
<td>0.3</td>
<td>1</td>
<td>( \theta_B ) = Data * (1-HiddenVector) / 1 * (1-HiddenVector)</td>
</tr>
<tr>
<td>THHHTHNNHTH</td>
<td>0.7</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**HiddenVector** = \( (1 \ 0 \ 0 \ 1 \ 0) \)

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

Data | HiddenVector | Parameters = (θ_A, θ_B)

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenVector</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTHHTTHTH</td>
<td>0.4 1</td>
</tr>
<tr>
<td>HHHHTHHHHH</td>
<td>0.9 0</td>
</tr>
<tr>
<td>HTHHHHTHHH</td>
<td>0.8 0</td>
</tr>
<tr>
<td>HTHTTHHHTT</td>
<td>0.3 1</td>
</tr>
<tr>
<td>THHHTHHHHTH</td>
<td>0.7 0</td>
</tr>
</tbody>
</table>

θ_A = Data * 1\textsuperscript{st} row of HiddenMatrix / 1 * 1\textsuperscript{st} row of HiddenMatrix

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

θ_B = Data * 2\textsuperscript{nd} row of HiddenMatrix / 1 * 2\textsuperscript{nd} row of HiddenMatrix

HiddenVector = (1 0 0 1 0)

Hidden Matrix = 1 0 0 1 0 = HiddenVector

0 1 1 0 1 = 1 - HiddenVector
From *Data* & *HiddenMatrix* to *Parameters*

<table>
<thead>
<tr>
<th>Data</th>
<th>HiddenMatrix</th>
<th>Parameters = $(\theta_A, \theta_B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTHHTTHTTH</td>
<td>0.4 0.97 0.03</td>
<td></td>
</tr>
<tr>
<td>HHHHTHHHHH</td>
<td>0.9 0.12 0.88</td>
<td></td>
</tr>
<tr>
<td>HTHHHHTTHH</td>
<td>0.8 0.29 0.71</td>
<td></td>
</tr>
<tr>
<td>HHTTTTHHTT</td>
<td>0.3 0.99 0.01</td>
<td></td>
</tr>
<tr>
<td>TTHHHHTHHTH</td>
<td>0.7 0.55 0.45</td>
<td></td>
</tr>
</tbody>
</table>

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

*HiddenVector* = (1 0 0 1 0)

*Hidden Matrix* =

\[
\begin{pmatrix}
0.97 & 0.03 & 0.29 & 0.99 & 0.55 \\
0.03 & 0.97 & 0.71 & 0.01 & 0.45
\end{pmatrix}
\]
From \textit{HiddenVector} to \textit{HiddenMatrix}

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

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

\textbf{HiddenVector:} assignments of data points to centers

<table>
<thead>
<tr>
<th>$\text{HiddenVector}$</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{HiddenMatrix}$</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<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>
<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>
<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>
<td>1</td>
</tr>
</tbody>
</table>
From *HiddenVector* to *HiddenMatrix*

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

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

$HiddenMatrix_{i,j}$: responsibility of center $i$ for data point $j$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

- $A$ is associated with center 1.
- $B$ is associated with center 2.
- $C$ is associated with center 3.
- $D$ is associated with center 3.
- $E$ is associated with center 2.
- $F$ is associated with center 1.
- $G$ is associated with center 3.
- $H$ is associated with center 1.
From *HiddenVector* to *HiddenMatrix*

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

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

$HiddenMatrix_{i,j}$: responsibility of center $i$ for data point $j$

$$
\begin{array}{cccccccc}
A & B & C & D & E & F & G \\
H & 0.70 & 0.15 & 0.73 & 0.40 & 0.15 & 0.80 & 0.05 & 0.05 \\
2 & 0.20 & 0.80 & 0.17 & 0.20 & 0.80 & 0.10 & 0.05 & 0.20 \\
3 & 0.10 & 0.05 & 0.10 & 0.40 & 0.05 & 0.10 & 0.90 & 0.75 \\
\end{array}
$$
Responsibilities and the Law of Gravitation

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

$$ \text{Force}_{i,j} = \frac{1}{\text{distance}(Data_j, Center_i)^2} $$

$$ \text{HiddenMatrix}_{ij} = \frac{\text{Force}_{i,j}}{\sum_{\text{all centers } j} \text{Force}_{i,j}} $$

<table>
<thead>
<tr>
<th></th>
<th>0.70</th>
<th>0.15</th>
<th>0.73</th>
<th>0.40</th>
<th>0.15</th>
<th>0.80</th>
<th>0.05</th>
<th>0.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>stars</td>
<td>0.20</td>
<td>0.80</td>
<td>0.17</td>
<td>0.20</td>
<td>0.80</td>
<td>0.10</td>
<td>0.05</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>0.05</td>
<td>0.10</td>
<td>0.40</td>
<td>0.05</td>
<td>0.10</td>
<td>0.90</td>
<td>0.75</td>
</tr>
</tbody>
</table>
Responsibilities and Statistical Mechanics

Responsibility of center $i$ for a data point $j$ is proportional to

$$\text{Force}_{i,j} = e^{-\beta \cdot \text{distance}(Data_j, Center_i)}$$

where $\beta$ is a stiffness parameter.

$$\text{HiddenMatrix}_{ij}: = \frac{\text{Force}_{i,j}}{\sum_{\text{all centers } j} \text{Force}_{i,j}}$$
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.
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.
Constructing the 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>10.1</td>
<td>12.8</td>
<td>2.0</td>
<td>1.0</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.
Constructing the Tree

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

\[
\{g_3, g_5\}
\]

<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>2.3</td>
<td>5.1</td>
<td>10.2</td>
<td>6.1</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td>( g_2 )</td>
<td>8.1</td>
<td>0.0</td>
<td>12.0</td>
<td>0.9</td>
<td>9.5</td>
<td>10.1</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>11.1</td>
<td>8.1</td>
<td>1.0</td>
<td>10.5</td>
<td>11.5</td>
<td></td>
</tr>
<tr>
<td>( g_4 )</td>
<td>7.7</td>
<td>0.9</td>
<td>11.2</td>
<td>0.0</td>
<td>9.2</td>
<td>9.5</td>
<td>12.0</td>
<td>1.6</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>( g_5 )</td>
<td>2.3</td>
<td>9.5</td>
<td>11.1</td>
<td>9.2</td>
<td>0.0</td>
<td>5.6</td>
<td>12.1</td>
<td>7.7</td>
<td>8.5</td>
<td></td>
</tr>
<tr>
<td>( g_6 )</td>
<td>5.1</td>
<td>10.1</td>
<td>8.1</td>
<td>9.5</td>
<td>5.6</td>
<td>0.0</td>
<td>9.1</td>
<td>8.3</td>
<td>9.3</td>
<td></td>
</tr>
<tr>
<td>( g_7 )</td>
<td>10.2</td>
<td>12.8</td>
<td>1.0</td>
<td>12.0</td>
<td>12.1</td>
<td>9.1</td>
<td>0.0</td>
<td>11.4</td>
<td>12.4</td>
<td></td>
</tr>
<tr>
<td>( g_8 )</td>
<td>6.1</td>
<td>2.0</td>
<td>10.5</td>
<td>1.6</td>
<td>7.7</td>
<td>8.3</td>
<td>11.4</td>
<td>0.0</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>( g_9 )</td>
<td>7.0</td>
<td>1.0</td>
<td>11.5</td>
<td>1.1</td>
<td>8.5</td>
<td>9.3</td>
<td>12.4</td>
<td>1.1</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>
Constructing the Tree

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

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

\[
\begin{align*}
g_1 & \quad 0.0 & 7.7 & 9.2 & 2.3 & 5.1 & 10.2 & 6.1 & 7.0 \\
g_2 & \quad 7.7 & 0.0 & 11.2 & 9.2 & 9.5 & 12.0 & 1.6 & 1.0 \\
g_3 & \quad 9.2 & 11.2 & 0.0 & 11.1 & 8.1 & 1.0 & 10.5 & 11.5 \\
g_4 & \quad 2.3 & 9.2 & 11.1 & 0.0 & 5.6 & 12.1 & 7.7 & 8.5 \\
g_5 & \quad 5.1 & 9.5 & 8.1 & 5.6 & 0.0 & 9.1 & 8.3 & 9.3 \\
g_6 & \quad 10.2 & 12.0 & 1.0 & 12.1 & 9.1 & 0.0 & 11.4 & 12.4 \\
g_7 & \quad 6.1 & 1.6 & 10.5 & 7.7 & 8.3 & 11.4 & 0.0 & 1.1 \\
g_8 & \quad 7.0 & 1.0 & 11.5 & 8.5 & 9.3 & 12.4 & 1.1 & 0.0
\end{align*}
\]
Identify the two closest clusters and merge them.

Constructing the Tree

<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>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>$g_2, g_4$</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>$g_3, g_5, g_8$</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>$g_6$</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>$g_7$</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>$g_8$</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>$g_9$</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>$g_{10}$</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

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

**return** $T$
Different Distance Functions Result in Different Trees

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

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

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

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

Cluster 1

Surge in expression at final checkpoint

Cluster 2

Cluster 3

Cluster 4

Cluster 5

Cluster 6
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 \( r \): \( M_{pq} = \frac{(M_{pq})'}{\sum_i (M_{iq})'} \)
6. Repeat steps 4 and 5 until a steady state is reached (convergence).
Markov Clustering Algorithm
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

$M[i][j] \leftarrow \sum_{k \in V} \frac{M[i][j]}{M[k][k]}$

$H \leftarrow$ 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.
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-tse/](http://distill.pub/2016/misread-tse/) 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.
Stochastic Neighbor Embedding

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

$$p_{j|i} = \frac{\exp\left(\frac{-||x_i - x_j||^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{-||x_i - x_j||^2}{2\sigma_i^2}\right)}.$$
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.
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.

**Equation 2**

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

**Equation 3**

$$\frac{\delta C}{\delta y} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}$$
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: \( X = x_1, x_2, \ldots, x_n \),

cost function parameters: perplexity \( \text{Perp} \);

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

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

begin

compute pairwise affinities \( p_{ji} \) with perplexity \( \text{Perp} \) (Equation 1)

set \( p_{ij} = \frac{p_{ji} + p_{ij}}{2n} \);

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

for \( t = 1 \) to \( T \) do

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

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

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

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/
A lot of genomic and medical data coming

- U.K. 5M from NHS (https://www.genomicsengland.co.uk)
- AllOfUs – 1M U.S. Patients with medical data
- Netherlands GoNL– 250trios – preclinical (http://www.nlgenome.nl/)
- Faroe islands 100k – pre-clinical
- Qatar 300k – pre-clinical
- UK 100k – clinical
- Genomics Medicine Ireland (GMI) with AbbVie – 45k w/clinical
- Iceland 2.5k – pre-clinical
- Poland 100K
- Swiss Genome 100K
- Geisinger Health 100K (with Regeneron)
- Astrozenica (2M with HLI)
- 1 million U.S. Veterans Project
- Newfoundland 100K
New Chinese Sequencer Promises 60 Human Genomes In A Day

By Bio-IT World Staff

October 25, 2018 | MGI Tech (part of BGI), introduced the next iteration of its sequencer, the MGISEQ-T7, at the 13th International Conference on Genomics (ICG-13) in Shenzhen. The company also announced new library and sample prep and an application for tumor mutation detection.

The proprietary MGI technology used in T7 delivers higher accuracy and improves efficiency through upgrades to the flowcell, fluid, and biochemical and optical system. The new sequencer delivers quadruple flowcell staging that allows simultaneous but independent operation of 1 to 4 flowcells in a single run.

The platform supports whole genome sequencing, ultra-depth exome sequencing, epigenome sequencing, and large-panel tumor gene detection, and has a daily data output capacity of up to 6TB. The company reports that MGISEQ-T7 can complete whole genome sequencing for up to 60 human genomes in a single day.

"MGI is developing at an unprecedented speed, and our technology has advanced to the leading market," said MGI CEO Feng Mu in a statement. "This new instrument demonstrates the level of MGI's innovation and commitment to progress: we continually challenge ourselves to make sequencing more affordable to biologists.

The MGISEQ-T7 includes:

- Single nucleotide polymorphisms (SNPs)
  - 1 every few hundred bp, mutation rate $* \approx 10^{-9}$

- Short indels (= insertion/deletion)
  - 1 every few kb, mutation rate $v. \text{variable}$

- 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

---

TGCATTGGGAC
TGCATTCCGAGG
TGCTATCTAATATCACGC
TGCTATCAGC---GC

$\leq 100bp$

1-5kb
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.
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).
To recreate $T$ from $\text{BWT}(T)$, repeatedly apply the rule: $T = \text{BWT}[\text{LF}(i)] + T$; $i = \text{LF}(i)$ where $\text{LF}(i)$ maps row $i$ to row whose first character corresponds to $i$’s last per LF Mapping. First step: $S = 2; T = \$. Second step: $s = \text{LF}[2] = 6; T = g\$. Third step: $s = \text{LF}[6] = 5; T = cg\$. 

\[ \begin{array}{ccccccc}
& g & c & g & a & c & g & a \\
\$ & a & c & a & a & c & g & g \\
a & a & c & g & a & c & g & a \\
a & a & c & g & g & a & c & a \\
a & a & c & g & a & c & g & $ \\
c & a & c & g & g & a & a & a \\
c & g & g & a & c & a & a & $ \\
g & g & a & c & a & c & g & $ \\
g & g & a & c & a & c & g & $ \\
\end{array} \]
Burrows-Wheeler Transform (BWT)

acaacg$

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

gc$aaac

Burrows-Wheeler Matrix (BWM)
Burrows-Wheeler Matrix

$\text{acaacg}$

$\text{aacg}$

$\text{acg}$

$\text{caacg}$

$\text{cg}$

$\text{g}$

$\text{acaacg}$

$\text{acg}$

$\text{caacg}$

$\text{cg}$

$\text{g}$
Burrows-Wheeler Matrix

\[
\begin{array}{cccc}
3 & aacg & ac & \\
1 & acaacg & $ & \\
4 & acg & aca & \\
2 & caacg & a & \\
5 & cg & acaa & \\
6 & g & acaac & \\
\end{array}
\]

See the suffix array?
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) $a c a a c g$
   $a a c g s a c$
   $a c a a c g$

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

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

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

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

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

   $a a c g s a c$
   $a c g s a c a$
   $c a a c g$
   $c g s a c a a$
   $g s a c a a c$
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?

CTGATGATGGACTACGCTACTACTGC TAGCTGTA

CTGAGGATGGACTACGCTACTACTGA TAGCTGTT

Individual

Reference
Why Not Use Assembly?

Multiple copies of a genome

Shatter the genome into reads

Sequence the reads

Assemble the genome with overlapping reads

AGAATATCA

TGAGAATAT

GAGAATATC

AGAATATCA

GAGAATATC

TGAGAATAT

...TGAGAATATCA...
Why Not Use Assembly?

• Constructing a de Bruijn graph takes a lot of memory.

• Hope: a machine in a clinic that would collect and map reads in 10 minutes.

• Idea: use existing structure of reference genome to help us sequence a patient’s genome.
Read Mapping

• **Read mapping**: determine where each read has high similarity to the reference genome.

CTGAGGATGGACTACGCTACTACTACTGATAGCTGTGTTT        Reference
GAGGA       CCACG               TGA–A        Reads

532
Why Not Use Alignment?

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

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

- Has runtime $O(|Patterns| \times |Genome|)$ for a collection of *Patterns*. 
Exact Pattern Matching

• Focus on a simple question: where do the reads match the reference genome exactly?

• Single Pattern Matching Problem:
  – **Input:** A string *Pattern* and a string *Genome*.
  – **Output:** All positions in *Genome* where *Pattern* appears as a substring.
Exact Pattern Matching

• Focus on a simple question: where do the reads match the reference genome *exactly*?

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

• We can simply iterate a brute force approach method, sliding each *Pattern* down *Genome*.

```
panamabananas
```

• *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: \( O(|Genome| \times |Patterns|) \)
  – \( |Patterns| = \) 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
panamabanananas
Success!

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

• Runtime of Trie Matching:
  – Trie Construction: $O(|Patterns|)$
  – Pattern Matching: $O(|Genome| * |LongestPattern|)$
Memory Analysis of TrieMatching

• Son completely forgot about memory!

• Our trie: 30 edges, $|\text{Patterns}| = 39$

• Worst case: # edges
  $= O(|\text{Patterns}|)$
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.
panamabanas

$\text{Root}$

$\text{panamabanas}$
Memory Trouble Once Again

- Worst case: the suffix trie holds $O(|Suffixes|)$ nodes.

- For a *Genome* of length $n$,
  $|Suffixes| = n(n - 1)/2 = O(n^2)$
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: $O(|Genome| + |Patterns|)$
  • Memory: $O(|Genome|)$

• However, big-O notation ignores constants!
  • The best known suffix tree implementations require $\sim 20$ times the length of $|Genome|$.
  • Can we reduce this constant factor?
Genome Compression

• Idea: decrease the amount of memory required to hold *Genome*.

• This indicates that we need methods of **compressing** a large genome, which is seemingly a separate problem.
Idea #1: Run-Length Encoding

• **Run-length encoding**: compresses a run of \( n \) identical symbols.

```
Genome
GGGGGGGGGGCCCCCCCCCCCCCAAAAAAAATTTTTTTTTTTTTTTTTTCCCCCG
```

• **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? → Convert repeats to runs

Genome → Genome*

Run-length encoding

CompressedGenome*
Form all cyclic rotations of “panamabanananas$”

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

Form all cyclic rotations of "panamabanananas$"

Burrows-Wheeler Transform:
Last column = smnbpbnnaaaaa$a
BWT: Converting Repeats to Runs

Genome

Burrows-Wheeler Transform! Convert repeats to runs

BWT(Genome)

Run-length encoding

Compression(BWT(Genome))
How Can We Decompress?

Genome

IS IT POSSIBLE?  ↓  Burrows-Wheeler Transform

BWT(Genome)

EASY  ↓  Run-length encoding

Compression(BWT(Genome))
Reconstructing banana

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

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

$banana$
$a$bana
ana$bana
ana$banana
banana$

3-mers

$ba$
$a$b
$ba$
$na$
$na$

Sort

banana
ana
na$;
ana
ana
ana

• We now know 3-mer composition of the circular string banana$

• Sorting gives us the first 3 columns of the matrix.
Reconstructing banana

$banana$

• We now know 4-mer composition of the circular string banana$

• Sorting gives us the first 4 columns of the matrix.
Reconstructing *banana*

<table>
<thead>
<tr>
<th>5-mers</th>
<th>Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>$banana$</td>
<td>$banana$</td>
</tr>
<tr>
<td>a$banana</td>
<td>a$bbn</td>
</tr>
<tr>
<td>ana$ban</td>
<td>anaab</td>
</tr>
<tr>
<td>anana$b</td>
<td>anaaa</td>
</tr>
<tr>
<td>banana$</td>
<td>bannn</td>
</tr>
<tr>
<td>na$banana</td>
<td>na$ba</td>
</tr>
<tr>
<td>nana$ba</td>
<td>nana$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>6-mers</th>
<th>Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{banana}$</td>
<td>$\text{banana}$</td>
</tr>
<tr>
<td>$\text{a$banana}$</td>
<td>$\text{anaaba}$</td>
</tr>
<tr>
<td>$\text{ana$ban}$</td>
<td>$\text{anaab}$</td>
</tr>
<tr>
<td>$\text{ananana$b}$</td>
<td>$\text{banan}$</td>
</tr>
<tr>
<td>$\text{bananana$}$</td>
<td>$\text{banan}$</td>
</tr>
<tr>
<td>$\text{nanana$ba}$</td>
<td>$\text{nanana}$</td>
</tr>
<tr>
<td>$\text{nanana$ba}$</td>
<td>$\text{nanana}$</td>
</tr>
<tr>
<td>$\text{nanana$ba}$</td>
<td>$\text{nanana}$</td>
</tr>
</tbody>
</table>

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

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

6-mers

Sort

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

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

\[ \begin{align*}
\$ & \text{banana} \\
\text{a} & \$ \text{banana} \\
\text{a} & \text{ana} \$ \text{ban} \\
\text{ana} & \$ \text{ban} \\
\text{ban} & \text{ana} \$ \\
\text{ana} & \$ \text{bana} \\
\text{na} & \$ \text{bana} \\
\text{nana} & \$ \text{ba} \\
\end{align*} \]

• We now know the entire matrix!

• Taking all elements in the first row (after \$) produces banana.
More Memory Issues

• Reconstructing *Genome* from $BWT(Genome)$ required us to store $|Genome|$ copies of $|Genome|$.

$$
\begin{align*}
  &\: banana \\
  a\: &\: banan \\
  an\: a\: &\: ban \\
  anan\: a\: &\: b \\
  banana\: &\: \\
  nann\: a\: &\: ba \\
  \end{align*}
$$

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

$panamabananas$
$abananas$panam
$amabananas$pan
$anamabananas$p
$ananas$panamab
$anas$panamaban
$as$panamaban
$bananas$panama
$abananas$pana
$namabananas$pa
$nanas$panamaba
$nas$panamaban
$panamabananas$
$s$panamabanana
Is It True in General?

These strings are sorted

$panamabananas$s $panamabananas$pan $namabanananas$p $nanas$panamab $nas$panamaban $s$panamabanana

Chop off a

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

bananas$panama
mabanananas$pan
namabanananas$p
nanas$panamab
nas$panamaban
s$panamabanan
Is It True in General?

These strings are sorted
Is It True in General?

These strings are sorted:

<table>
<thead>
<tr>
<th></th>
<th>$\text{panamabananas}$</th>
<th>$\text{bananas}$</th>
<th>$\text{panamabananas}$</th>
<th>$\text{mabananas}$</th>
<th>$\text{pananamabananas}$</th>
<th>$\text{nas}$</th>
<th>$\text{panamabanana}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\text{a}$</td>
<td>$\text{bananas}$</td>
<td>$\text{panam}$</td>
<td>$\text{mabananas}$</td>
<td>$\text{pananamabananas}$</td>
<td>$\text{nas}$</td>
<td>$\text{panamabanana}$</td>
</tr>
<tr>
<td>2</td>
<td>$\text{amabananas}$</td>
<td>$\text{pan}$</td>
<td>$\text{nanas}$</td>
<td>$\text{panamab}$</td>
<td>$\text{pananas}$</td>
<td>$\text{panamaban}$</td>
<td>$\text{s}$</td>
</tr>
<tr>
<td>3</td>
<td>$\text{anamabananas}$</td>
<td>$\text{p}$</td>
<td>$\text{nanas}$</td>
<td>$\text{panamab}$</td>
<td>$\text{pananas}$</td>
<td>$\text{panamaban}$</td>
<td>$\text{s}$</td>
</tr>
<tr>
<td>4</td>
<td>$\text{ananas}$</td>
<td>$\text{panamab}$</td>
<td>$\text{nanan}$</td>
<td>$\text{panamaban}$</td>
<td>$\text{pananas}$</td>
<td>$\text{panamaban}$</td>
<td>$\text{s}$</td>
</tr>
<tr>
<td>5</td>
<td>$\text{anpas}$</td>
<td>$\text{panamabana}$</td>
<td>$\text{nananas}$</td>
<td>$\text{panamaban}$</td>
<td>$\text{pananas}$</td>
<td>$\text{panamaban}$</td>
<td>$\text{s}$</td>
</tr>
<tr>
<td>6</td>
<td>$\text{as}$</td>
<td>$\text{panamabanana}$</td>
<td>$\text{bananas}$</td>
<td>$\text{panam}$</td>
<td>$\text{mabananas}$</td>
<td>$\text{pananamabananas}$</td>
<td>$\text{nas}$</td>
</tr>
</tbody>
</table>

Chop off $\text{a}$:

- Still sorted:
  - $\text{bananas}$
  - $\text{panamabananas}$
  - $\text{mabananas}$
  - $\text{pananamabananas}$
  - $\text{nas}$
  - $\text{panamabanana}$

Add $\text{a}$ to end:

- Ordering doesn’t change!
  - $\text{bananas}$
  - $\text{panamabananas}$
  - $\text{mabananas}$
  - $\text{pananamabananas}$
  - $\text{nas}$
  - $\text{panamabanana}$

These strings are still sorted:

- $\text{bananas}$
- $\text{panamabananas}$
- $\text{mabananas}$
- $\text{pananamabananas}$
- $\text{nas}$
- $\text{panamabanana}$
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
More Efficient BWT Decompression

$1\text{panamabananas}s_1$
$a_1\text{bananas}$|$panam_1$
$a_2\text{mabananas}$|$pan_1$
$a_3\text{namabananas}$|$p_1$
$a_4\text{nananas}$|$panamaba_1$
$a_5\text{nas}$|$panamaban_2$
$a_6\text{s}$|$panamabanana_3$
$b_1\text{ananas}$|$panama_1$
$m_1\text{abanananas}$|$pana_2$
$n_1\text{amabananas}$|$pa_3$
$n_2\text{ananas}$|$panamaba_4$
$n_3\text{as}$|$panamaban_5$
$p_1\text{anamabananas}$|$s_1$
$s_1\text{spanamabanana}$a_6$

579
More Efficient BWT Decompression

- Memory: $2|\text{Genome}| = O(|\text{Genome}|)$. 

Diagram: 

$ s_1 \text{panamabananas}_1 $  
$a_1 \text{bananaspam}_1 $  
$a_2 \text{mabananas}_1 \text{pam}_1 $  
$a_3 \text{namabananas}_1 \text{p}_1 $  
$a_4 \text{nanas}_1 \text{panamab}_1 $  
$a_5 \text{nas}_1 \text{panamaban}_2 $  
$a_6 \text{s}_1 \text{panamabanana}_3 $  
$b_1 \text{ananas}_1 \text{panama}_1 $  
$m_1 \text{abananaspam}_2 $  
$n_1 \text{amabananas}_1 \text{pana}_3 $  
$n_2 \text{ananaspamaba}_4 $  
$n_3 \text{as}_1 \text{panamaban}_5 $  
$p_1 \text{anamabananas}_1 \text{p}_1 $  
$s_1 \text{spanamabanana}_6 $
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 $\text{BWT}(Genome)$ as our data structure instead?
Finding Pattern Matches Using BWT

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

$1_{panamabanananas}$
$a1_{bananasonam}$
$a2_{mabananas}pan1$
$a3_{namabananas}p1$
$a4_{nas}panamab1$
$a5_{nas}panamaban2$
$a6_{}pamabanan3$
$b1_{ananasa}panama1$
$m1_{abananas}pana2$
$n1_{amabananas}p3$
$n2_{anas}panamaba4$
$n3_{as}panamaban5$
p1_{anamabananas}$
s1_{panamabanaa}$
Finding Pattern Matches Using BWT

• Searching for ana in panamabanananas

$\_1$p_\text{anamabanananas}\_1
a_1$b_\text{ananas}$$p_\text{anam}\_1
a_2$m_\text{abanananas}$$p_\text{ana}\_1
a_3$n_\text{amabananas}$$p\_1
a_4$n_\text{anas}$$p_\text{anamaba}\_1
a_5$n_\text{as}$$p_\text{anamaban}\_2
a_6$s$$\text{panamabananana}\_3
b_1$a_\text{nananas}$$p_\text{ama}\_1
m_1$a_\text{bananas}$$p_\text{ana}\_2
n_1$m_\text{abanananas}$$p_\text{a}\_3
n_2$n_\text{anas}$$p_\text{anamaba}\_4
n_3$a_\text{nas}$$p_\text{anamabana}\_5
p_1$p_\text{anamabanananas}\_1
s_1$s$$\text{panamabananana}\_6
Finding Pattern Matches Using BWT

- Searching for **`ana`** in **`panamabanananas`**
Finding Pattern Matches Using BWT

• Searching for **ana** in **panamabanananas**
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 \texttt{ana} occurs 3 times, but where?

\begin{verbatim}
$1 \text{panamabananas}$
$ a_1 \text{bananas} \text{panam}$
$ a_2 \text{mabanananas} \text{pan}$
$ a_3 \text{na} \text{mabanananas} \text{pan}$
$ a_4 \text{nanas} \text{panamanab}$
$ a_5 \text{na} \text{as} \text{panamaban}$
$ a_6 \text{as} \text{panamabana}$
$ b_1 \text{anananas} \text{panama}$
$ m_1 \text{abananas} \text{pana}$
$ n_1 \text{amabananas} \text{pana}$
$ n_2 \text{anas} \text{panamaba}$
$ n_3 \text{as} \text{panamaban}$
$ p_1 \text{ananabananas}$
$ s_1 \text{panamabanan}$
\end{verbatim}
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.

```plaintext
panamabananas$
```

1 3

<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>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>p</td>
<td>p</td>
<td>p</td>
<td>p</td>
<td>p</td>
<td>p</td>
<td>p</td>
</tr>
<tr>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
</tbody>
</table>

```plaintext
1 3
panamabananas$_1$
bananas$_2$
mabananas$_3$
amabananas$_4$
ana$_5$
as$_6$

panamabanana$$_1$
bananas$$_2$
mabananas$$_3$
amabananas$$_4$
ana$$_5$
as$$_6$

589
Using the Suffix Array to Find Matches

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

```plaintext
panamabanananas$
```

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1$</td>
<td>panamabanananas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1$</td>
<td>bananas$panam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_2$</td>
<td>mabananas$pan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_3$</td>
<td>namabanananas$p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_4$</td>
<td>nanas$panamab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_5$</td>
<td>nas$panamaban</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_6$</td>
<td>s$panamabanana</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_1$</td>
<td>ananas$panama</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$m_1$</td>
<td>abanananas$pana</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_1$</td>
<td>amabananas$pa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_2$</td>
<td>as$panamabana</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_3$</td>
<td>$panamabana</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_1$</td>
<td>anamabanananas$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_1$</td>
<td>$panamabanana</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.

```plaintext
panamabanananas$
```

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1_{panamabananas}$</td>
<td>$a_1_{bananas}$</td>
<td>$a_2_{mabanananas}$</td>
<td>$a_3_{nanabananas}$</td>
</tr>
<tr>
<td>$a_4_{anas}$</td>
<td>$a_5_{nas}$</td>
<td>$a_6_{s}$</td>
<td>$b_1_{ananas}$</td>
</tr>
<tr>
<td>$b_1_{bananas}$</td>
<td>$m_1_{ababananas}$</td>
<td>$n_1_{amabananas}$</td>
<td>$n_2_{anas}$</td>
</tr>
<tr>
<td>$n_3_{as}$</td>
<td>$p_1_{anamananas}$</td>
<td>$s_1_{$panamabanana}$</td>
<td>$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>1</th>
<th>3</th>
<th>$p_1$panamabanananas$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>a$1$bananass$panam$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a$2$banananas$panam$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a$3$namabanananas$panam$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a$4$annas$panamab</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a$5$nas$panamaban</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a$6$s$panamabanana</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b$1$anananas$panama</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m$1$anananas$panamab</td>
<td></td>
</tr>
<tr>
<td></td>
<td>n$1$amabanananas$pana</td>
<td></td>
</tr>
<tr>
<td></td>
<td>n$2$anas$panamaba</td>
<td></td>
</tr>
<tr>
<td></td>
<td>n$3$as$panamabana</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>p$1$ananabana$panamab</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>s$1$panamabana$panamab</td>
<td></td>
</tr>
</tbody>
</table>
```
Using the Suffix Array to Find Matches

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

\[
\begin{array}{|c|}
\hline
1 & 1 \\
2 & 5 \\
3 & 3 \\
4 & 1 \\
5 & 7 \\
6 & 13 \\
7 & \\
\hline
\end{array}
\]

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$

$panamabanananas$
Using the Suffix Array to Find Matches

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$1$</td>
<td>panamabananas</td>
<td>$a_1$</td>
<td>bananas</td>
<td>$a_2$</td>
</tr>
<tr>
<td></td>
<td>panamabananas</td>
<td>$a_3$</td>
<td>mabananas</td>
<td>$a_4$</td>
<td>nanas</td>
</tr>
<tr>
<td></td>
<td>$a_5$</td>
<td>nas</td>
<td>bananas</td>
<td>panamab</td>
<td>$a_6$</td>
</tr>
<tr>
<td></td>
<td>$b_1$</td>
<td>ananas</td>
<td>bananas</td>
<td>panama</td>
<td>$m_1$</td>
</tr>
<tr>
<td></td>
<td>nanas</td>
<td>$n_1$</td>
<td>panamabanana</td>
<td>$n_2$</td>
<td>$p_1$</td>
</tr>
<tr>
<td></td>
<td>$n_3$</td>
<td>as</td>
<td>panamabana</td>
<td>$p_1$</td>
<td>$s_1$</td>
</tr>
<tr>
<td></td>
<td>$s_1$</td>
<td>$s$</td>
<td>$s$</td>
<td>panamabanana</td>
<td>6</td>
</tr>
</tbody>
</table>
Using the Suffix Array to Find Matches

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

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 13 | $1_{panamabananas}$ | 5 | $a_1_{bananas}$ | 3 | $a_2_{mabanananas}$ | 1 | $a_3_{namabanananas}$ | 7 | $a_4_{nananas}$ | 9 | $a_5_{nas}$ | 11 | $a_6_{s}$ | panamabananas |
| 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  |
| 5  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  |
| 3  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  |
| 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  |
| 7  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  |
| 9  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  |
| 11 | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  | panamabananas | 1  |
Using the Suffix Array to Find Matches

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

```
panamabananasa$  
$ 1     panamabananasa
  a 1     bananas$panama
  a 2     mabananas$panam
  a 3     namabananasa$p
  a 4     nanas$panamab
  a 5     nas$panamaban
  a 6     s$panamabanana
  b 1    ananasa$panama
  m 1     abananasa$pan
  n 1     amabananasa$pa
  n 2     anas$panamaba
  n 3     as$panamaban
  p 1     anamabananas$1
  s 1     $panamabana
```
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 Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>$1panamabananas$</td>
</tr>
<tr>
<td>5</td>
<td>$a1bananas$panam</td>
</tr>
<tr>
<td>3</td>
<td>$a2mabananas$pan</td>
</tr>
<tr>
<td>1</td>
<td>$a3namabanananas$p</td>
</tr>
<tr>
<td>7</td>
<td>$a4nanas$panamab</td>
</tr>
<tr>
<td>9</td>
<td>$a5nas$panamaban</td>
</tr>
<tr>
<td>11</td>
<td>$a6s$panamabanana</td>
</tr>
<tr>
<td>6</td>
<td>$b1ananas$panama</td>
</tr>
<tr>
<td>4</td>
<td>$m1abananas$pana</td>
</tr>
<tr>
<td>2</td>
<td>$n1amabanananas$pa</td>
</tr>
<tr>
<td>8</td>
<td>$n2anas$panamaba</td>
</tr>
<tr>
<td>10</td>
<td>$n3as$panamabana</td>
</tr>
<tr>
<td>1</td>
<td>$p1anamabananas$</td>
</tr>
<tr>
<td>1</td>
<td>$s1$panamabana</td>
</tr>
</tbody>
</table>

597
Using the Suffix Array to Find Matches

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

```
panamabanan
```

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>$1panamabanananas 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>a 1bananass$panam 1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a 2mabananas$pan 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a 3namabananas$pa 1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>a 4nanas$panamab 1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>a 5nas$panamaban 2</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>a 6s$panamabanana 3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>b 1ananas$panama 1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>m 1abananas$pana 2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>n 1amabananas$pa 3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>n 2anas$panamaba 4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>n 3as$panamabana 5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>p 1anamabanananas$ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s 1$panamabana 6</td>
<td></td>
</tr>
</tbody>
</table>
Using the Suffix Array to Find Matches

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

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$panamabanananas$</td>
<td>$a_1$bananas$panam$</td>
<td>$a_2$mabananas$pan$</td>
<td>$a_3$namabananas$pan$</td>
<td>$a_4$ananas$panamaban$</td>
<td>$a_5$nas$panamaban$</td>
<td>$a_6$s$panamabanana$</td>
<td>$b_1$ananasa$panama$</td>
<td>$m_1$abananas$pana$</td>
<td>$n_1$amabananas$pana$</td>
<td>$n_2$ananas$panamaba$</td>
<td>$n_3$as$panamabana$</td>
<td>$p_1$ananama$panama$</td>
</tr>
</tbody>
</table>
```

panamabanananas$
Using the Suffix Array to Find Matches

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

```
panamabanananas$
```

|   | $1_{panamabanananas}$ | $a_{1_{bananas}}$ | $a_{2_{mabananas}}$ | $a_{3_{namabananas}}$ | $a_{4_{nanas}}$ | $a_{5_{nas}}$ | $a_{6_{s}}$ | $b_{1_{ananas}}$ | $m_{1_{abananas}}$ | $n_{1_{amabananas}}$ | $n_{2_{anas}}$ | $n_{3_{as}}$ | $p_{1_{anamabananas}}$ | $s_{1_{$panamabanananas}}$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</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>11</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>10</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>9</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>8</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>7</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>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>
</tr>
<tr>
<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>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>
<td></td>
<td></td>
</tr>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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$`.  

|   | $1_{panamabanananas}$ | 5 | $a_1_{banan}as$ | $\text{panama}_1$ | 3 | $a_2_{mabana}nas$ | $\text{panama}_1$ | 1 | $a_3_{na}mabananas$ | $\text{panama}_1$ | 7 | $a_4_{na}nas$ | $\text{panama}_1$ | 9 | $a_5_{na}as$ | $\text{panama}_2$ | 11 | $a_6_{s}$ | $\text{panamabanana}_3$ | 6 | $b_1_{anan}as$ | $\text{panama}_1$ | 4 | $m_1_{aban}anas$ | $\text{panama}_2$ | 2 | $n_1_{amaban}anas$ | $\text{pa}_3$ | 8 | $n_2_{an}as$ | $\text{panama}_4$ | 10 | $n_3_{as}$ | $\text{panama}_5$ | 0 | $p_1_{anamaban}anas$ | $\text{s}_1$ | 12 | $s_1_{}$ | $\text{panamabana}_6$ |
The Suffix Array: Memory Once Again

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

![Suffix Array Diagram]

[13 5 3 1 7 9 11 6 4 2 8 10 0 1]
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: $\sim 4 \times |\text{Genome}|$. 

```
[13 5 3 1 7 9 11 6 4 2 8 10 0 1]
```
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:

```
Pattern       acttggct
Genome ...ggcactagggctc...
```
Method 1: Seeding

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

```
Pattern      acttggtc
Genome      ...ggcacactaggctcc...
```

• One of the substrings must match!
Method 1: Seeding

• **Theorem:** If *Pattern* occurs in *Genome* with *d* mismatches, then we can divide *Pattern* into *d* + 1 “equal” pieces and find at least one exact match.
Method 1: Seeding

• Say we are looking for at most $d$ mismatches.

• Divide each of our strings into $d + 1$ smaller pieces, called seeds.

• Check if each Pattern has a seed that matches Genome exactly.

• If so, check the entire Pattern against Genome.
Method 2: BWT Saves the Day Again

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

Now we extend all strings with at most 1 mismatch.

<table>
<thead>
<tr>
<th></th>
<th># Mismatches</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1panymabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>$a_1bananas$panam$</td>
<td>0</td>
</tr>
<tr>
<td>$a_2mabanananas$pan$</td>
<td>1</td>
</tr>
<tr>
<td>$a_3namabanananas$pan$</td>
<td>1</td>
</tr>
<tr>
<td>$a_4ananas$panamab$</td>
<td>0</td>
</tr>
<tr>
<td>$a_5nas$panamabana$</td>
<td>0</td>
</tr>
<tr>
<td>$a_6s$panamabanan$</td>
<td>0</td>
</tr>
<tr>
<td>$b_1ananas$panama$</td>
<td>1</td>
</tr>
<tr>
<td>$m_1abanananas$pana$</td>
<td>1</td>
</tr>
<tr>
<td>$n_1amabanananas$pan$</td>
<td>0</td>
</tr>
<tr>
<td>$n_2ananas$panamaba$</td>
<td>0</td>
</tr>
<tr>
<td>$n_3as$panamabana$</td>
<td>0</td>
</tr>
<tr>
<td>$p_1anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>$s_1$panamabananana$</td>
<td>0</td>
</tr>
</tbody>
</table>
Method 2: BWT Saves the Day Again

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

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

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

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

<table>
<thead>
<tr>
<th>3-mer</th>
<th># Mismatches</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1\text{panamabanananas}$</td>
<td>1</td>
</tr>
<tr>
<td>$a_1\text{bananas}$\text{panam}_1$</td>
<td>1</td>
</tr>
<tr>
<td>$a_2\text{mabananas}$\text{pan}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$a_3\text{mabananas}$\text{p}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$a_4\text{bananas}$\text{panamab}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$a_5\text{nas}$\text{panamab}_2$</td>
<td>0</td>
</tr>
<tr>
<td>$a_6$\text{panamabanan}_3$</td>
<td>0</td>
</tr>
<tr>
<td>$b_1\text{ananas}$\text{panama}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$m_1\text{abananas}$\text{pana}_2$</td>
<td>0</td>
</tr>
<tr>
<td>$n_1\text{amabananas}$\text{p}_3$</td>
<td>0</td>
</tr>
<tr>
<td>$n_2\text{anas}$\text{panamab}_4$</td>
<td>0</td>
</tr>
<tr>
<td>$n_3\text{as}$\text{panamaban}_5$</td>
<td>0</td>
</tr>
<tr>
<td>$p_1\text{anamabananas}$\text{p}_1$</td>
<td>0</td>
</tr>
<tr>
<td>$s_1$\text{panamabanana}_6$</td>
<td>0</td>
</tr>
</tbody>
</table>
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>
</tr>
</thead>
<tbody>
<tr>
<td>$1^{panamabanananas}$</td>
</tr>
<tr>
<td>a1bananas$panam$_1</td>
</tr>
<tr>
<td>a2mabanananas$panam$_1</td>
</tr>
<tr>
<td>a3namabanananas$pana$_1</td>
</tr>
<tr>
<td>a4ana$panamabananas$panamab$_1</td>
</tr>
<tr>
<td>a5nas$panamabanana$_2</td>
</tr>
<tr>
<td>a6s$panamabananan$_3</td>
</tr>
<tr>
<td>b1anananas$panamabana$_1</td>
</tr>
<tr>
<td>m1abanananas$pana$_2</td>
</tr>
<tr>
<td>n1amabanananas$pana$_3</td>
</tr>
<tr>
<td>n2anas$panamabana$_4</td>
</tr>
<tr>
<td>n3nas$panamabana$_5</td>
</tr>
<tr>
<td>p1ananamabananas$panamabananas$$_1</td>
</tr>
<tr>
<td>s1$panamabananan$a$_6</td>
</tr>
</tbody>
</table>
Method 2: BWT Saves the Day Again

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

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

Reference

BWT( Reference )

Query:
AATGATACGGCGACCACCGAGATCTA
More Burrows-Wheeler

Reference

BWT(Reference)

Query:
AATGATACGGCGACCAACCAGATCTA
More Burrows-Wheeler

Reference

BWT(Reference)

Query:
AATGATACGGCGACCACCGAGATCTA
More Burrows-Wheeler

Reference

BWT(Reference)

Query:
AATGATACGGCGACCCAGATCTA
More Burrows-Wheeler

Reference

BWT(Reference)

Query:
AATGATACGGCGACCAACCGAGATCTA
More Burrows-Wheeler

Reference

BWT(Reference)

Query:
AATGAGACGGCGACCACCGAGATCTA
More Burrows-Wheeeler

Reference

BWT(Reference)

Query:
AATGATACGGCGACCACCGAGATCTA
More Burrows-Wheeler

Reference

BWT(Reference)

Query:
AATG\textcolor{cyan}{TACGGCGACCAACCGAGATCTA}
More Burrows-Wheeler

Reference

BWT(Reference)

Query:

AATGT TACGGCGACCCACCGAGATCTA
Burrows Wheeler Transform (BWT)

Example:

11
012345678901
S = agcagcagac $ where the end of sequence pseudo-symbol, $, is less than all proper symbols.

$ \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>$$agcagcagac$</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>g act$$agcagcag</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>c agact$$agcagcag</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>c agcagacag act$</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>$ agcagcagcag $</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>g cagact$$agcagcag</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>g cagcagacag act$</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>a ct$$agcagcag</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>a gact$$agcagcag</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>a gcagcagacag act$</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>a gcagcagacag act$</td>
</tr>
</tbody>
</table>
Genome Sequencers

- Roche/454
- AB SOLiD
- Illumina MiSeq
- Complete Genomics
- Oxford Nanopore MinION
- Illumina NovaSeq 6000
- Oxford Nanopore GridION

... and more! All produce data with different properties.
The Genomic Era

1990-2003: The Human Genome Project (HGP) provides a complete and accurate sequence of all DNA base pairs that make up the human genome and finds 20,000 to 25,000 human genes.
There are too many (5109) genes in the region. Please narrow the region to enable exon navigation.
1 Sequencing

Reference: TTTATCGCTTCCATGACGCAG
read1: ATCGCATCC
read2: TATCGCATC
read3: CATCCATGA
read4: CGCTTCCAT
read5: CCATGACGC
read6: TTCCATGAC

2 Read Mapping

3 Variant Calling

4 Scientific Discovery
1 Sequencing

Read Mapping

Bottlenecked in Mapping!!

Illumina HiSeq4000

300 M bases/min

on average

2 M bases/min (0.6%)
The Read Mapping Bottleneck

Illumina HiSeq4000

300 Million bases/minute

2 Million bases/minute

150X slower
637

82801 agacttttcc gacggtccgga aaagttccatg gggggaatgg ggttatgagg ggaaagacta
82861 ttatgattat cggcggcggga cgttccaaagg tctacgttcc gcggcggacct ttatattggc
82921 cagccatcct acagttttgag ccattttaaa attaccgccg aatcggagcc aattagtttt
82981 ggggagcgcc aacgtgaaaa aatctggaaga taaaatattg agtaaatagtt taatcagatt
83041 agtttttcata ttctttttccc aaaccttaaat atacacataa ttttgcaatag attttgggat
83101 gtttttaacaat gttaatctttttt aagttctcaat tcgacatttt attttttttttt ttttttttttgt
83161 gaaaccataag aagttttttag gttaaaaaatt cgaatgcccgt cgttcaactt ggattcaatt
83221 tgtgtgttccg acgttgatcat caaggagagcc ctcttgcaccc cttggagagaca ctcgcctgtc
83281 caattcaaat aacatcttgcg ccaggggcttt cccacatccaa atgcacctgcgt agtgcattca
83341 aatctggactt cgctgcgtatg aatttcagggc attcgacaaac tgaattcaaat cactccaaatt
83401 aggaagaattt atttcgtcaca cccgagtcgag ctggaagaaatt ttcagggcct tctacgattt
83461 ggattgtggag tgcgtgcatg ggtcgccattt ggagtggtgg agttttgagg attttggggttt
83521 gtggtgtcgac gggtgtctggg taccggatgt agggagttggt ctaacactgcgt atcggaaacta
83581 aggctttcagtt gacctcaattta atttttagag catttcaag acacttcgac gcagagaaac
83641 ttgagctcag gtttgctcattt ggagcgcagc atcgatttggt tgggcatgg tcttttcatat
83701 ggcctggtcct gcctggagttt ccgttgctgc tgaatatttt ttaacccgctgg cccggggtggt
83761 aaaaattggaa cggaggagggg tgaaggagggg cggaggagggg cgtgaagagg tgggggttaat
83821 attttttgaa ccagtgggttta aacgcgacccct tttcgagatttgc ttaataaatt aataatcgtc
83881 ataatgcgcg acggtcagcag tttttctctatt taaaagaaaaa taaaacacgcga agaaaaatgc
83941 agccaaatttt tgggatgggga aatgagcatttt ttgggtgccag ggagtggtggg gaagaaaaatgc
84001 gggggcaaggg cggggcgcca atggatccgta ttttgatatttt tgggatatgt atttttgacc
84061 gccaaacgcag cgtgagttcag aaggaagcgc gtcttttctttttttt cttggagagaag cggggtggttt
84121 aaaaattgggg tatttttctc ctgtttttttttttttttttttttttttttttttgt gggtgggtgta
84181 tggggttagg ggggctgccgcacct aacgtctccttt tatttttaat cttttttttttt ttatatttttaa
84241 aacatttttc acggtcacaac atatcgctgga tggctgtgtttt cttttttttttttt tgggctgacat
84301 aatccttcct aacgtggggg aaaaaataagtt tggagagagg tggagagagg gggggagggg
84361 gggggtggtcc gggggttggg gttggaaggag agattttgtgc gcctttggttgg gggccttttca
84421 gaaaaagccca gcgaagtcgtg aaagctgcttt atatatcgcctttt aaaaatatg ggggcttggg
84481 acacccacat gcaatcactgct aatctgccaaa tgggtgctgcga cctgtttgtt tatagccttt
84541 gatcgtacaaag aagaaaaactt gctcatattc ctaaaatggag gaaaaagtttt cattttgttttt
84601 tgtctttgac aacaatatggg gggagagaaa ttattttttta taatattcat ttatccagga
84661 attataattg tggtaagttg gttggaagtttg tggggttttttt ttttatttttt tttttttttttttt
84721 gcagagagaa tttggcggtttt ggtttatataa aataataaaaatttacta aatctttatgcc
84781 acgttatatt gcattccaaaca atccataaccgg tttttttttttttt tttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
We know the sequence—but can we understand it?

Anna Pavlovna's drawing room was gradually filling. The highest Petersburg society was assembled there: people differing widely in age and character but alike in the social circle to which they belonged. Prince Vasili's daughter, the beautiful Helene, came to take her father to the ambassador's entertainment; she wore a ball dress and her badge as maid of honor. The youthful little Princess Bolkonskaya, known as la femme la plus seduisante de Petersbourg, was also there. She had been married during the previous winter, and being pregnant did not go to any large gatherings, but only to small receptions. Prince Vasili's son, Hippolyte, had come with Mortemart, whom he introduced. The Abbe Morio and many others had also come.

To each new arrival Anna Pavlovna said, "You have not yet seen my aunt," or "You do not know my aunt?" and very gravely conducted him or her to a little old lady, wearing large bows of ribbon in her cap, who had come sailing in from another room as soon as the guests began to arrive; and slowly turning her eyes from the visitor to her aunt, Anna Pavlovna mentioned each one's name and then left them.

--Tolstoy, War and Peace
Гостинная Анны Павловны начала понемногу наполняться. Приехала высшая знать Петербурга, люди самые разнородные по возрастам и характерам, но одинаковые по обществу, в каком все жили; приехала дочь князя Василия, красавица Элен, заехавшая за отцом, чтобы с ним вместе ехать на праздник посланника. Она была в шифре и бальном платье. Приехала и известная, как la femme la plus séduisante de Pétersbourg ₁, молодая, маленькая княгиня Болконская, прошлую зиму вышедшая замуж и теперь не выезжавшая в большой свет по причине своей беременности, но ездившая еще на небольшие вечера. Приехал князь Ипполит, сын князя Василия, с Мортемаром, которого он представил; приехал и аббат Морио и многие другие.

— Вы не видели еще, — или: — вы не знакомы с ma tante? ₂ — говорила Анна Павловна приезжавшим гостям и весьма серьезно подводила их к маленькой старушке в высоких бантах, выплывшей из другой комнаты, как скоро стали приезжать гости, называла их по имени, медленно переводя глаза с гостя на ma tante, и потом отходила.

Все гости совершали обряд приветствования никому не известной, никому не интересной и не нужной тетушки. Анна Павловна с грустным, торжественным участием следила за их приветствиями, молчающе одобряя их. Ma tante каждому говорила в одних и тех же выражениях о его здоровье, о своем здоровье и о здоровье ее величества, которое нынче было, слава Богу, лучше. Все подходившие, из приличия не выказывая поспешности, с чувством облегчения исполненной тяжелой обязанности отходили от старушки, чтоб уж весь вечер ни

---Tolstoy, War and Peace
Understanding the genome

Even if we did, we don’t know the grammar, or punctuation:

Tolstoy, War and Peace
Most of Bioinformatics focuses on genes, mRNA (i.e. gene activity) and proteins. Cells express different subset of the genes in different tissues and under different conditions.
A) Each triplet of bases codes for one amino acid.
B) Genes differ for the amount of messenger RNA and protein molecules they produce (variable among cells type, position and time regulation).
C) Potentially the DNA strands could code for 6 proteins.
The 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.

**How to identify Genes and gene parts?**

Hidden Markov models
Splice Sites

Transcription start

Promoter

5' UTR

Intron

Stop codon

5GCCATGCCCTTCCTCAAGCCTGAGT

Donor site

5' splice site

GGCCAGAAACAATACAAACCAC

Accoptor site

Poly-A site

3' UTR

Exon

344
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.
The dishonest casino model

P(1|F) = 1/6
P(2|F) = 1/6
P(3|F) = 1/6
P(4|F) = 1/6
P(5|F) = 1/6
P(6|F) = 1/6

P(1|L) = 1/10
P(2|L) = 1/10
P(3|L) = 1/10
P(4|L) = 1/10
P(5|L) = 1/10
P(6|L) = 1/2
**Definition:** A hidden Markov model (HMM)

- **Alphabet** \( \Sigma = \{ b_1, b_2, ..., b_M \} \)
- **Set of states** \( Q = \{ 1, ..., K \} \)
- **Transition probabilities** between any two states
  
  \[
  a_{ij} = \text{transition prob from state } i \text{ to state } j
  \]
  
  \[
  a_{i1} + ... + a_{iK} = 1, \text{ for all states } i = 1...K
  \]

- **Start probabilities** \( a_{0i} \)
  
  \[
  a_{01} + ... + a_{0K} = 1
  \]

- **Emission probabilities** within each state
  
  \[
  e_i(b) = P( x_i = b \mid \pi_i = k)
  \]
  
  \[
  e_i(b_1) + ... + e_i(b_M) = 1, \text{ for all states } i = 1...K
  \]
A Hidden Markov Model is memory-less

At each time step $t$, the only thing that affects future states is the current state $\pi_t$

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

Given a sequence \( x = x_1 \ldots x_N \),

A parse of \( x \) is a sequence of states \( \pi = \pi_1, \ldots, \pi_N \).
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)

\[
P(x, \pi) = P(x_1, ..., x_N, \pi_1, ...., \pi_N) =
P(x_N, \pi_N \mid \pi_{N-1}) P(x_{N-1}, \pi_{N-1} \mid \pi_{N-2}) \ldots P(x_2, \pi_2 \mid \pi_1)
\]

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

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

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

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

\[
\frac{1}{2} \times P(1 \mid \text{Fair}) P(\text{Fair} \mid \text{Fair}) P(2 \mid \text{Fair}) P(\text{Fair} \mid \text{Fair}) \ldots P(4 \mid \text{Fair}) = \\
\frac{1}{2} \times (1/6)^{10} \times (0.95)^9 = .0000000521158647211 = 0.5 \times 10^{-9}
\]
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

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

$\frac{1}{2} \times P(1 \mid \text{Loaded}) P(\text{Loaded, Loaded}) \ldots P(4 \mid \text{Loaded}) = \frac{1}{2} \times (1/10)^8 \times (1/2)^2 \times (0.95)^9 = .00000000078781176215 = 7.9 \times 10^{-10}$

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

\[ \pi = L, L, \ldots, L \? \]

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

So, it is 100 times more likely the die is loaded
The three main questions on HMMs

1. Evaluation
   GIVEN a HMM M, and a sequence x,
   FIND $\text{Prob}[x | M]$

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 \textbf{LEARNING} problem we always write \( P[ x | \theta ] \) to emphasize that we are seeking the \( \theta \) that maximizes \( P[ x | \theta ] \).
Decoding

GIVEN $x = x_1x_2\ldots x_N$

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

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

We can use dynamic programming!

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

$= \text{Probability of most likely sequence of states ending at state } \pi_i = k$
Decoding – main idea

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

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

What is $V_k(i+1)$?

From definition,

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

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

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

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

Input: $x = x_1 \ldots x_N$

**Initialization:**

$V_0(0) = 1$  \hspace{1cm} (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

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

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 $\sim 1/6$ 6’s, followed by a portion of $\sim 1/2$ 6’s...

$x = 123456123456...123456626364656...1626364656$

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

```
            FFF..........................F
            LLL..........................L
```

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 $x = 123416231\textcolor{red}{6261636461623416}1221341$

Most likely path: $\pi = \text{FF}......\text{F}$
However: marked letters more likely to be L than unmarked letters
Evaluation

We will develop algorithms that allow us to compute:

\[ P(x) \quad \text{Probability of } x \text{ given the model} \]

\[ P(x_i...x_j) \quad \text{Probability of a substring of } x \text{ given the model} \]

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

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

We want to calculate

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

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

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

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

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

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

**Termination:**

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

Where, $a_{k0}$ is the probability that the terminating state is $k$ (usually $= a_{0k}$)
Relation between Forward and Viterbi

**VITERBI**

**Initialization:**
- $V_0(0) = 1$
- $V_k(0) = 0$, for all $k > 0$

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

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

**FORWARD**

**Initialization:**
- $f_0(0) = 1$
- $f_k(0) = 0$, for all $k > 0$

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

**Termination:**
- $P(x) = \sum_k f_k(N) a_{k0}$
Motivation for the Backward Algorithm

We want to compute
\[ P(\pi_i = k \mid x), \]
the probability distribution on the \( i^{\text{th}} \) position, given \( x \)

We start by computing
\[
P(\pi_i = k, x) = P(x_1 \ldots x_i, \pi_i = k, x_{i+1} \ldots x_N)
\]
\[= P(x_1 \ldots x_i, \pi_i = k) \cdot P(x_{i+1} \ldots x_N \mid x_1 \ldots x_i, \pi_i = k)\]
\[= P(x_1 \ldots x_i, \pi_i = k) \cdot P(x_{i+1} \ldots x_N \mid \pi_i = k)
\]
\[
\text{Forward, } f_k(i) \quad \text{Backward, } b_k(i)
\]
The Backward Algorithm – derivation

Define the backward probability:

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

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

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

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

\[ = \sum_l e_l(x_{i+1}) a_{kl} b_l(i+1) \]
The Backward Algorithm

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

**Initialization:**

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

**Iteration:**

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

**Termination:**

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

The GENSCAN Web Server at MIT

Identification of complete gene structures in genomic DNA

For information about Genescan, 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

\[
\begin{align*}
\text{GT: signals start of intron} & \\
\text{AG: signals end of intron}
\end{align*}
\]

revcomp(CT)=AG

revcomp(AC)=GT
A simple gene model
A probabilistic gene model

Every box stores transition probabilities for outgoing arrows.
Every arrow stores emission probabilities for emitted nucleotides.

$\text{Pr(TACAGTAGATATGA)} = 0.0001$
$\text{Pr(AACAGT)} = 0.001$
$\text{Pr(AACAGTAC)} = 0.002$
...
• For a given sequence, a **parse** is an assignment of gene structure to that sequence.

• In a parse, every base is labeled, corresponding to the content it **(is predicted to)** belongs to.

• In our simple model, the parse contains only “I” **(intergenic)** and “G” **(gene)**.

• A more complete model would contain, e.g., “-” for intergenic, “E” for exon and “I” for intron.
The probability of a parse

\[ \Pr(\text{parse } P | \text{ sequence } S, \text{ model } M) = 0.67 \times 0.0000543 \times 1.00 \times 0.0000000142 \times 0.75 \times 0.0000789 = 3.057 \times 10^{-18} \]
Improved model topology

- Draw a model that includes introns
Improved model topology

Start

Transcription start

5' splice site

Transcription stop

3' splice site

End
Improved model topology

Start

Transcription start

5’ splice site

Transcription stop

3’ splice site

4 intergenics
1 intron
4 exons

End
Improved model topology

Start

Transcription start

5’ splice site

Initial exon

Intron

Internal exon

Single exon

Final exon

3’ splice site

Transcription stop

End
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

Diagram showing the relationship between 5’ splice site, intron, and 3’ 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
- \(E_{\text{sngl}}\) – single exon (intronless) (translation start -> stop codon)
- \(E_{\text{init}}\) – initial exon (translation start -> donor splice site)
- \(E_{k}\) – phase k internal exon (acceptor splice site -> donor splice site)
- \(E_{\text{term}}\) – terminal exon (acceptor splice site -> stop codon)
- \(I_{k}\) – phase k intron: 0 – between codons; 1 – after the first base of a codon; 2 – after the second base of a codon
Genescan 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
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: $\text{Sn} = \frac{N_{\text{True Positives}}}{N_{\text{All True}}}$
  - “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: $\text{Sp} = \frac{N_{\text{True Positives}}}{N_{\text{All Positives}}}$
  - “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: \( \text{Sn} = \frac{N_{\text{True Positives}}}{N_{\text{All True}}} \)
  
  - “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: \( \text{Sp} = \frac{N_{\text{True Positives}}}{N_{\text{All Positives}}} \)
  
  - “What fraction of my predictions are true?”

- In general, increases

\[
CC = \frac{(TP)(TN) - (FP)(FN)}{\sqrt{(AN)(PP)(AP)(PN)}}
\]

\[
AN = TN + FP; \ AP = TP + FN;
\]

\[
PP = TP + FP; \ PN = TN + FN
\]
Graphic View of Specificity and Sensitivity

\[ S_n = \frac{\text{True Positive}}{\text{All True}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \]

\[ S_p = \frac{\text{True Positive}}{\text{All Positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \]

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)

OR by pasting sequence(s) in FASTA format:
>AAA39861.1 opsin [Mus musculus]
MGTEGPNFYVPSNFVGVQGRPFQSYLAEAEPWQFSMLAAYMFPLLIVGLGFPPFNLTYTVQHKLR
PLNYLILNLAADDLMSVGGPTTLTSTSLGYFYVGTTCNLCBGFAILGSEIALWMSLVLAIEYVVVC
KPWSSVITRIHHMVTVWEALACASAAPLAVSVTIPGQCEGQSCGIDYTLKPSVWESFIVYMFV
HFTIPMVIFPPFYCGQFVTVEEAABAQGEDSATQAKKEVFMWVIIMVIFPLCNLPYASVAFYIFTBQG
SNFGPIMPTRPAFFAKSSSIYNPVYMLNQFRNCMLTTLCCGKPGDASATKETSTQVAPA

Output format:
- Extensive, with graphics
- Extensive, no graphics
- One line per protein

Other options:
- Use old model (version 1)

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.
The PSIPRED Protein Sequence Analysis Workbench

The PSIPRED Protein Sequence Analysis Workbench aggregates several UCL structure prediction methods into one location. Users can submit sequence, perform the predictions of their choice and receive the results of the prediction via e-mail or the web.

For a summary of the available methods you can read More...

NOTE: users who need to run our methods on a large number of proteins should consider downloading our software using the menu on the left (Navigation -> Software Download).

The PSIPRED Team
Current Contributors David T. Jones, Daniel Buchan, Domenica Cozzetto & Kevin Bryson
Previous Contributors Tim Nugent, Federico Minneci, Anna Lobley, Sean Ward, Liam J. McGuffin

For queries regarding PSIPRED: psipred@cs.ucl.ac.uk
TMHMM: uses cyclic model with 7 states for
- TM helix core
- TM helix caps on the N- and C-terminal side
- non-membrane region on the cytoplasmic side
- 2 non-membrane regions on the non-cytoplasmic side (for short and long loops to account for different membrane insertion mechanism)
- a globular domain state in the middle of each non-membrane region
Example for TMHMM
www.cbs.dtu.dk/services/TMHMM/

>gi|218694017|ref|YP_002401684.1| membrane protein; channel [Escherichia coli 55989]
MQDLISQVEDLAGIEIDHTTSMVMIFGIIFLTAVVVHIILHWVVLRTFEKRAIASS
RLWLQIIITQNKLFH
RLAFTLQGIIVNIQAVFWLQKGTEAADILTTCAQLWIMMYALLSVFSLLDVILNL
AQKFPAASQPLKGI
FQGIKLIGAILVGILMISLLIGQSPAILISGLGAMAAVLMLVFKDPILGLVAGIQLS
ANDMLKLGDWLEM
PKYGADGAVIDIGLTTV keyValue|VRNWDNTITTIPTWSLVSDFKNWSGMSASGGR
IKRISIDVTSIRFLDED
EMQRLNKAHLLLKPYLTTRHQEINEWNRQQGSTESILNLRRMTNIGTFRAYLN
EYLRNHPRIRKDMTLMVR
QLAPGDNLPLEIYAFTNTVVWLEYESIQADIFDHIFAIVEEFGLRLHQSPTGN
DIRSLAGAFKQ
# Sequence Length: 274
# Sequence Number of predicted TMHs: 7
# Sequence Exp number of AAs in TMHs: 153.74691
# Sequence Exp number, first 60 AAs: 22.03833
# Sequence Total prob of N-in: 0.04171
# Sequence POSSIBLE N-terminus signal sequence

<table>
<thead>
<tr>
<th>Sequence</th>
<th>TMHMM2.0</th>
<th>outside</th>
<th>1 26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>TMhelix</td>
<td>27 49</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>inside</td>
<td>50 61</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>TMhelix</td>
<td>62 94</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>outside</td>
<td>95 103</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>TMhelix</td>
<td>104 126</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>inside</td>
<td>127 130</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>TMhelix</td>
<td>131 153</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>outside</td>
<td>154 157</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>TMhelix</td>
<td>159 180</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>inside</td>
<td>181 200</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>TMhelix</td>
<td>201 223</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>outside</td>
<td>224 227</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>TMhelix</td>
<td>228 250</td>
</tr>
<tr>
<td>Sequence</td>
<td>TMHMM2.0</td>
<td>inside</td>
<td>251 274</td>
</tr>
</tbody>
</table>

http://www.cbs.dtu.dk/services/TMHMM-2.0/
Back to DNA computing

We turn to biology not just as a metaphor, but as an actual implementation technology ...

Harold Abelson et al., MIT, 2000
Discover magazine published an article in comic strip format about Leonard Adleman's DNA computation.
Bioinformatics: what is for?

- helping doctors and understanding biology:
- Computing with DNA and other biological molecules
- Using DNA to storage information

most of Bioinformatics focuses on genes, mRNA (i.e. gene activity) and proteins
Summary:
Adleman’s DNA computation solved a traveling salesman problem of seven cities. He used DNA techniques to assemble itineraries at random. 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.

Using DNA technologies (PCR, gel electrophoresis and magnetic beads) he selected itineraries from initial city to final city (each city only once).
DNA for computing:
Represent Each City By A DNA Strand of 20 Bases

<table>
<thead>
<tr>
<th>City</th>
<th>DNA Strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>City1</td>
<td>ATGCTCAGCTACTATAGCGA</td>
</tr>
<tr>
<td>City2</td>
<td>TGCAGTGTAACCTAGCATATAT</td>
</tr>
<tr>
<td>City3</td>
<td>GCATATGTTACACTGTACAA</td>
</tr>
<tr>
<td>City4</td>
<td>TTATTAGCGTGCGGCTATG</td>
</tr>
<tr>
<td>City5</td>
<td>CCGCGATAGGTCTAGATTTCC</td>
</tr>
</tbody>
</table>

Etc.

Represent Each Air Route By Mixed Complementary Strands

<table>
<thead>
<tr>
<th>Route</th>
<th>DNA Strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>City 1-&gt;2</td>
<td>TGATATCGCTACGCTACATG</td>
</tr>
<tr>
<td>City 2-&gt;3</td>
<td>ATCGTGATACGTATACCAT</td>
</tr>
<tr>
<td>City 3-&gt;4</td>
<td>GTGACATGTTAATAATCGCA</td>
</tr>
<tr>
<td>City 4-&gt;5</td>
<td>CGCCGGATACGGCGCTATCA</td>
</tr>
<tr>
<td>City 5-&gt;6</td>
<td>GATCTAAAGGTATGCATACG</td>
</tr>
</tbody>
</table>

Etc.

L. Adelman, *Scientific American*, pp. 54-61 (Aug 1998);
figures from Martyn Amos

(a) cities

Vertex 1  Vertex 2  Vertex 3
Vertex 4  Vertex 5  Vertex 6
Vertex 7

selection for length and initial/end points

routes

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
V5 to V2  V5 to V6
V6 to V2  V6 to V7

(b)
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.
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 (=binds) 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.
Itineraries Selection:  
Start and End with Correct Cities

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 cycles 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.
References


Random access in large-scale DNA data storage

Lee Organick¹, Siena Dumas Ang², Yuan-Jyue Chen², Randolph Lopez³, Sergey Yekhanin², Konstantin Makarychev²,⁵, Miklos Z Racz²,⁵, Govinda Kamath²,⁵, Parikshit Gopalan²,⁵, Bichlien Nguyen², Christopher N Takahashi¹, Sharon Newman¹,⁵, Hsing-Yeh Parker², Cyrus Rashtchian², Kendall Stewart¹, Gagan Gupta², Robert Carlson², John Mulligan², Douglas Carmean², Georg Seelig¹,⁴, Luis Ceze¹ & Karin Strauss²

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.

<table>
<thead>
<tr>
<th>Data</th>
<th>File size</th>
<th>Number of DNA sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK GO (HD video)</td>
<td>44.2 MB</td>
<td>3.2 million</td>
</tr>
<tr>
<td>Classical music collection (Music)</td>
<td>13.9 MB</td>
<td>890,000</td>
</tr>
</tbody>
</table>

A comparison to research achievements shows that our coding scheme has similar logical redundancy, but requires lower sequencing coverage to recover files.
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.
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.
Future Directions of Bioinformatics
Typical research pipeline in bioinformatics

Genome Sequencing & assembly

Phylogeny

Storage

1. Isolate human DNA sequence

2. Translate DNA sequence into amino acid sequences

3. Find similar sequences in databases of model organism proteins (red areas reflect no differences; peach, small variations)

4. Model human protein structure based on known structure of a similar protein from a model organism (red area is encoded by the sequence shown)

5. Find drug that binds to modeled protein
Machine learning makes expert knowledge scalable

Impact on Personalized Medicine
- **Computer vision**: Classify pictures in dermatology, radiology, etc.
- **Natural language processing**: Annotating free text electronic health documents
- **Data mining**: Identifying hidden patterns in large clinical and biological datasets

Dermatologist-level classification of skin cancer with deep neural networks

Andre Esteva*, Brett Kupreś*, Roberto A. Nueva†, Justin Ko‡, Susan M. Swetter§, Helen M. Blau¶ & Sebastian Thrun∥
Data-driven advice for applying machine learning to bioinformatics problems

Randal S. Olson†*, William La Cava†*, Zairah Mustahsan, Akshay Varik, and Jason H. Moore†

Institute for Biomedical Informatics, University of Pennsylvania
Philadelphia, PA 19104, USA
†E-mails: rso@randalolson.com, lacava@upenn.edu and jhmoore@upenn.edu

As the bioinformatics field grows, it must keep pace not only with new data but with new algorithms. Here we contribute a thorough analysis of 13 state-of-the-art, commonly used machine learning algorithms on a set of 165 publicly available classification problems in order to provide data-driven algorithm recommendations to current researchers. We present a number of statistical and visual comparisons of algorithm performance and quantify the effect of model selection and algorithm tuning for each algorithm and dataset. The analysis culminates in the recommendation of five algorithms with hyperparameters that maximize classifier performance across the tested problems, as well as general guidelines for applying machine learning to supervised classification problems.

Keywords: machine learning; data science; best practices; benchmarking; bioinformatics
Heat map showing the percentage out of 165 datasets a given algorithm outperforms another algorithm in terms of best accuracy on a problem. The algorithms are ordered from top to bottom based on their overall performance on all problems. Two algorithms are considered to have the same performance on a problem if they achieved an accuracy within 1% of each other.
Example class
1 Bioinformatics (PL)

(a) What are the usage and the limitations of the Bootstrap technique in phylogeny?  
[6 marks]

*Answer:* This is a procedure of resampling of the sites in an alignment and tree reconstructions of all the pseudo alignments; it depends on the size of the alignment (length of the sequences and their number). The percentage of times each interior branch is given a value of 1 is noted. This is known as the bootstrap value. As a general rule, if the bootstrap value for a given interior branch is 95% or higher, then the topology at that branch is considered correct. The presence of several repeated columns decreases the amount of information in each pseudoalignment.

(c) How can you evaluate the results obtained (number of clusters and their relative position) using the K means algorithm for clustering?  
[5 marks]

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

Bioinformatics

(a) Discuss the space–time complexity of dynamic programming algorithms in sequence alignment. [7 marks]

(b) Discuss with one example how to score a multiple sequence alignment. [5 marks]
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.
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]
(a) What are the usage and the limitations of the Bootstrap technique in phylogeny? [6 marks]

(b) We often use Hidden Markov Models (HMM) to predict a pattern (for instance the exons). How can you compute the number of True Positives, True Negatives, False Positives and False Negatives and use them to evaluate your HMM? [6 marks]

(c) How can you evaluate the results obtained (number of clusters and their relative position) using the K means algorithm for clustering? [5 marks]
We often use Hidden Markov Models (HMM) to predict a pattern (for instance the exons). How can you compute the number of True Positives, True Negatives, False Positives and False Negatives and use them to evaluate your HMM?

[6 marks]

Answer:

(i) be predicted to occur: Predicted Positive (PP)

(ii) be predicted not to occur: Predicted Negative (PN)

(iii) actually occur: Actual Positive (AP)

(iv) actually not occur: Actual Negative (AN)

(v) True Positive \( TP = PP \cap AP \)

(vi) True Negative \( TN = PN \cap AN \)

(vii) False Negative \( FN = PN \cap AP \)

(viii) False Positive \( FP = PP \cap AN \)

(ix) Sensitivity: probability of correctly predicting a positive example \( Sn = TP/(TP + FN) \)

(x) Specificity: probability of correctly predicting a negative example \( Sp = TN/(TN + FP) \) or

(xi) probability that positive prediction is correct \( Sp = TP/(TP + FP) \)
AS YOU CAN CLEARLY SEE IN SLIDE 397...

GAAAAH!

“POWERPOINT” POISONING.