BioInformatics 2009-2010
12 lectures -- Pietro Lio', pl219

Bioinformatics is focused on developing algorithms to be used in biological and medical researches. Molecular 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.

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-CCTGAGCCAACTATTGATGAA-3
3-GGACTCGGTTGATAACTACTT-5

Gene: hereditary information located on the chromosomes and consisting of DNA (say 1000 bases);

Genome: an organism’s genetic material; human genome= 46 pieces (chromosomes) with overall length 3 x 10^9 base.

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.

Protein
The Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions in the Manhattan grid.

Goal: Find the longest path in a weighted grid.

Input: A weighted grid $G$ with two distinct vertices, one labeled “source” and the other labeled “sink”

Output: The longest path in $G$ from “source” to “sink”

Greedy: comparing number of attractions by moving one block east or south.
Computing the score for a point \((i, j)\) by the recurrence relation:

- Calculate optimal path score for each vertex in the graph
- Each vertex’s score is the maximum of the prior vertices score plus the weight of the respective edge in between
- The only hitch is that one must decide on the order in which visit the vertices; By the time the vertex \(x\) is analyzed, the values \(s_y\) for all its predecessors \(y\) should be computed.

The running time is \(n \times m\) for a \(n\) by \(m\) grid

**Manhattan Is Not A Perfect Grid**

- The score at point \(B\) is given by:
  \[ s_b = \max \left\{ s_{A_1, B} + \text{weight of the edge between } (i-1, j) \text{ and } (i, j), 
  s_{A_2, B} + \text{weight of the edge between } (i, j-1) \text{ and } (i, j) \right\} \]

**Alignment: 2 row representation**

Given 2 DNA sequences \(v\) and \(w\):

\[ v : \quad A \quad T \quad G \quad T \quad A \quad T \quad m = 7 \]
\[ w : \quad A \quad T \quad C \quad G \quad T \quad A \quad n = 7 \]

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

<table>
<thead>
<tr>
<th>letters of (v)</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>letters of (w)</td>
<td>A</td>
<td>T</td>
<td>C</td>
<td>G</td>
<td>T</td>
<td>-</td>
<td>A</td>
</tr>
</tbody>
</table>

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 
  \( v = v_1 v_2 \ldots v_m \) and \( w = w_1 w_2 \ldots w_n \)
  
* The LCS of \( v \) and \( w \) is a sequence of positions in
  
\( v: 1 \leq i_1 < i_2 < \ldots < i_t \leq m \)

and a sequence of positions in

\( w: 1 \leq j_1 < j_2 < \ldots < j_t \leq n \)

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

**LCS Problem as Manhattan Tourist Problem- Edit Graph for LCS Problem**

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

**LCS: Example**

<table>
<thead>
<tr>
<th>i coords:</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>elements of ( v )</td>
<td>A</td>
<td>T</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>T</td>
<td>G</td>
<td>A</td>
<td>T</td>
</tr>
<tr>
<td>elements of ( w )</td>
<td>-</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>T</td>
<td>-</td>
<td>A</td>
<td>-</td>
</tr>
</tbody>
</table>

**Computing LCS**

Let \( v_i \) = prefix of \( v \) of length \( i \): \( v_1 \ldots v_i \)

and \( w_j \) = prefix of \( w \) of length \( j \): \( w_1 \ldots w_j \)

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

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

Every Path in the Grid corresponds to an Alignment.

\[
\begin{array}{cccccc}
W & A & T & C & G \\
0 & | & | & | & | \\
1 & 2 & 3 & 4 & 5 \\
2 & & & & & \\
3 & & & & & \\
4 & & & & & \\
\end{array}
\]
The Edit distance between two strings is the minimum number of operations (insertions, deletions, and substitutions) to transform one string into the other.

**Hamming distance**
- Always compares
- $i^{th}$ letter of $v$ with $i^{th}$ letter of $w$

**Edit distance**
- May compare
- $i^{th}$ letter of $v$ with $j^{th}$ letter of $w$

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

Edit distance:
$d(v, w) = 2$
Computing edit distance is a non-trivial task

---

**Alignment as a Path in the Edit Graph**

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

---

**Edit Distance: Example**

TGCATAT $\rightarrow$ ATCCGAT in 4 steps

- TGCATAT $\rightarrow$ (insert A at front)
- ATGCATA $\rightarrow$ (delete 6th T)
- ATGCGTA $\rightarrow$ (substitute G for 5th A)
- ATCCGAT $\rightarrow$ (substitute C for 3rd G)

**Alignment: Dynamic Programming**

$s_{i,j} = \begin{cases} 
  s_{i-1,j-1} + 1 & \text{if } v_i = w_j \\
  \max(s_{i-1,j} + 0, s_{i,j-1} + 0) & \text{otherwise}
\end{cases}$

This recurrence corresponds to the Manhattan Tourist problem (three incoming edges into a vertex) with all horizontal and vertical edges weighted by zero.
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).

Fasta Format

```plaintext
>gi|18089116|gb|BC020718.1| Homo sapiens I factor
AAATTTCAAAAGAATACCTGGAGTGGAAAAGAGTTCTCAGCAGAGACAAAGACCCGAACACCTCCAACA
TGAAGCTTCTTCATGTTTTCCTGTTATTTCTGTGCTTCCACTTAAGGTTTTGCAAGGTCACTTATACATC
TCAAGAGGATCTGGTGGAGAAAAAGTGCTTAGCAAAAAAATATACTCACCTCTCCTGCGATAAAGTCTTC
TGCCAGCCATGGCAGAGATGCATTGAGGGCACCTGTGTTTGTAAACTACCGTATCAGTGCCCAAAGAATG
GCACTGCAGTGTGTGCAACTAACAGGAGAAGCTTCCCAACATACTGTCAACAAAAGAGTTTGGAATGTCT
TCATCCAGGGACAAAGTTTTTAAATAACGGAACATGCACAGCCGAAGGAAAGTTTAGTGTTTCCTTGAAG
CATGGAAATACAGATTCAGAGGGAATAGTTGAAGTAAAACTTGTGGACCAAGATAAGACAATGTTCATAT
GCAAAAGCAGCTGGAGCATGAGGGAAGCCAACGTGGCCTGCCTTGACCTTGGGTTTCAACAAGGTGCTGA
TACTCAAAGAAGGTTTAAGTTGTCTGATCTCTCTATAAATTCCACTGAATGTCTACATGTGCATTGCCGA
GGATTAGAGACCAGTTTGGCTGAATGTACTTTTACTAAGAGAAGAACTATGGGTTACCAGGATTTCGCTG
ATGTGGTTTGTTATACACAGAAAGCAGATTCTCCAATGGATGACTTCTTTCAGTGTGTGAATGGGAAATA
CATTTCTCAGATGAAAGCCTGTGATGGTATCAATGATTGTGGAGACCAAAGTGATGAACTGTGTTGTAAA
GCATGCCAAGGCAAAGGCTTCCATTGCAAATCGGGTGTTTGCATTCCAAGCCAGTATCAATGCAATGGTG
AGGTGGACTGCATTACAGGGGAAGATGAAGTTGGCTGTGCAGGCTTTGCATCTGTGGCTCAAGAAGAAAC
AGAAATTTTGACTGCTGACATGGATGCAGAAAGAAGACGGATAAAATCATTATTACCTAAACTATCTTG
GGAGTTAAAAACAGAATGCACATTCGAAGGAAACGAATTGTGGGAGGAAAGCGAGCACAACTGGAAAAAA
TGAAGCAAATCTCATTGGATATTTTTAAAGGTCTCCACAGAGTTTATGCCATATTGGAATTTTGTTGTATA
ATTCTCAAATAATAATTTGUGAGCCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

GenBank Format

```
1. BC020718: Homo sapiens I factor [gi:18089115] Link
LOCUS       BC020718                1249 bp    mRNA    linear   PRI 06-OCT-2003
DEFINITION  Homo sapiens I factor (complement), mRNA (cDNA clone MGC:22501
VERSE     BC020718.1  GI:18089116
KEYWORDS    MGC.
SOURCE      Homo sapiens (human)
FEATURES             Location/Qualifiers
source          1..1249
/organism="Homo sapiens"
/mol_type="mRNA"
/db_xref="taxon:9606"
/clone="MGC:22501 IMAGE:4716122"
/gene="IF"
/gene="synonym: FI"
/gene="db_xref="GeneID:3426"
/gene="db_xref="MIM:217030"
/CDS             70..1203
/codon_start=1
/product="IF protein"
/protein_id="AAH20718.1"
/db_xref="GI:18089117"
/db_xref="GeneID:3426"
/db_xref="MIM:217030"
/translation="MKLLHVFLLFLCFHLRFCKVTYTSQEDLVEKKCLAKKYTHLSCD
KVFCQPWQRCIEGTCVCKLPYQCPKNGTAVCATNRRSFPTYCQQKSLECLHPGTKFLN
NGTCTAEGKFSVSLKHGNTDSEGIVEVKLVDQDKTMFICKSSWSMREANVACLDLGFQ
QGADTQRRFKLSDLSINSTECLHVHCRGLETSLAECTFTKRRTMGYQDFADVVCYTQK
ADSTPDDFQCVNGKYISQMKACDGINDCGDQSDELCCKACQGKGFHCKSGVCIPSQY
QCNGEVDCITGEDEVGCAGFASVAQEETEILTADMDAERRRIKSLLPKLSCGVKNRMH
RRKRVGSGMRILGKMQILSDFPGKHVAYLVEFCGLK"
```

1/23/10
- BioJava – [www.biojava.org](http://www.biojava.org)
- BioPerl – [www.bioperl.org](http://www.bioperl.org) (till now the dominant language in bioinformatics; loosely typed)
- BioPython – [www.biopython.org](http://www.biopython.org)
- BioCorba – [www.biocorba.org](http://www.biocorba.org) (can be used to tying it all together; strongly typed)

### BioInformatics 2: sequence alignment

|------------|----------|----------|--------|

Notice three possible cases:

1. $x_i$ aligns to $y_j$
   
   $x_1, \ldots, x_i, x$
   
   $y_1, \ldots, y_j, y$

   $F(i,j) = F(i-1, j-1) + m$, if $x_i = y_j$
   
   $F(i,j) = F(i-1, j-1) - s$, if not

2. $x_i$ aligns to a gap
   
   $x_1, \ldots, x_i, x$
   
   $y_1, \ldots, y_j, -$  

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

3. $y_j$ aligns to a gap
   
   $x_1, \ldots, x_i, -$  
   
   $y_1, \ldots, y_j, y$

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

How do we know which case is correct?

**Inductive assumption:**

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

Then,

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

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

---

**Definition**

Given two strings $x = x_1x_2 \ldots x_m$, $y = y_1y_2 \ldots y_n$

an alignment is an assignment of gaps to positions $0, \ldots, n$ in $x$, and $0, \ldots, m$ in $y$, so as to line up each letter in one sequence with either a letter, or a gap in the other sequence.
• 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-ACT---TAATG--CAGGGGACACG--A-GCATCAGA--GAC

AAATGCGCC--GTGCT--T--TTCAG----CA-GTTATG--T--CAGAT--C
```

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

```
tccTGATATGTCAGgggacacgagcatgcagagac

aattgccgccgtcgttttcagCCCCAGTTATGTCAGatc
```

### The Needleman-Wunsch Algorithm (Global alignment)

1. **Initialization**
   
   $F(0, 0) = 0$
   
   $F(i, 0) = i \times d$
   
   $F(0, j) = j \times d$

2. **Main Iteration**: Filling-in partial alignments
   
   For each $i = 1, \ldots, M$
   
   For each $j = 1, \ldots, N$

   $F(i, j) = \max$

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

   $P(i, j) =$

   - UP if [case 1]
   - LEFT if [case 2]
   - DIAG if [case 3]

3. **Termination**: $F(M, N)$ is the optimal score, and from $P(M, N)$ can trace back optimal alignment

   Complexity: Space: $O(mn)$; Time: $O(mn)$

   Filling the matrix $O(mn)$

   Backtrace $O(mn)$

### The local alignment: Smith-Waterman algorithm

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

- e.g. $x = aaaaacctccccgggg$
- $y = cccgggacaccaacc$

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

**Iteration**: $F(i, j) = \max$

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

**Termination**:

1. If we want the best local alignment...

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

2. If we want all local alignments scoring > t

   For all $i, j$ find $F(i, j) > t$, and trace back
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:
\[
\gamma(n) = \begin{cases} 
0 & \text{if } n = 0 \\
\gamma(n-1) & \text{otherwise}
\end{cases}
\]
for all n, \( \gamma(n+1) - \gamma(n) \leq \gamma(n) - \gamma(n-1) \)

Initialization: same

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

Termination: same

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

A compromise: affine gaps

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

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

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

Termination: same

Banded Dynamic Programming

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

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

\[
X_i \quad | \quad i - j < k(N) \\
Y_j \quad | \quad i = j + k(N)
\]

Out of range

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

Banded DP

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

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

\[
X_i \quad | \quad i - j < k(N) \\
Y_j \quad | \quad i = j + k(N)
\]

Out of range

Note that for diagonals, \( i+j = \text{constant.} \)

Time, Space: \( O(N \times k(N)) \) << \( O(N^2) \)
Computing Alignment Path Requires Quadratic Memory

**Alignment Path**
- Space complexity for computing alignment path for sequences of length $n$ and $m$ is $O(nm)$
- We need to keep all backtracking references in memory to reconstruct the path (backtracking)

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 Alignment Score: Recycling Columns

Only two columns of scores are saved at any given time

Crossing the Middle Line

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

Define

$$length(i)$$

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

\[\text{length(mid)} = \text{optimal length} = \max_{0 \leq i \leq n} \text{length}(i)\]

**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

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

**Length\((i) = \text{Prefix}(i) + \text{Suffix}(i)\)**
- 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 \(\text{length}(i)\)
BioInformatics 3: can we align Sequences in Subquadratic Time?

- Partition the $n \times n$ grid into blocks of size $t \times t$
- We are comparing two sequences, each of size $n$, and each sequence is sectioned off into chunks, each of length $t$
- Sequence $u = u_1 \ldots u_n$ becomes $|u_1 \ldots u_t| \; |u_{t+1} \ldots u_{2t}| \; \ldots \; |u_{n-t+1} \ldots u_n|$
- Sequence $v = v_1 \ldots v_n$ becomes $|v_1 \ldots v_t| \; |v_{t+1} \ldots v_{2t}| \; \ldots \; |v_{n-t+1} \ldots v_n|$

Block Alignment Problem

- Goal: Find the longest block path through an edit graph
- Input: Two sequences, $u$ and $v$ partitioned into blocks of size $t$. This is equivalent to an $n \times n$ edit graph partitioned into $t \times t$ subgrids
- Output: The block alignment of $u$ and $v$ with the maximum score (longest block path through the edit graph)
Constructing Alignments within Blocks

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

Block Alignment: Dynamic Programming

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

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

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

Block Alignment Runtime

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

$$O(\frac{n}{t}\times\frac{n}{t}) = O(n^2/t^2)$$

if we don’t count the time to compute each $\beta_{ij}$
• Computing all $\beta_{ij}$ requires solving $(n/t)\times(n/t)$ mini block alignments, each of size $(t\times t)$
• Computing all $\beta_{ij}$ takes time $O(\frac{n}{t}\times\frac{n}{t}\times t\times t) = O(n^2)$
• This is the same as dynamic programming
• How do we speed this up?
Four Russians Technique
(Arlazarov, Dinic, Kronrod, Faradzev)

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

Look-up Table for Four Russians Technique

Each sequence has $t$ nucleotides

```
AAAAAA  AAAAC  AAAAG  AAAAT  AAAACA  ...
AAAAAA  AAAAC  AAAAG  AAAAT  AAAACA  ...
```

Look-up table “Score”

- size is only $n$, instead of $(n/t)^2$ $(n/t)$

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

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

Four Russians Speedup Runtime

• Since computing the lookup table Score of size $n$ takes $O(n)$ time, the running time is mainly limited by the $(n/t)^2 \cdot (n/t)$ accesses to the lookup table
• Each access takes $O(\log n)$ time
• Overall running time: $O\left( \left\lceil n^2/t^2 \right\rceil \cdot \log n \right)$
• Since $t = \log n$, substitute in:
• $O\left( \left\lceil n^2/(\log n)^2 \right\rceil \cdot \log n \right) \geq O\left( n^2/\log n \right)$

So Far...

• We can divide up the grid into blocks and run dynamic programming only on the corners of these blocks
• In order to speed up the mini-alignment calculations to under $n^2$, we create a lookup table of size $n$, which consists of all scores for all $t$-nucleotide pairs
• Running time goes from quadratic, $O(n^2)$, to subquadratic: $O(n^2/\log n)$
Four Russians Speedup for LCS

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

Traversing Blocks for LCS

- Given alignment scores $s_{ij}$ in the first row and scores $s_{*,j}$ in the first column of a $t \times t$ mini square, compute alignment scores in the last row and column of the minisquare.
- To compute the last row and the last column score, we use these 4 variables:
  - alignment scores $s_{*,i}$ in the first row
  - alignment scores $s_{*,j}$ in the first column
  - substring of sequence $u$ in this block (4^t possibilities)
  - substring of sequence $v$ in this block (4^t possibilities)
- If we used this to compute the grid, it would take quadratic, $O(n^2)$ time, but we want to do better.

Block Alignment vs. LCS

- In block alignment, we only care about the corners of the blocks.
- In LCS, we care about all points on the edges of the blocks, because those are points that the path can traverse.
- Recall, each sequence is of length $n$, each block is of size $t$, so each sequence has $\frac{n}{t}$ blocks.

Four Russians Speedup

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

• Alignment scores in LCS are monotonically increasing, and adjacent elements can’t differ by more than 1
• Example: 0,1,2,2,3,4 is ok; 0,1,2,4,5,8, is not because 2 and 4 differ by more than 1 (and so do 5 and 8)
• Therefore, we only need to store quadruples whose scores are monotonically increasing and differ by at most 1

Efficient Encoding of Alignment Scores

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

Reducing Lookup Table Size

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

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

RNA structure: great variety!

Pseudoknow are too difficult
RNA Secondary Structure

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

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

---

The Nussinov Folding Algorithm

Example: GGGAAAUCC

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

**Initialisation**

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

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

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

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

Objective: To find the secondary structure with the maximal number of base pairs under the pseudo-knot exclusion constraint.


Filling-stage. Scores for subsequences are recursively computed from and recorded in a quadratic table.

Trace-back. Reconstruction of filling steps indicates optimal structure.

Time-complexity: \(O(N^3)\)
Nussinov Folding Algorithm
After scores for subsequences of length 4

\[
 y(i, j) = \max_{k < j} \left[ y(i, k) + y(k + 1, j) \right]
\]

Two optimal substructures for same subsequence

Nussinov Folding Algorithm
After scores for subsequences of length 5

\[
 y(i, j) = \max_{k < j} \left[ y(i, k) + y(k + 1, j) \right]
\]

Nussinov Folding Algorithm
After scores for subsequences of length 6

\[
 y(i, j) = \max_{k < j} \left[ y(i, k) + y(k + 1, j) \right]
\]

Nussinov Folding Algorithm
After scores for subsequences of length 7

\[
 y(i, j) = \max_{k < j} \left[ y(i, k) + y(k + 1, j) \right]
\]
Nussinov Folding Algorithm
After scores for subsequences of length 8

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

<table>
<thead>
<tr>
<th>i</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Nussinov Folding Algorithm
After scores for subsequences of length 9

\[
y(i, j) = y(i + 1, j) + \delta(i, j)
\]

<table>
<thead>
<tr>
<th>i</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Nussinov algorithm: fill-stage

Algorithm: Nussinov RNA folding, fill stage

1. Initialization:
   \[ y(i, j = 1) = 0 \quad \text{for} \ i = 2 \text{ to } L \]
   \[ y(i, j) = 0 \quad \text{for} \ i = 1 \text{ to } L \]

2. Recursion:
   starting with all subsequences of length \( \lfloor \frac{L}{2} \rfloor \) to length \( L \):
   \[
y(i, j) = \max \begin{cases} y(i + 1, j) \\ y(i + 1, j - 1) + \delta(i, j) \\
\max_{k=i}^{j-1} [y(i, k) + y(k + 1, j)]
\end{cases}
\]

<table>
<thead>
<tr>
<th>i</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

- Pink: joining of substructures 1..4 and 5..8
- Green: addition of paired bases 1,7
- Blue: addition of unpaired base 3 or 7
Combining Optimal Pairwise Alignments into Multiple Alignment

1) Align each sequence against each other giving a similarity matrix; Similarity = exact matches / sequence length (percent identity)
2) Create Guide Tree using the similarity matrix; Guide tree roughly reflects evolutionary relations
3) Progressive Alignment guided by the tree

Progressive Alignment

\[
\begin{align*}
V_1 & \quad V_2 & \quad V_3 & \quad V_4 \\
\text{v}_1 & \quad 0.59 & \quad 0.33 & \quad 0.62 \\
\text{v}_2 & \quad 0.87 & \quad 0.28 & \quad - \\
\text{v}_3 & \quad 0.17 & \quad - & \quad - \\
\text{v}_4 & \quad - & \quad - & \quad - \\
\end{align*}
\]

Calculate:

\[
\begin{align*}
V_{1,2} & = \text{alignment} (V_1, V_2) \\
V_{1,3,4} & = \text{alignment} (V_{1,2}, V_3) \\
V_{2,3,4} & = \text{alignment} (V_{2,3}, V_4)
\end{align*}
\]
How does it work?

- Starting with a group of 7 sequences from different species
- Do pairwise alignments between all 7 sequences
- Score given for similarity, higher score indicates more similar

<table>
<thead>
<tr>
<th></th>
<th>HAHU</th>
<th>HBHU</th>
<th>HANO</th>
<th>HHO</th>
<th>MYWH</th>
<th>PILH</th>
<th>LGHB</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAHU</td>
<td></td>
<td>21.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HBHU</td>
<td>32.9</td>
<td></td>
<td>19.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HANO</td>
<td>20.7</td>
<td>39.0</td>
<td></td>
<td>20.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHO</td>
<td>11.0</td>
<td>9.8</td>
<td>10.3</td>
<td>9.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MYWH</td>
<td>9.3</td>
<td>8.6</td>
<td>9.6</td>
<td>8.4</td>
<td>7.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PILH</td>
<td>7.1</td>
<td>7.3</td>
<td>7.5</td>
<td>7.4</td>
<td>7.3</td>
<td>4.3</td>
<td></td>
</tr>
<tr>
<td>LGHB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Increasing Similarity

- Cluster the sequences by similarity to create a guide tree
- Branch length is proportional to estimated divergence between the two sequences
Sum of Pairs Score (SP-Score)

- Consider pairwise alignment of sequences $a_i$ and $a_j$ imposed by a multiple alignment of $k$ sequences.
- Denote the score of this suboptimal (not necessarily optimal) pairwise alignment as $s^*(a_i, a_j)$.
- Sum up the pairwise scores for a multiple alignment:

$$s(a_1, ..., a_k) = \sum_{i,j} s^*(a_i, a_j)$$

**Given $a_1, a_2, a_3, a_4$:**

$$s(a_1...a_4) = 2s^*(a_1, a_2) + s^*(a_1, a_3) + s^*(a_1, a_4) + s^*(a_2, a_3) + s^*(a_2, a_4) + s^*(a_3, a_4)$$
**FASTA - Heuristic -**

- Problem of Dynamic Programming: D.P. computes the score in a lot of useless areas for optimal sequence
- Heuristic*: Good local alignment should have some exact match subsequence.

Hi level algorithm
Let q be a query
max $\leftarrow 0$
For each sequence, s in DB
  compare q with s and compute a score, y
if max < y
  max $\leftarrow y$
  bestSequence $\leftarrow s$
Return bestSequence

*Heuristic for Dynamic Programming

**FASTA - Algorithm -**

- Step 1
  Find all hot-spots
  // Hot spots are pairs of words of length k that exactly match

- Step 1 in detail
  Use look-up Table
  Query : G A T C A G T T A
  Sequence: G G A T C G A

- Step 2: Score the Hot-spot and locate the ten best diagonal runs
- Step 3: Combine sub-alignments into one alignment with GAP
- Step 4
  # Consider weighted direct graph.
  # Let node be a sub-alignment found in step 1
  # Let u and v be nodes
  # Edge (u,v) exists if alignment u is before in the sequence.
  # Each edge has gap penalty (negative)
  # Find the maximum weight path
**FASTA - Algorithm -**

- **Step 5**
  Use the dynamic programming in restricted area around the best-score alignment to find out the higher-score alignment larger than the best-score alignment.

**Summary of the algorithm**
1. Find all hot-spots
2. Score each Hot-spot and locate the ten best diagonal runs.
3. Combine sub-alignments into one alignment.
4. Score each alignment with gap penalty and pick up the best-score alignment.
5. Use the dynamic programming in restricted area around the best-score alignment to find out the alignment greater than the best-score alignment.

**BLAST**

Basic Local Alignment Search Tool

- Heuristic but evaluating the result statistically.
Homologous sequence are likely to contain a short high scoring word pair, a hit.
BLAST tries to extend it on the both sides to get optimal sequence.

**FASTA - Complexity -**

# Step 1 and 2  // select the best 10 diagonal runs
Let \( n \) be a sequence from DB
\[ O(n) \text{ because Step 1 just uses look up table} \]
\[ O(n) << O(mn) \quad m,n = 100 \text{ to } 200 \]

# Step 3 and 4  // compute the MAX Weight Path
Let \( r \) be the number of sub-alignments. \( (r = 10) \)
\[ O(r^2) < O(mn) \]

# Step 5  // compute partial D.P.
Depends on the restricted area < \( O(mn) \)
BLAST - Algorithm -

1. Step 1: preprocessing Query
   Compile the short-hit scoring word list from query.
   The length of query word, \( w \), is 3 for protein search, 11 for DNA.
   Threshold T is 13

   - Query: LAALLKCKTPQGQRLVNQWIKQPLMD
   - Neighborhood words:
     - Neighborhood score Threshold (T=13)

   - Neighborhood words
     - Neighborhood words

2. Step 2: Scanning DB
   For each words list, identify all exact matches with DB sequences

   - Query Word
     - Neighborhood Word list
     - Sequences in DB
     - Sequence 1
     - Sequence 2

   - The purpose of Step 1 and 2 is as same as FASTA

   - Method 1: Hash Table
     - Query: LAALLKCKTPQGQRLVNQWIKQPLMD
     - Hash Table

   - Hash Table
     - word | position
     - AAA | 1, 2, 15, 16...
     - AGL | 2, 3, 10, 11...
     - AAA | 2, 15, 43...
     - IAA | 1, 5, 7...
     - GLL | 3, 8, 24...
     - VVQ | 4, 21, 25..
**BLAST - Algorithm -**

- Step 2-3
  Method 2: Finite Automata

- Step 3
  - Search optimal alignment
  Let $S$ be a score of hit-word
  For each hit-word, extend ungapped alignment in both directions.

- Step 4
  Evaluate the alignment statistically
  Stop extension when $E$-value (depending on score $S$) become less than threshold.
  The hit-word is called High Scoring Segment Pair. BLAST return it

| Sequence | A T T A G .......... |
|----------------------|
| Hit Word | ............. |

$E$-value = the number of HSPs having score $S$ (or higher) expected to occur only by chance.

$\Rightarrow$ Smaller $E$-value, more significant in statistics

Bigger $E$-value, by chance

**BLAST - Algorithm -**

- Definition of $E$-Value
  The expected number of HSP with the score at least $S$ is:

$$E = K * n * m * e^{-AS}$$

$K$, $\lambda$ is constant depending on model
$n$, $m$ are the length of query and sequence

The probability of finding at least one such HSP is:

$$P = 1 - e^E$$

$\Rightarrow$ If a word is hit by chance ($E$-value is bigger), $P$ become smaller.

**BLAST - Running Time -**

- Running Time on a Pentium 4
  - The length of Query : 153
  - DB size : 5997 sequences

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.P</td>
<td>16.989 [s]</td>
</tr>
<tr>
<td>FASTA</td>
<td>0.618 [s]</td>
</tr>
<tr>
<td>BLAST</td>
<td>0.118 [s]</td>
</tr>
</tbody>
</table>
A spaced seed is formed by two words, one from each input sequence, that match at positions specified by a fixed pattern – a word over symbols # and _ interpreted as a match and a don’t care symbol respectively. For example, pattern ##_# specifies that the first, second and fourth positions must match and the third one may contain a mismatch.

PatternHunter was the first method that used carefully designed spaced seeds to improve the sensitivity of DNA local alignment. Spaced seeds have been shown to improve the efficiency of lossless filtration for approximate pattern matching, namely for the problem of detecting all matches of a string of length \( m \) with \( q \) possible substitution errors (an \((m, q)\)-problem). Other software use some specific spaced seeds and random spaced seeds.

BLAST uses “consecutive seeds”
- In BLAST, we often use the consecutive model with weight 11.
  - GAGTACTCAACACCACACATCGTGCCGCTGAAAAAT
  - GAATCTCAACAGCATCAGTGCCGCAATCGAAAAAT
  - → 111111111111 → ... → ... → 111111111111 →
- However, it fails to find the alignment in the two sequence.
Dilemma: Sensitivity vs Speed

**Similarity**
How similar it is between two sequences?
Usually mean that the probability of the same symbol appear in anywhere of two sequences.

**Sensitivity**
The probability to find a local alignment.
needs shorter seeds
too many random hits, slow computation

**Speed** – needs longer seeds, lose distant homologies

**Specificity**
In all local alignments, how many alignments are homologous

PatternHunter uses “non-consecutive seed”

- In PatternHunter, we often use the spaced model with weight 11 and length 18.

```
GAGTACTCAACACACATCACGTGGCAATGGAAAAT
|| || || || || || || ||
GAATACTCAACAGCAACATCAATGGGCAGCAGAAAAT
111010010100110111
```

- Higher hit probability
- Lower expected number of random hits

A trivial comparison between spaced and consecutive seed

- Consider 111 and 1101.
- To fail seed 111, we can use
  - 110110110110...
  - 66.66% similarity
- But we can prove, seed 1101 will hit every region with 61% similarity for sufficient long region.

Simulated sensitivity curves
Simulated sensitivity curves:

- Solid curves: Multiple (1, 2, 4, 8, 16) weight-12 spaced seeds.
- Dashed curves: Optimal spaced seeds with weight = 11, 10, 9, 8.

Typically, "Doubling the seed number" gains better sensitivity than "decreasing the weight by 1".

Sensitivity curves:

Proof

- Suppose there is a length 100 region which is not hit by 1101.
- We can break the region into blocks of 1*0*. Besides the last block, the other blocks have the following few cases:
  - 10^b for b>=1
  - 110^b for b>=2
  - 1110^b for b>=2
- In each block, similarity <= 3/5.
- The last block has at most 3 matches.
- So, in total there are at most 61 matches in 100 positions. The similarity is <=61%.

Formalize

- Given i.i.d. sequence (homology region) with Pr(1) =p and Pr(0)=1-p for each bit:

  110011101110101110111101101110111011101

  111*1**1*1**11*111

- Which seed is more likely to hit this region:
  - BLAST seed: 1111111111
  - Spaced seed: 111*1**1*1**11*111
Expect Less, Get More

- Lemma: The expected number of hits of a weight $W$ length $M$ seed model within a length $L$ region with homology level $p$ is $(L-M+1)p^W$
- Proof. $E(#\text{hits}) = \sum_{i=1}^{L-M+1} p^W$

Example: In a region of length 64 with $p=0.7$
- $Pr(\text{BLAST seed hits})=0.3$
- $E(\# \text{ of hits by BLAST seed})=1.07$
- $Pr(\text{optimal spaced seed hits})=0.466$, 50% more
- $E(\# \text{ of hits by spaced seed})=0.93$, 14% less

Observations of spaced seeds

- Seed models with different shapes can detect different homologies.
- Two consequences:
  - Some models may detect more homologies than others
    - More sensitive homology search
    - PatternHunter I
  - Can use several seed models simultaneously to hit more homologies
    - Approaching 100% sensitive homology search
    - PatternHunter II

Why Is Spaced Seed Better?

A wrong, but intuitive proof: seed $s$, interval $I$, similarity $p$
$E(#\text{hits}) = Pr(s \text{ hits}) E(#\text{hits} | s \text{ hits})$
Thus:
$Pr(s \text{ hits}) = (L^p / E(#\text{hits} | s \text{ hits})$

For optimized spaced seed, $E(#\text{hits} | s \text{ hits})$

<table>
<thead>
<tr>
<th>111<em>1**1</em>1**11*111</th>
<th>Non overlap</th>
<th>Prob</th>
</tr>
</thead>
</table>
| 111*1**1*1**11*111   | 6           | $p^6$
| 111*1**1*1**11*111   | 6           | $p^6$
| 111*1**1*1**11*111   | 6           | $p^6$
| 111*1**1*1**11*111   | 7           | $p^7$

...$
- For spaced seed: the divisor is $1+p^0+p^1+p^2+p^3+...$
- For BLAST seed: the divisor is bigger: $1+p+p^2+p^3+...$

Example of a hit using a spaced seed:

```
GACTACCTCAACACAACTATCTGGCACTGGGAAAT...
```

- BLAST: redundant hits
- PatternHunter

```
TTGACCTCACC
```

This results in > 1 hit and creates clusters of redundant hits

```
CAA??A??A??CT??TA??TG
```
Why is PH better?

BLAST may also miss a hit

<table>
<thead>
<tr>
<th>GA/TACTCAACA/CAACAT/TA/TGGGCA/GAAAAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA/TACTCAACA/CAACAT/CA/TGGGCA/GCAAAT</td>
</tr>
</tbody>
</table>

In this example, despite a clear homology, there is no sequence of continuous matches longer than length 9. BLAST uses a length 11 and because of this, BLAST does not recognize this as a hit!
Resolving this would require reducing the seed length to 9, which would have a damaging effect on speed.

PatternHunter II:
-- Smith-Waterman Sensitivity, BLAST Speed
(Ji, Ma, Kisman, Tromp, J. Bioinfo Comput. Biol. 2004)

- The biggest problem for BLAST was low sensitivity (and low speed). Massive parallel machines are built to do S-W exhaustive dynamic programming.
- Spaced seeds give PH a unique opportunity of using several optimal seeds to achieve optimal sensitivity, this was not possible by BLAST technology.
- PH II has with multiple optimal seeds.
- PH II approaches Smith-Waterman sensitivity, and 3000 times faster.

BioInformatics 5

Molecular Evolution: Fitch and Sankoff Algorithms

Terminal Nodes

Branches or Lineages

Ancestral Node or ROOT of the Tree

Internal Nodes

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

Parsimony Approach

- Applies Occam’s razor principle to identify the simplest explanation for the data
- Assumes observed character differences resulted from the fewest possible mutations
- Seeks the tree that yields lowest possible parsimony score - sum of cost of all mutations found in the tree

(a) Parsimony Score=3

(b) Parsimony Score=2
**Small Parsimony**

- **Input**: Tree $T$ with each leaf labeled by an $m$-character string.
- **Output**: Labeling of internal vertices of the tree $T$ minimizing the parsimony score.
- We can assume that every leaf is labeled by a single character, because the characters in the string are independent.

**Weighted Small Parsimony Problem**

- **Input**: Tree $T$ with each leaf labeled by elements of a $k$-letter alphabet and a $k \times k$ scoring matrix ($\delta$).
- **Output**: Labeling of internal vertices of the tree $T$ minimizing the weighted parsimony score.
- For Small Parsimony problem, the scoring matrix is based on Hamming distance $d_H(v, w) = 0$ if $v = w$ ; $d_H(v, w) = 1$ otherwise.

---

**Sankoff Algorithm: Dynamic Programming**

- Calculate and keep track of a score for every possible label at each vertex
  - $s_i(v) =$ minimum parsimony score of the subtree rooted at vertex $v$ if $v$ has character $t$
- The score at each vertex is based on scores of its children:
  - $s_i(parent) = \min_j \{ s_j(left \, child) + \delta_j \} + \min_j \{ s_j(right \, child) + \delta_j \}$

---

**Unweighted vs. Weighted**

**Small Parsimony**

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

**Weighted Parsimony**

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Weighted Parsimony Score**: 22

**Small Parsimony Score**: 5

---

**Sankoff Algorithm (cont.)**

- Begin at leaves:
  - If leaf has the character in question, score is 0
  - Else, score is $\infty$
Sankoff Algorithm (cont.)

\[
s_t(v) = \min_i \{ s_i(u) + \delta_{i,v} \} + \min_j \{ s_j(w) + \delta_{j,v} \}
\]

Repeat for right subtree

Repeat for T, G, and C

Sankoff Algorithm (cont.)

\[
s_A(v) = \min_i \{ s_i(u) + \delta_{i,A} \} + \min_j \{ s_j(w) + \delta_{j,A} \}
\]

Repeat for T, G, and C

\[
s_j(u) + \delta_{j,A} + 9 = 9
\]
Sankoff Algorithm (cont.)

Repeat for root

Smallest score at root is minimum weighted parsimony score
In this case, 9 – so label with T

Sankoff Algorithm:
Traveling down the Tree

The scores at the root vertex have been computed by going up the tree
After the scores at root vertex are computed the Sankoff algorithm moves down the tree
and assign each vertex with optimal character.

9 is derived from 7 + 2
So left child is T,
And right child is T

And the tree is thus labeled...
Fitch Algorithm

• Solves Small Parsimony problem;
• Dynamic programming in essence;

1) Assign a set of possible letters to every vertex, traversing the tree from leaves to root
• Each node’s set is the combination of its children’s sets (leaves contain their label)
  – E.g. if the node we are looking at has a left child labeled (A, C) and a right child labeled (A, T), the node will be given the set {A}

2) Assign labels to each vertex, traversing the tree from root to leaves
• Assign root arbitrarily from its set of letters
• For all other vertices, if its parent’s label is in its set of letters, assign it its parent’s label
• Else, choose an arbitrary letter from its set as its label

Parsimony Example

Say we have an alignment of 4 DNA sequences of 3 bases each

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>C</td>
<td>G</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total number of substitutions

Tree 1: 4
Tree 2: 5
Tree 3: 6

• substitution

Sankoff vs. Fitch

• The Sankoff algorithm gives the same set of optimal labels as the Fitch algorithm
• For Sankoff algorithm, character t is optimal for vertex v if \( s_t(v) = \min_{1 \leq i \leq k} s_i(v) \)
  – Denote the set of optimal letters at vertex v as \( S(v) \)
  • If \( S(\text{left child}) \) and \( S(\text{right child}) \) overlap, \( S(\text{parent}) \) is the intersection
  • Else it’s the union of \( S(\text{left child}) \) and \( S(\text{right child}) \)
  • This is also the Fitch recurrence
• Complexity:
• Fitch: \( O(mnk) \); Sankoff: \( O(mnk^2) \)
• m characters, n leaves, k possible values for a character
Large Parsimony Problem

- **Input:** An \( n \times m \) matrix \( M \) describing \( n \) species, each represented by an \( m \)-character string.
- **Output:** A tree \( T \) with \( n \) leaves labeled by the \( n \) rows of matrix \( M \), and a labeling of the internal vertices such that the parsimony score is minimized over all possible trees and all possible labelings of internal vertices.
- Possible search space is huge, especially as \( n \) increases.
  - \((2n - 3)!!\) possible rooted trees
  - \((2n - 5)!!\) possible unrooted trees
- Problem is NP-complete; Exhaustive search only possible with small \( n < 10 \).

### Edit Distance vs. Tree Distance

- Given \( n \) sequences, we can compute the \( n \times n \) distance matrix \( D_{ij} \).
- \( D_{ij} \) may be defined as the edit distance between a gene in species \( i \) and species \( j \), where the gene of interest is sequenced for all \( n \) species.

\[ D_{ij} = \text{edit distance between } i \text{ and } j \]
- Note the difference with

\[ d_{ij}(T) = \text{tree distance between } i \text{ and } j \]

### BioInformatics 6

#### Distance in Trees

**Distance in Trees**

- \( d_{ij}(T) \) – tree distance between \( i \) and \( j \)

**Example:**

\[ d_{i,j} = 12 + 13 + 14 + 17 + 13 = 69 \]

#### Fitting Distance Matrix

- Given \( n \) sequences, we can compute the \( n \times n \) distance matrix \( D_{ij} \).
- Evolution of these sequences is described by a tree that we don’t know.
- We need an algorithm to construct a tree that best fits the distance matrix \( D_{ij} \).

\[ Fitting \text{ means } D_{ij} = d_{ij}(T) \]

Lengths of path in an (unknown) tree \( T \)

- Edit distance between species (known)
Reconstructing a 3 Leaved Tree

• Tree reconstruction for any 3x3 matrix is straightforward
• We have 3 leaves $i, j, k$ and a center vertex $c$

Observe:

$\begin{align*}
    d_{ic} + d_{jc} &= D_{ij} \\
    d_{ic} + d_{kc} &= D_{ik} \\
    d_{jc} + d_{kc} &= D_{jk}
\end{align*}$

Trees with > 3 Leaves

• A tree with $n$ leaves has $2n-3$ edges

• This means fitting a given tree to a distance matrix $D$ requires solving a system of “n choose 2” equations with 2$n$-3 variables

• This is not always possible to solve for $n > 3$

Additive Distance Matrices

Matrix $D$ is ADDITIVE if there exists a tree $T$ with $d_{ij}(T) = D_{ij}$

NON-ADDITIVE otherwise
Distance Based Phylogeny Problem

• Goal: Reconstruct an evolutionary tree from a distance matrix
• Input: n x n distance matrix \( D_{ij} \)
• Output: weighted tree \( T \) with \( n \) leaves fitting \( D \)

• If \( D \) is additive, this problem has a solution and there is a simple algorithm to solve it

Using Neighboring Leaves to Construct the Tree

• Find neighboring leaves \( i \) and \( j \) with parent \( k \)
• Remove the rows and columns of \( i \) and \( j \)
• Add a new row and column corresponding to \( k \), where the distance from \( k \) to any other leaf \( m \) can be computed as:

\[
D_{km} = \frac{D_{im} + D_{jm} - D_{ij}}{2}
\]

Finding Neighboring Leaves

• Closest leaves aren’t necessarily neighbors
• \( i \) and \( j \) are neighbors, but \( (d_{ij} = 13) > (d_{jk} = 12) \)

• Finding a pair of neighboring leaves is a nontrivial problem!

Degenerate Triples

• A degenerate triple is a set of three distinct elements \( 1 \leq i,j,k \leq n \) where \( D_{ij} + D_{jk} = D_{ik} \)
• Element \( j \) in a degenerate triple \( i,j,k \) lies on the path from \( i \) to \( k \) (or is attached to this path by an edge of length 0).

• If distance matrix \( D \) has a degenerate triple \( i,j,k \) then \( j \) can be “removed” from \( D \) thus reducing the size of the problem.

• If distance matrix \( D \) does not have a degenerate triple \( i,j,k \), one can “create” a degenerative triple in \( D \) by shortening all hanging edges (in the tree).
Shortening Hanging Edges to Produce Degenerate Triples

- Shorten all “hanging” edges (edges that connect leaves) until a degenerate triple is found

Finding Degenerate Triples

- If there is no degenerate triple, all hanging edges are reduced by the same amount $\delta$, so that all pair-wise distances in the matrix are reduced by $2\delta$.
- Eventually this process collapses one of the leaves (when $\delta = \text{length of shortest hanging edge}$), forming a degenerate triple $i,j,k$ and reducing the size of the distance matrix $D$.
- The attachment point for $j$ can be recovered in the reverse transformations by saving $D_{ij}$ for each collapsed leaf.

Reconstructing Trees for Additive Distance Matrices

Additive Phylogeny Algorithm

1. AdditivePhylogeny($D$)
2. if $D$ is a 2 x 2 matrix
3. $T = \text{tree of a single edge of length } D_{1,2}$
4. return $T$
5. if $D$ is non-degenerate
6. $\delta = \text{trimming parameter of matrix } D$
7. for all $1 \leq i \neq j \leq n$
8. $D_{ij} = D_{ij} - 2\delta$
9. else
10. $\delta = 0$
AdditivePhylogeny (cont’d)

1. Find a triple $i, j, k$ in $D$ such that $D_{ij} + D_{jk} = D_{ik}$
2. $x = D_{ij}$
3. Remove $i^th$ row and $j^th$ column from $D$
4. $T = \text{AdditivePhylogeny}(D)$
5. Add a new vertex $v$ to $T$ at distance $x$ from $i$ to $k$
6. Add $j$ back to $T$ by creating an edge $(v, j)$ of length 0
7. for every leaf $l$ in $T$
8. if distance from $l$ to $v$ in the tree $\neq D_{ij}$
9. output “matrix is not additive”
10. return
11. Extend all “hanging” edges by length $\delta$
12. return $T$

The Four Point Condition

- AdditivePhylogeny provides a way to check if distance matrix $D$ is additive

- An even more efficient additivity check is the “four-point condition”

- Let $1 \leq i, j, k, l \leq n$ be four distinct leaves in a tree

The Four Point Condition (cont’d)

Compute: 1. $D_{ij} + D_{kl}$, 2. $D_{ik} + D_{jp}$, 3. $D_{il} + D_{jk}$

$2$ and $3$ represent the same number: the length of all edges + the middle edge (it is counted twice)

$1$ represents a smaller number: the length of all edges – the middle edge

The Four Point Condition: Theorem

- The four point condition for the quartet $i, j, k, l$ is satisfied if two of these sums are the same, with the third sum smaller than these first two

- **Theorem**: An $n \times n$ matrix $D$ is additive if and only if the four point condition holds for every quartet $1 \leq i, j, k, l \leq n$
Least Squares Distance Phylogeny Problem

• If the distance matrix $D$ is NOT additive, then we look for a tree $T$ that approximates $D$ the best:

  \[ \text{Squared Error} : \sum_{i,j} (d_{ij}(T) - D_{ij})^2 \]

  • Squared Error is a measure of the quality of the fit between distance matrix and the tree; we want to minimize it.

• **Least Squares Distance Phylogeny Problem**: finding the best approximation tree $T$ for a non-additive matrix $D$ (NP-hard).

Neighbor Joining Algorithm

• In 1987 Naruya Saitou and Masatoshi Nei developed a neighbor joining algorithm for phylogenetic tree reconstruction

  • Finds a pair of leaves that are close to each other but far from other leaves: implicitly finds a pair of neighboring leaves

  • Advantages: works well for additive and other non-additive matrices, it does not have the flawed molecular clock assumption (see UPGMA).

Neighbor Joining

• Tree modified by joining pairs of sequences

  • Pair is chosen by calculating sum of branch lengths, $S$, for the corresponding tree (joining $m$ and $n$; $i$ are the other nodes); $d_{ij}$ are the distance matrix values.

  \[ S_{mn} = \frac{\sum d_{mn} + d_{in}}{2(N-2)} \]

  \[ = \frac{\sum d_{mn}}{2} \]

  \[ = \frac{\sum d_{ij}}{N-2} \]

  (If A and B are joined):

  \[ (A, B) \rightarrow (C, D) \]

  \[ (A, B) \rightarrow (C, D) \]

  \[ (A, B) \rightarrow (C, D) \]
**Neighbour Joining Algorithm**

- Identify i,j as neighbours if their “distance” is the shortest.
- Combine i,j into a new node u.
- Update the distance matrix.
- Distance of u from the rest of the tree is calculated.
- If only 3 nodes are left – finish.

**Why does using $S_{ij}$ give us $O(n^3)$ complexity?**

1. If $N$ represents the number of leaves at each stage, we compute $S_{12}, S_{13}, S_{14}, \ldots, S_{23}, \ldots, S_{(N-1,N)}$ which about $N^2$ computations.
2. We have $N$ stages (we start off with a matrix of $N \times N$, and at each stage the matrix is reduced by 1) → so we’ve reached $N \times N^2 = N^3$.
3. Each $S_{ij}$ we compute, requires us to sum over all of the elements in the matrix – once again, $N^2$ computations, so now we’ve reached $N \times N^2 \times N^2 = N^5$.

**Why does $M_{ij}$ give us complexity of $O(N^3)$?**

In $M_{ij}$ we only have to evaluate $r_i$ and $r_j$ each round. This can be achieved in $O(1)$, if we compute these terms once at the beginning of the round.

Thus, if we return to the list that built the complexity of $S_{ij}$:

- Stage 1 and 2 remain with the same complexity → $O(N^3)$.
- Stage 3 is reduced to $O(1)$, and thus we get a total of $O(N^3)$. 
UPGMA: Unweighted Pair Group Method with Arithmetic Mean

- UPGMA is a clustering algorithm that:
  - Computes the distance between clusters using average pairwise distance
  - Assigns a height to every vertex in the tree, effectively assuming the presence of a molecular clock and dating every vertex
  - The algorithm produces an ultrametric tree: the distance from the root to any leaf is the same (this corresponds to a constant molecular clock: leaves in the tree are assumed to accumulate mutations and thus evolve) at the same rate.

Clustering in UPGMA

Given two disjoint clusters $C_i, C_j$ of sequences,

$$d_{ij} = \frac{1}{|C_i| \times |C_j|} \sum_{p \in C_i, q \in C_j} d_{pq}$$

Note that if $C_k = C_i \cup C_j$, then the distance to another cluster $C_l$ is:

$$d_{kl} = \frac{|C_i| \times d_{il} + |C_j| \times d_{jl}}{|C_i| + |C_j|}$$

UPGMA Algorithm

**Initialization:**
- Assign each $x_i$ to its own cluster $C_i$
- Define one leaf per sequence, each at height 0

**Iteration:**
- Find two clusters $C_i$ and $C_j$ such that $d_{ij}$ is min
- Let $C_k = C_i \cup C_j$
- Add a vertex connecting $C_i, C_j$ and place it at height $d_{ij}/2$
- Delete $C_i$ and $C_j$

**Termination:**
- When a single cluster remains

Weakness
BioInformatics 7: Likelihood for a tree

Aligned sequences for 4 taxa; What is the prob that this tree generated the data?

(A) 1 2 3
(1) C ... G G A C A C T T A ... C
(2) ... A G A C A C T C T A ... C
(3) C ... G G A T A G T T A A ... C
(4) C ... G G A T A T C C T A G ... C

Calculating L for a tree

- Root the tree at any internal node (models are time-reversible)
- Assumption of independence allows to calculate L for each site separately
- Then combine the likelihoods into a total value at the end
- To calculate L for some site j, we must consider all possible scenarios by which the tip sequences could have evolved; Specifically, the root (6) may have had A, C, T, or G.
- For each of these possibilities, the other internal node (5) also might have possessed any of the 4 nucleotides

(B) (1) (3)
(2)
(4)

(D)  
\[ L_{ij} = \text{Prob} \left( \begin{array}{c} C \ A \\ C \ G \end{array} \right) + \text{Prob} \left( \begin{array}{c} C \ A \\ C \ G \end{array} \right) \]
\[ + \cdots + \text{Prob} \left( \begin{array}{c} C \ A \\ C \ G \end{array} \right) \]

(E) \[ L = L_{(1)} \cdot L_{(2)} \cdots L_{(N)} = \prod_{j=1}^{N} L_{(j)} \]
Calculating $L$ for a tree

- Because the probability of any single observation is an extremely small number, we evaluate the log of the likelihood instead.
- Probabilities are accumulated as the sum of logs of the single-site likelihoods.

\[
\log L = \log L(1) + \log L(2) + \ldots + \log L(N) = \sum_{j=1}^{N} \log L(j)
\]

Typical assumptions of ML substitution models

- The probability of any change is independent of the prior history of the site (a Markov Model).
- Substitution probabilities do not change with time or over the tree (a homogeneous Markov process).
- Change is time reversible e.g. the rate of change of A to T is the same as T to A.

Bootstrapping to get the best trees

Main outline of algorithm:
1. Select random columns from a multiple alignment – one column can then appear several times.
2. Build a phylogenetic tree based on the random sample from (1).
3. Repeat (1), (2) many (say, 1000) times.
4. Output the tree that is constructed most frequently.

Jackknifing: Similar to bootstrapping: Generates a number of randomized data sets that are sampled without replacements -> each data set is smaller than the original.
If few positions tipped the balance between one topology and another, different topologies will appear as each replicate dataset is evaluated.

Nearest Neighbor Interchange
- A Branch Swapping algorithm
- Only evaluates a subset of all possible trees
- Defines a neighbor of a tree as one reachable by a nearest neighbor interchange
  - A rearrangement of the four subtrees defined by one internal edge
  - Only three different rearrangements per edge
- Start with an arbitrary tree and check its neighbors
- Move to a neighbor if it provides the best improvement in parsimony score
- No way of knowing if the result is the most parsimonious tree
- Could be stuck in local optimum

Stepwise addition
- Given a text composed from an alphabet of 32 letters (each letter equally probable)
- Person A chooses a letter X (randomly)
- Person B wants to know this letter
- B may ask only binary questions
- Question: how many binary questions must B ask in order to learn which DNA base was chosen by A?
- Answer: entropy H(X), Here: H(X) = 5 bit

BioInformatics 8: Information theory
- How many binary questions must person B ask in order to learn which DNA base was chosen by person A?
- Purines
- Pyrimidines
- 1 bit
Conditional entropy

- Given a text composed from an alphabet of 32 letters (each letter equally probable)
- Person A chooses a letter X (randomly)
- Person B wants to know this letter
- B may ask only binary questions
- A may tell B the letter Y preceding X
- Question: how many binary questions must B ask in order to learn which letter X was chosen by A
- Answer: conditional entropy \( H(X|Y) \); \( H(X|Y) \leq H(X) \)
- In worst case – namely if B ignores all “information” in Y about X – B needs \( H(X) \) binary questions
- Under no circumstances should B need more than \( H(X) \) binary questions

Mutual information

Compare two situations:
- I: learn X without knowing Y
- II: learn X with knowing Y
- How many binary questions in case of I? \( H(X) \)
- How many binary questions in case of II? \( H(X|Y) \)

- Question: How many binary questions could B save in case of II?
- Question: How many binary questions could B save by knowing Y?
- Answer: \( I(X;Y) = H(X) - H(X|Y) \) where \( I(X;Y) \) = information in Y about X
- \( H(Y|X) \leq H(X) \rightarrow I(X;Y) \geq 0 \)
- Example 1: random sequence composed of A, C, G, T (equally probable)
- \( H(X) = 2 \) bit; \( H(X|Y) = 2 \) bit; \( I(X;Y) = H(X) - H(X|Y) = 0 \) bit
- Example 2: deterministic sequence … ACGT ACGT ACGT ACGT …
- \( H(X) = 2 \) bit; \( H(X|Y) = 0 \) bit; \( I(X;Y) = H(X) - H(X|Y) = 2 \) bit

Identifying Motifs and generating Motif Logo

- Genes are turned on or off by regulatory proteins;
- These proteins bind to a short DNA sequence called a motif (TFBS)
- So finding the same motif in multiple genes’ regulatory regions suggests a regulatory relationship amongst those genes
- Motifs can mutate on non important bases
- The five motifs in five different genes have mutations in position 3 and 5
- Representations called motif logos illustrate the conserved and variable regions of a motif

Information Content of a DNA Motif

Information at position \( j \): \( I_j = H_{\text{before}} - H_{\text{after}} \)

Motif probabilities: \( p_k \) (\( k = A, C, G, T \))

Background probabilities: \( q_k = \frac{1}{4} \) (\( k = A, C, G, T \))

\[
I_j = -\sum_{i=1}^{4} q_k \log q_k - \sum_{i=1}^{4} p_k \log p_i = 2 - H_j
\]

\( I_{\text{motif}} = \sum_{j=1}^{w} I_j = 2w - H_{\text{motif}} \) (motif of width \( w \) bases)

Log base 2 gives entropy/information in ‘bits’
### Entropy estimation of alignment

- Define frequencies for the occurrence of each letter in each column of multiple alignment
  
  \[ p_A = 1, \ p_T = p_G = p_C = 0 \] (1st column)
  
  \[ p_A = 0.75, \ p_T = 0.25, \ p_G = p_C = 0 \] (2nd column)
  
  \[ p_A = 0.50, \ p_T = 0.25, \ p_C = 0.25, \ p_G = 0 \] (3rd column)

- Compute entropy of each column
  
  \[ -\sum_{X=A,T,G,C} p_X \log p_X \]

### Information Content

- In a positional weight matrix, PWM, convert frequencies to probabilities
- PWM \( W \): \( W_{\beta k} \) = frequency of base \( \beta \) at position \( k \)
- \( q_\beta \) = frequency of base \( \beta \) by chance
- Information content of \( W \):
  
  \[
  \sum_k \sum_{\beta \in \{A,C,G,T\}} W_{\beta k} \log \frac{W_{\beta k}}{q_\beta}
  \]

- If \( W_{\beta k} \) is always equal to \( q_\beta \), i.e., if \( W \) is similar to random sequence, information content of \( W \) is 0.
- If \( W \) is different from \( q \), information content is high.
Entropy of an Alignment: Example

_column entropy:_

\[-(p_A \log p_A + p_C \log p_C + p_G \log p_G + p_T \log p_T)\]

- Column 1 = \[-(1 \times \log(1) + 0 \times \log(0) + 0 \times \log(0) + 0 \times \log(0))\]
  = 0

- Column 2 = \[-(\frac{1}{4} \times \log(\frac{1}{4}) + \frac{3}{4} \times \log(\frac{3}{4}) + 0 \times \log(0) + 0 \times \log(0))\]
  = \[-(\frac{1}{4} \times (-2) + \frac{3}{4} \times (-0.415))\]
  = +0.811

- Column 3 = \[-(\frac{1}{4} \times \log(\frac{1}{4}) + \frac{1}{4} \times \log(\frac{1}{4}) + \frac{1}{4} \times \log(\frac{1}{4}) + \frac{1}{4} \times \log(\frac{1}{4}))\]
  = 4 \times \[-(\frac{1}{4} \times (-2))\]
  = +2.0

- Alignment Entropy = 0 + 0.811 + 2.0 = +2.811

Splice Sites

- Donor site:
  - start of intron
  - consensus GT
  - also called 5' splice site

- Acceptor site:
  - end of intron
  - consensus AG
  - also called 3' splice site

- Introns can be inserted in the middle of a codon!

Recognize splice sites

Position-specific scoring matrix

\[S = S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9\]

Odds Ratio

\[R = \frac{P(S+) \times P(S) \times P(S_1) \times \cdots \times P(S_9) \times P(S_1)}{P(S-) \times P(S) \times P(S_1) \times \cdots \times P(S_9) \times P(S_1)}\]

Score

\[s = \log_2 R\]
Motifs: Profiles and Consensus

- Line up the patterns by their start indexes
  \[ S = (s_1, s_2, ..., s_j) \]
- Construct matrix profile with frequencies of each nucleotide in columns
- Consensus nucleotide in each position has the highest score in column
- Think of consensus as an "ancestor" motif, from which mutated motifs emerged
- The distance between a real motif and the consensus sequence is generally less than that for two real motifs

Predicting the number of sites

Association between adjacent bases will lead to association between more distant bases, and an estimate of how far the relations extend may be found from Markov Chain theory.

Without invoking any biological mechanism, a Markov chain of order k supposes that the base present at a certain position in a sequence depends only on the bases present at the previous k positions.

\[
\begin{align*}
  p(G|GATC) &= p(G|GATC) \\
  p(G|GATC) &= p(G|GATC) \\
  p(G|GATC) &= p(G|GATC) \\
  p(G|GATC) &= p(G|GATC)
\end{align*}
\]

For a zero order Markov chain we estimate the frequency of a word from base composition alone.

For a first order Markov chain model can be used to estimate the same frequency.

For a second order Markov chain that uses di and tri nucleotide frequencies.

For a third order Markov chain.

Gibbs Sampling

- **Gibbs Sampling** is an iterative procedure that discards one l-mer after each iteration and replaces it with a new one.
- Gibbs Sampling proceeds slowly and chooses new l-mers at random increasing the odds that it will converge to the correct solution.

Yeast genome: Frequencies of each hexanucleotide were plotted from highest to lowest abundance along with values determined by each Markov chain.
How Gibbs Sampling Works

1) Randomly choose starting positions 
   \( s \) = \( (s_1, ..., s_5) \) and form the set of \(-mer\)s associated with these starting positions.
2) Randomly choose one of the \( t \) sequences.
3) Create a profile \( p \) from the other \( t-1 \) sequences.
4) For each position in the removed sequence, calculate the probability that the \(-mer\) starting at that position was generated by \( p \).
5) Choose a new starting position for the removed sequence based on the probabilities calculated in step 4.
6) Repeat steps 2-5 until there is no improvement

Gibbs Sampling Algorithm

**Input:**
\( t = 5 \) sequences, motif length \( l = 8 \)
1. GTAAACAATTTTATAGC
2. AAAATTTACCCTCGAAAGG
3. CCGTACTGACGCGTGG
4. TGAGTAAAGACGTCCCA
5. TACTTTAAACCTCTGCAA

**Gibbs Sampling**
1) Randomly choose starting positions, \( s = (s_1, s_2, s_3, s_4, s_5) \) in the 5 sequences:
   \( s_1 = 7 \) GTAAACAATTTTATAGC
   \( s_2 = 11 \) AAAATTTACCCTCGGAAGG
   \( s_3 = 9 \) CCGTACTGACGCGTGG
   \( s_4 = 4 \) TGAGTAAAGACGTCCCA
   \( s_5 = 1 \) TACTTTAAACCTCTGCAA

For each position, \( r_i \) in the omitted sequence, \( s_i \), calculate a weight:
\[
\prod_{k=r_i}^{r_i+W-1} \prod_{i=1}^{l} \rho_{i,j}^{(s_i,j)} \prod_{k=r_i}^{r_i+W-1} \prod_{i=1}^{l} \rho_{i,0}^{(s_i,j)}
\]

i.e. the probability of motif to background score

New motif location in \( s_i \) is chosen according to these weights. That is, instead of giving each position in the sequence equal weight so that each position has a \( \frac{1}{l} \) chance of being selected, the chance of being selected is proportional to the weight. Large weight (meaning higher chance of the motif begin positioned there) gives large chance of selection.
Gibbs Sampling: an Example

2) Choose one of the sequences at random:

**Sequence 2**: AAAATTTACCTTAGAAGG

3) Create profile \( p \) from \( l \)-mers in remaining 4 sequences:

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

Consensus String: TAAATAC

5) Create a distribution of probabilities of \( l \)-mers \( \text{prob}(b|P) \), and randomly select a new starting position based on this distribution.

a) To create this distribution, divide each probability \( \text{prob}(b|P) \) by the lowest probability:

- Starting Position 1: \( \text{prob}(\text{AAAATTA}|P) = 0.000732 / 0.000122 = 6 \)
- Starting Position 2: \( \text{prob}(\text{AAATTTA}|P) = 0.000122 / 0.000122 = 1 \)
- Starting Position 8: \( \text{prob}(\text{ACCTTAG}|P) = 0.000183 / 0.000122 = 1.5 \)
  
  Ratio = 6 : 1 : 1.5

b) Define probabilities of starting positions according to computed ratios

- Probability (Selecting Starting Position 1): \( 6/(6+1+1.5) = 0.706 \)
- Probability (Selecting Starting Position 2): \( 1/(6+1+1.5) = 0.118 \)
- Probability (Selecting Starting Position 8): \( 1.5/(6+1+1.5) = 0.176 \)

4) Calculate the \( \text{prob}(b|P) \) for every possible 8-mer in the removed sequence:

| String          | \( \text{prob}(b|P) \) |
|-----------------|------------------------|
| AAAATTTACCTTAGA | 0.000732               |
| AAAATTTACCTTAGA | 0.000122               |
| AAAATTTACCTTAGA | 0                      |
| AAAATTTACCTTAGA | 0.000183               |
| AAAATTTACCTTAGA | 0                      |
| AAAATTTACCTTAGA | 0                      |
| AAAATTTACCTTAGA | 0                      |
| AAAATTTACCTTAGA | 0                      |

5) Create a distribution of probabilities of \( l \)-mers \( \text{prob}(b|P) \), and randomly select a new starting position based on this distribution.

6) We iterate the procedure again with the above starting positions until we cannot improve the score any more.
BioInformatics 9
The dishonest casino model

A Hidden Markov model is Memoryless:

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)

\[
P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots, \pi_N) = \\
P(x_N, \pi_N | \pi_{N-1}) P(x_{N-1}, \pi_{N-1} | \pi_{N-2}) \ldots P(x_2, \pi_2 | \pi_1) P(x_1, \pi_1) \\
= a_{01} a_{12} \ldots a_{N-1,N} e_{1}(x_1) \ldots e_{N}(x_N)
\]
The three main questions on HMMs

1. Evaluation
GIVEN a HMM \( M \), and a sequence \( x \),
FIND \( \text{Prob}(x \mid M) \)

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

3. Learning
GIVEN a HMM \( M \), with unspecified transition/emission probs.,
and a sequence \( x \),
FIND parameters \( \theta = (e_i(.), a_{ij}) \) that maximize \( P(x \mid \theta) \)

Let’s not be confused by notation

\[ P(x \mid M) : \text{The probability that sequence } x \text{ was generated by the model; The model is: architecture (#states, etc) + parameters } \theta = (e_i(.),) \]

So, \( P(x \mid \theta) \), and \( P(x) \) are the same, when the architecture, and the entire model, respectively, are implied

Similarly, \( P(x, \pi \mid M) \) and \( P(x, \pi) \) are the same

In the LEARNING problem we always write \( P(x \mid \theta) \) to emphasize that we are seeking the \( \theta \) that maximizes \( P(x \mid \theta) \)

Decoding

GIVEN \( x = x_1x_2\ldots x_N \)

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

\[ \pi^* = \arg\max_{\pi} P(x, \pi) \]

We can use dynamic programming!

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

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

Decoding – main idea

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

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

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

From definition,

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

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

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

= \( e_i(x_{i+1}) \max_{\pi_i} a_{ik} V_i(i) \)
The Viterbi Algorithm

Input:

\( x = x_1 \ldots x_n \)

Initialization:

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

\( V_j(0) = 0, \text{ for all } k > 0 \)

Iteration:

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

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

Termination:

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

Traceback:

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

\[\pi^*_{i-1} = \text{Ptr}_{\pi^*_i}(i)\]

Generating a sequence by the model

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

1. Start at state \( \pi_1 \) according to prob \( a_{0\pi_1} \)
2. Emit letter \( x_1 \) according to prob \( e_{\pi_1}(x_1) \)
3. Go to state \( \pi_2 \) according to prob \( a_{\pi_1\pi_2} \)
4. ... until emitting \( x_n \)

A couple of questions

Given a sequence \( x \),

- What is the probability that \( x \) was generated by the model?
- Given a position \( i \), what is the most likely state that emitted \( x_i \)?

Example: the dishonest casino

Say \( x = 12341623162616364616234161221341 \)

Most likely path: \( \pi = FF \ldots F \)
However: marked letters more likely to be L than unmarked letters
Evaluation
We will develop algorithms that allow us to compute:

- \( P(x) \) \begin{align*}
& \text{Probability of } x \text{ given the model} \\
& \text{Probability of } \pi_i = k \mid x \text{ given } x \\
& \text{Probability that the } i\text{th state is } k, \text{ given } x
\end{align*}

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

The Forward Algorithm
We want to calculate

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

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

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

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

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

The Forward Algorithm – derivation
Define the forward probability:

\[
f_k(i) = P(x_1...x_i, \pi_i = l) \quad \text{for all } k, \ i
\]

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

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

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

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

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

Iteration:
\[
f_k(i) = e_i(x_i) \sum_k f_k(i-1) a_{ki}
\]

Termination:
\[
P(x) = \sum_k f_k(N) a_{k0}
\]

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

**VITERBI**
- **Initialization:**
  \[ V_0(0) = 1 \]
  \[ V_k(0) = 0, \text{ for all } k > 0 \]
- **Iteration:**
  \[ V(i) = e_i(x_i) \max_k V_k(i-1) a_{ij} \]
- **Termination:**
  \[ P(x, \pi^*) = \max_k V_k(N) \]

**FORWARD**
- **Initialization:**
  \[ f_0(0) = 1 \]
  \[ f_k(0) = 0, \text{ for all } k > 0 \]
- **Iteration:**
  \[ f(i) = e_i(x_i) \sum_k f(i-1) a_{ij} \]
- **Termination:**
  \[ P(x) = \sum_k f_k(N) a_{kj} \]

**Motivation for the Backward Algorithm**
We want to compute
\[ P(\pi_i = k | x) \]
the probability distribution on the \(i\)th position, given \(x\)

We start by computing
\[ P(\pi_i = k, x) = P(x_{1i}, x_{i+1}\ldots x_N, \pi_i = k) \]
\[ = P(x_{1i}, x_{i+1}\ldots x_N | x_{1i}=\pi_i, x_{i+1}\ldots x_N, \pi_i = k) \]
\[ = P(x_{1i}, \pi_i = k) P(x_{i+1}\ldots x_N | \pi_i = k) \]
\[ = P(x_{1i}, \pi_i = k) P(x_{i+1}\ldots x_N | \pi_i = k) \]

\[ \text{Forward, } f(i) \]
\[ \text{Backward, } b(i) \]

**The Backward Algorithm – derivation**
Define the backward probability:
\[ b_k(i) = P(x_{i+1}\ldots x_N | \pi_i = k) \]
\[ = \sum_{\pi_{i+1}\ldots \pi_N} P(x_{i+1}, \pi_{i+1}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N | \pi_i = k) \]
\[ = \sum_{l} \sum_{\pi_{i+1}\ldots \pi_N} P(x_{i+2}, \pi_{i+2}, \ldots, x_N, \pi_{i+1}, \pi_{i+2}, \ldots, \pi_N | \pi_i = k) \]
\[ = \sum_{l} e_i(x_{i+1}) a_{il} \sum_{\pi_{i+1}\ldots \pi_N} P(x_{i+2}, \pi_{i+2}, \ldots, x_N, \pi_{i+1}, \pi_{i+2}, \ldots, \pi_N | \pi_i = l) \]
\[ = \sum_{l} e_i(x_{i+1}) a_{il} b_l(i+1) \]

**The Backward Algorithm**
We can compute \(b_k(i)\) for all \(k, i\), using dynamic programming
- **Initialization:**
  \[ b_k(N) = a_{kd}, \text{ for all } k \]
  What is the running time, and space required, for Forward, and Backward?
  - **Time:** \(O(K^2N)\)
  - **Space:** \(O(KN)\)
- **Iteration:**
  \[ b_k(i) = \sum_{l} e_i(x_{i+1}) a_{il} b_l(i+1) \]
- **Termination:**
  \[ P(x) = \sum_{l} a_{0i} e_i(x_{1}) b_l(1) \]
Assume we are given a DNA sequence that begins in an exon, contains one splice site and ends in an intron. The problem is to identify where the switch from exon to intron occurred.

For us to guess intelligently, the sequences of exons, splice sites and introns must have different statistical properties.

Let’s imagine some simple differences: say that exons have a uniform base composition on average (25% each base), introns are A+T rich (say, 40% each for A/T, 10% each for C/G), and the 5'SS consensus nucleotide is almost always a G (say, 95% G and 5% A).

Starting from this information, we can draw an HMM that invokes three states, one for each of the three labels we might assign to a nucleotide: E (exon), 5 (5'SS) and I (intron).

How confident are we that the fifth G is the right choice?

Our confidence will depend on posterior decoding.

Posterior decoding uses two dynamic programming algorithms called Forward and Backward, which have some similarity with Viterbi, but they sum over possible paths instead of choosing the best.
Genescan model

- Duration of states – length distributions of
  - Exons (coding)
  - Introns (non-coding)
- Signals at state transitions
  - ATG Codon for gene start
  - Stop Codon TAG/TGA/TA
  - Exon/Intron and Intron/Exon Splice Sites
- Emissions
  - Coding potential and frame at exons
  - Intron emissions

GenScan

- N - intergenic region
- P - promoter
- F - 5' untranslated region
- \( E_{\text{rel}} \) – single exon (intronless) (translation start -> stop codon)
- \( E_{\text{ini}} \) – 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
Assessing performance: Sensitivity and Specificity

- Testing of predictions is performed on sequences where the gene structure is known
- **Sensitivity** is the fraction of known genes (or bases or exons) correctly predicted: \(Sn = \frac{\text{True Positives}}{\text{All True}}\)
  - “Am I finding the things that I’m supposed to find?”
- **Specificity** is the fraction of predicted genes (or bases or exons) that correspond to true genes: \(Sp = \frac{\text{True Positives}}{\text{All Positives}}\)
  - “What fraction of my predictions are true?”
- In general, increasing one decreases the other

<table>
<thead>
<tr>
<th>Method</th>
<th>Sn</th>
<th>Sp</th>
<th>Ac</th>
<th>ME</th>
<th>WE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>0.686</td>
<td>0.724</td>
<td>0.667</td>
<td>0.197</td>
<td>0.173</td>
</tr>
<tr>
<td>OMA</td>
<td>0.638</td>
<td>0.690</td>
<td>0.650</td>
<td>0.225</td>
<td>0.183</td>
</tr>
<tr>
<td>GENE</td>
<td>0.745</td>
<td>0.759</td>
<td>0.721</td>
<td>0.155</td>
<td>0.231</td>
</tr>
<tr>
<td>TRIBE</td>
<td>0.776</td>
<td>0.759</td>
<td>0.750</td>
<td>0.173</td>
<td>0.173</td>
</tr>
</tbody>
</table>

Sn = Sensitivity
Sp = Specificity
Ac = Approximate Correlation
ME = Missing Exons
WE = Wrong Exons

**Correlation Coefficient**

\[
CC = \frac{(TP)(TN) - (FP)(FN)}{\sqrt{(TN)(FP)(AP)(PN)}}
\]

\[
AN = TN + FP, AP = TP + FN, PP = TP + FP, PN = TN + FN
\]

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 of a transmembrane helix, an intracellular loop or globular domain region, or an extracellular loop or domain region.

TMHMM: uses cyclic model with 7 states for
- TM helix core
- TM helix caps on the N- and C-terminal side
- non-membrane region on the cytoplasmic side
- 2 non-membrane regions on the non-cytoplasmic side (for short and long loops to account for different membrane insertion mechanisms)
- a globular domain state in the middle of each non-membrane region
The transitions from state 3 to non-adjacent states model the length distribution of trans-membrane helices.

**BioInformatics 10: microarray**

Microarrays measure the activity (expression level) of the genes under varying conditions/time points; expression level is estimated by measuring the amount of mRNA for that particular gene; a gene is active if it is being transcribed; more mRNA usually indicates more gene activity.

Millions of identical probes per feature (25 base-long single-strand DNA)
Green = Expression level low with respect to reference sample.
Red = Expression level high with respect to reference sample.
Black = Expression level comparable to reference sample.
The columns are ordered such that similar expression profiles neighbor each other.

K-Means Clustering: Lloyd Algorithm

1. **Lloyd Algorithm**
2. Arbitrarily assign the \( k \) cluster centers
3. while the cluster centers keep changing
4. Assign each data point to the cluster \( C_i \) corresponding to the closest cluster representative (center) \( (1 \leq i \leq k) \)
5. After the assignment of all data points, compute new cluster representatives according to the center of gravity of each cluster, that is, the new cluster representative is
   \[ \sum v \setminus |C| \quad \text{for all} \ v \text{ in } C \quad \text{for every cluster } C \]

*This may lead to merely a locally optimal clustering.

K-Means Clustering Problem: Formulation

- **Input**: A set, \( V \), consisting of \( n \) points and a parameter \( k \)
- **Output**: A set \( X \) consisting of \( k \) points (cluster centers) that minimizes the squared error distortion \( d(V,X) \) over all possible choices of \( X \)

1-Means Clustering Problem: an Easy Case

- **Input**: A set, \( V \), consisting of \( n \) points
- **Output**: A single points \( x \) (cluster center) that minimizes the squared error distortion \( d(V,x) \) over all possible choices of \( x \)

1-Means Clustering problem is easy. However, it becomes very difficult (NP-complete) for more than one center.

An efficient heuristic method for K-Means clustering is the Lloyd algorithm.
Conservative K-Means Algorithm

- Lloyd algorithm is fast but in each iteration it moves many data points, not necessarily causing better convergence.
- A more conservative method would be to move one point at a time only if it improves the overall clustering cost
  - The smaller the clustering cost of a partition of data points is the better that clustering is
  - Different methods (e.g., the squared error distortion) can be used to measure this clustering cost

K-Means “Greedy” Algorithm

1. ProgressiveGreedyK-Means(k)
2. Select an arbitrary partition P into k clusters
3. while forever
4. bestChange $\leftarrow$ 0
5. for every cluster C
6. for every element i not in C
7. if moving i to cluster C reduces its clustering cost
8. if (cost(P) - cost($P_{i \rightarrow C}$)) > bestChange
9. bestChange $\leftarrow$ cost(P) - cost($P_{i \rightarrow C}$)
10. $i^*$ $\leftarrow$ i
11. $C^*$ $\leftarrow$ C
12. if bestChange > 0
13. Change partition P by moving $i^*$ to $C^*$
14. else
15. return P

Squared Error Distortion

- Given a data point v and a set of points X, define the distance from v to X as the (Euclidean) distance from v to the closest point from X.
- Given a set of n data points $V = \{v_1, \ldots, v_n\}$ and a set of k points X, define the Squared Error Distortion $d(V, X) = \sum_{i=1}^{n} d(v_i, X)^2 / n$.
Clustering Affinity Search Technique (CAST)-1

Affinity = a measure of similarity between a gene, and all the genes in a cluster.
Threshold affinity = user-specified criterion for retaining a gene in a cluster, defined as %age of maximum affinity at that point

1. Create a new empty cluster C1.
2. Set initial affinity of all genes to zero
3. Move the two most similar genes into the new cluster.

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>G6</th>
<th>G7</th>
<th>G8</th>
<th>G9</th>
<th>G10</th>
<th>G11</th>
<th>G12</th>
<th>G13</th>
<th>G14</th>
<th>G15</th>
</tr>
</thead>
</table>

4. Update the affinities of all the genes (new affinity of a gene = its previous affinity + its similarity to the gene(s) newly added to the cluster C1)

| Current cluster C1 |

ADD GENES:
5. While there exists an unassigned gene whose affinity to the cluster C1 exceeds the user-specified threshold affinity, pick the unassigned gene whose affinity is the highest, and add it to cluster C1. Update the affinities of all the genes accordingly.

REMOVE GENES:
6. When there are no more unassigned high-affinity genes, check to see if cluster C1 contains any elements whose affinity is lower than the current threshold. If so, remove the lowest-affinity gene from C1. Update the affinities of all genes by subtracting from each gene's affinity, its similarity to the removed gene.
7. Repeat step 6 while C1 contains a low-affinity gene
8. Repeat steps 5-7 as long as changes occur to the cluster C1.
9. Form a new cluster with the genes that were not assigned to cluster C1, repeating steps 1-8.
10. Keep forming new clusters following steps 1-9, until all genes have been assigned to a cluster.

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.

Unlike most clustering algorithms, the MCL does not require the number of expected clusters to be specified beforehand.
The basic idea underlying the algorithm is that dense clusters correspond to regions with a larger number of paths.

A random walk 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.

The algorithm iterates three steps.
Given a network with n vertexes, it takes the corresponding n × n adjacency matrix A and normalises each column to obtain a stochastic matrix M. It takes the \( X_k \) power \( M^k \) of this matrix (expansion) and then the \( X_n \) power \( M^n \) of every element (inflation).
The expansion parameter \( k \) is often taken equal to 2, while the granularity of the clustering is controlled by tuning the inflation parameter \( r \).
Principle Components Analysis (PCA)

A sample of \( n \) observations in the 2-D space \( X = (X_1, X_2) \)

Goal: to account for the variation in a sample in as few variables as possible, to some accuracy

- the 1st PC \( Z_1 \) is a minimum distance fit to a line \( X \) in space
- the 2nd PC \( Z_2 \) is a minimum distance fit to a line in the plane perpendicular to the 1st PC

PCAs are a series of linear least squares fits to a sample, each orthogonal to all the previous.

**PCA - Steps**

Input: a dataset \( S = \{s^i,...,s^n\} \), \( s^i = \{x_1^i,...,x_d^i\} \)

- Subtract the mean from each dimension

- Compute the covariance matrix \( \Sigma \) for the \( d \) dimensions
  - The covariance of two variables \( X \) and \( Y \):
    \[
    \text{cov}(X, Y) = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})}{(n-1)}
    \]
  - The covariance matrix:
    \[
    \Sigma(X, Y) = \Sigma(Y, X) = \text{cov}(X, Y)
    \]

**PCA – Steps (cont.)**

- Compute the eigenvectors and eigenvalues of the covariance matrix

- Choose the most informative PCs, construct a feature vector
  - Eigenvectors with highest eigenvalues carry the most information
  - Feature vector is simply the combination of all eigenvectors chosen
    \[
    \text{FeatureVector} = (eig_1, eig_2, ..., eig_d)
    \]

- Transform dataset to the new axis system
  - For \( s \in S \):
    \[
    s' = \text{FeatureVector}^T \times s = \begin{bmatrix} eig_1 & eig_2 & \cdots & eig_d \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}
    \]
When Things Get Messy...

- PCA is fine when initial dimension is not too big
  - Space and time complexity are of $O(d^2)$ - size of covariance matrix

- Otherwise – we have a problem...
  - E.g. when $d=10^4$ ⇒ time/space complexity is $O(10^8)$...

- Luckily an alternative exists: SVD

Eigengenes, Eigenarrays and SVD

- The idea:
  - Use the singular value decomposition (SVD) theorem for transforming the dataset from the gene/array space to the eigengene/eigenarray space

- Eigengenes, eigenarrays and eigenvalues:
  - Each dimension is represented by an eigengene/eigenarray/eigenvalue triplet
  - Eigenvalues are used for ranking dimensions

Singular Value Decomposition (SVD)

- Theorem: if $E$ is a real $M$ by $N$ matrix, then there exist orthogonal matrices
  
  $U = [u_1, ..., u_M] \in \mathbb{R}^{M \times M}$ and $V = [v_1, ..., v_N] \in \mathbb{R}^{N \times N}$

  such that

  $E = U \cdot W \cdot V^T$

  Where

  $W = \text{diag}(\sigma_1, ..., \sigma_p)$

  and

  $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p \geq 0, \quad p = \min(m,n)$

SVD

- $\sigma_i$ is the $i^{th}$ singular value of $E$.
  $u_i$ and $v_i$ are the $i^{th}$ left singular vector and right singular vector of $E$, respectively.

- It holds that

  $E \cdot v_i = \sigma_i \cdot u_i$

  $E^T \cdot u_i = \sigma_i \cdot v_i$

  $\min(M, N) = 1 : 1$

- Efficient algorithms for calculating the SVD exist
Orthogonality of Decomposition

\[ E = U \cdot W \cdot V^T \]
\[ V = [v_1, ..., v_k], \quad v_i = [v_i^1, ..., v_i^M] \]
\[ U = [u_1, ..., u_k], \quad u_i = [u_i^1, ..., u_i^N] \]
\[ W = \text{diag}(\sigma_1, ..., \sigma_p) \]

\[ v_i = [v_i^1, ..., v_i^M] \]
\[ u_i = [u_i^1, ..., u_i^N] \]
\[ \sigma_i = \sigma_i^1 \]

\[ U \cdot W \cdot V^T = \sum_{\alpha \in \mathcal{A}} \alpha \cdot u^\alpha \cdot v^\alpha \]

SVD and Microarray analysis

- Reduction from the N genes x M arrays to p eigengenes x p eigenarrays space
  - W is the eigenexpression matrix
  - U represents the expression of genes over eigenarrays
  - V represents the expression of eigengenes over arrays

- The "fraction of eigenexpression":
  \[ p_i = \frac{\sigma_i}{\sum_{j=1}^{p} \sigma_j} \]

- "Shannon entropy" of the dataset:
  \[ 0 < d = -\frac{1}{\log(p)} \sum_{\alpha \in \mathcal{A}} p_\alpha \log(p_\alpha) \leq 1 \]

BioInformatics 11: genetic networks

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

\[ \text{Transcription} \rightarrow \text{MRNA}_B \rightarrow \text{Translation} \rightarrow \text{B} \]
\[ \text{Production of mRNA}_B \text{ is enhanced by E} \]

\[ \text{E only} \]

\[ \text{B is a Primary Target of E} \]
\[ \text{Production of mRNA}_B \text{ is enhanced by E} \]

\[ \text{D is a Secondary Target of E} \]
\[ \text{Production of mRNA}_D \text{ is enhanced by B} \]
• in the presence of both CX and E we see increased expression of mRNA_B but not of mRNA_D
• this will be one of the principles we can use to differentiate between primary targets of E (such as B) and secondary targets of E (such as D)

<table>
<thead>
<tr>
<th>Conditions</th>
<th>mRNA_B</th>
<th>mRNA_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>E</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>CX</td>
<td>Low(?)</td>
<td>Low (?)</td>
</tr>
<tr>
<td>E and CX</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

Perturbation Static Graph Model

• Motivation: perturb a gene network one gene at a time and use the affected genes in order to discriminate direct vs. indirect gene-gene relationships
• Perturbations: gene knockouts, over-expression, etc.
Method:
1. For each gene \( \beta \), compare the control experiment to perturbed experiment and identify the differentially expressed genes
2. Use the most parsimonious graph that yields the graph of 1. as its reachable graph

How to reconstruct a large genetic network from \( n \) gene perturbations in fewer than \( n^2 \) steps

- How can we distinguish between direct and indirect relationships in a network based on microarray data?
- Additional Assumption needed
- Next: minimize # relationships
An example

- (a) gene network
- (b) adjacency list
- (c) accessibility list
- Goal: (c) -> (a)

The figure illustrates three graphs (Figs. B,C,D) with the same accessibility list Acc (Fig. A). There is one graph (Fig. D) that has Acc as its accessibility list and is simpler than all other graphs, in the sense that it has fewer edges. Let’s call Gpars the most parsimonious network compatible with Acc.

Algorithm

- Step1: Graphs without cycles only (acyclic directed graph)
- Step2: Graphs with cycles

- Step 1: Shortcut:

A shortcut-free graph compatible with an accessibility list is a unique graph with the fewest edges among all graphs compatible with the accessibility list, i.e., a shortcut-free graph is the most parsimonious graph.

Step1

- A theorem: Let Acc(G) be the accessibility list and Adj(G) be the adjacency list at an acyclic directed graph, its most parsimonious graph, and V( ) the set of all nodes of . Then the following identity holds

$$\forall i \in V(G_{pars}) \quad Adj(i) = Acc(i) \cup \bigcup_{j \in Acc(i)} Acc(j)$$

In words, for each node i the adjacency list Adj(i) of the most parsimonious genetic network is equal to the accessibility list Acc(i) after removal of all nodes that are accessible from any node in Acc(i).
An example

\[ \text{Adj}(1) = \text{Acc}(1) - (\text{Acc}(2) + \text{Acc}(3) + \text{Acc}(4) + \text{Acc}(5) + \text{Acc}(6)) = (2, 3, 4, 5, 6) - (3 \cup (5, 6)) = (2, 4) \]

The algorithm of step 1

1. for all nodes \(i\) of \(G\) \n2. \(\text{Adj}(i) = \text{Acc}(i)\) \n3. for all nodes \(i\) of \(G\) \n4. if node \(i\) has not been visited \n5. call PRUNE_ACC(i) \n6. end if \n7. \(\text{PRUNE_ACC}(i)\) \n8. for all nodes \(j \in \text{Acc}(i)\) \n9. if \(i \in \text{Adj}(j)\) \n10. declare \(j\) as visited. \n11. else \n12. call PRUNE_ACC(i) \n13. end if \n14. for all nodes \(j \in \text{Acc}(i)\) \n15. for all nodes \(k \in \text{Adj}(j)\) \n16. if \(k \in \text{Acc}(i)\) \n17. delete \(k\) from \(\text{Adj}(i)\) \n18. end if \n19. declare node \(i\) as visited \n20. end PRUNE_ACC(i)

Step 1

- A Corollary: Let \(i, j,\) and \(k\) be any three pairwise different nodes of an acyclic directed shortcut-free graph \(G\). If \(j\) is accessible from \(i\), then no node \(k\) accessible from \(j\) is adjacent to \(i\).

Step 2: How about graphs with cycles?

- Two different cycles have the same accessibility list
- Perturbations of any gene in the cycle influences the activity of all other genes in the same cycle.
- Can’t decide a unique graph if cycle happens
- Not an algorithmic but an experimental limitation
The algorithm of step 2

• Basic idea: Shrink each cycle (strongly connected components) into one node and apply the algorithm of step 1.

• A corollary: Let $i$ and $j$ ($i \neq j$) be two nodes of a directed graph $G$. $i$ and $j$ are in the same component iff $i \in \text{Acc}(j)$ and $j \in \text{Acc}(i)$.

• A graph after shrinking all the cycles into nodes is called a condensation graph.

---

The algorithm of step 2

```plaintext
for all nodes $i$ of $G$
  if component[$i$] has not been defined
    create new node $x$ of $G^*$
    component[$x$] = $i$
  for all nodes $j \in \text{Acc}(i)$
    if $j \in \text{Acc}(i)$
      component[$j$] = $x$
    end if
  end if
for all nodes $j \in \text{Acc}(i)$
  if component[$i$] = component[$j$]
    if component[$i$] = component[$j$]
      add component[$i$] to Acc(component[$i$])
    end if
  end if
end if
```

---

Missing genes and messy data

• Some genes are difficult to perturb

• Problem: some information is missing for certain genes. How well does the algorithm perform in such cases?

• Simulation: Randomly generate graphs with pre-specified nodes and edges. Then eliminate pre-specified fraction of nodes from the accessibility list. Apply the algorithm to both graphs without elimination and with elimination.
Limitation of the algorithm

- Unable to resolve cycled graphs
- Require more data than conventional methods using gene expression correlations.
- There are many networks consistent with the given accessibility list. The algorithm constructs the most parsimonious one.
- The same problem was proposed around 1980 which is called “transitive reduction”.
- The transitive reduction of a directed graph \( G \) is the directed graph \( G' \) with the smallest number of edges such for every path between vertices in \( G \), \( G' \) has a path between those vertices.
- An \( O(V) \) algorithm for computing transitive reduction of a planar acyclic digraph was proposed by Sukhamay Kundu. (\( V \) is the number of nodes in \( G \))

The Gillespie algorithm

A reaction rate \( w_i \) is associated to each reaction step. These probabilities are related to the kinetics constants.
Initial number of molecules of each species are specified.
The time interval is computed stochastically according to the reaction rates.
At each time interval, the reaction that occurs is chosen randomly according to the probabilities \( w_i \), and both the number of molecules and the reaction rates are updated.

BioInformatics lecture 12

System Biology
1. Large scale integration of information on molecules, genes, cell, tissue, organ, organism, health
2. Markup language
   - development of SBML (Systems Biology Markup Language) for representing biochemical networks and CellML for electrophysiology, mechanics, energetics and general pathway. SBML is an XML-based markup language for describing the biochemical network models that arise in Systems Biology.
3. Computational models
   - development of models that are “anatomically based” and “biophysically based” to link gene, protein, cell, tissue, organ and whole body systems physiology.
   Methodologies: differential equations and stochastic algorithms

Gillespie algorithm

Probability that reaction \( r \) occurs

\[ P_r = \frac{w_r}{\sum_{i=1}^{R} w_i} \]

Reaction \( r \) occurs if

\[ P_{r-1} < z_1 \leq P_{r-1} + P_r \]

Time step to the next reaction

\[ \Delta t = \frac{1}{\sum_{i=1}^{R} w_i} ln \frac{1}{z_2} \]

Gillespie algorithm

In practice...

1. Calculate the transition probability \( w_i \) and the variables \( X_i \) (A,B,C etc).
2. Generate \( z_1 \) and \( z_2 \) and calculate the reaction that occurs as well as the time till this reaction occurs.
3. Increase \( t \) by \( \Delta t \) and adjust \( X \) to take into account the occurrence of the reaction that just occurred.