>gi|18089116|gb|BC020718.1| Homo sapiens I factor
AAATTTCAAAAGAATACCTTGAGTGGAAGAAGAGTTCTCAAGCAGAGACAAAGACCCCGAACACCTCCCAACA
TGAAGCTTCTCTCATGTTTTTCTGTATTTTCTGTGCTTCCACTTAAGTTTTTGCAGCTCACTTTATACATC
TCAAGAGGATCTGCTGGAGGAGAAAAATGCTTAGCAGAAAAATATATACCTACCTCTCTCTCTGCCTAGGTA
AGCTCAGCCATGGCAGAGATGCAATGAGGGCAGCTGTGTTTGTAAACTACGAGATCAGTGCAAAAGAATG
GCACTGCACTGTGGCAACTAACAGGAGAAGCTTCCAACATCTGTCAGCAAAAGAGAGTTGGAGATGGTCTG
ATCAGGAAATACAGATTCAGAGGGAATAGTTGAAGTAAAACTTGTGGACCAAGATAAGACAATGTTCATAT
GCAAAAGCAGCTGGAGCATGAGGGAGCCACCTGGCCTGCCCCCTTGATGCTGCTGAGTGAATGGGAAATA
CATTTCTCAGATGAAAGCCTGTGATGGTATCAATGATTGTGGAGACCAAAGTGATGAACTGTGTTGTAAA
GCATGCCAAGGCAAAGGCTATTGCAAATCGGGTGTTTGCATTCCAAGCCAGTATCAATGGTGAGGTGGAC
ATTACAGGGGAAGATGAAGTTGGCTGTGCAGGCTTTGCATCTGTGGCTCAAGAAGAAAC
AGAAATTTTGACTGCTGACATGGATGCAGAAAGAAGACGGATAAAATCATTATTGGATATTTTTAAAGGT
CTCCACAGTTTATGCCATATTGGAAATTGTGTTGAT
AATTCTCAAATAAATATTTTGTTGAGGCGAAAFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
1: BC020718. Reports Homo sapiens I factor (complement), mRNA (cDNA clone MGC:22501)

**DEFINITION**  Homo sapiens I factor (complement), mRNA (cDNA clone MGC:22501)

**VERSION**  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"
  
  - /tissue_type="Liver"
  
  - /clone_lib="NIH_MGC_76"
  
  - /lab_host="DH10B"
  
  - /notes="Vector: pDNR-LIB"

- **gene**  1..1249
  
  - /gene="IF"
  
  - /note="synonym: FI"
  
  - /db_xref="GeneID:3426"
  
  - /db_xref="MIM:217030"

- **CDS**  70..1203
  
  - /gene="IF"
  
  - /codon_start=1
  
  - /product="IF protein"
  
  - /protein_id="AAH20718.1"
  
  - /db_xref="GI:18089117"
  
  - /db_xref="GeneID:3426"
  
  - /db_xref="MIM:217030"

  - /translation="MKLLHVFLLFLCFHLRFCKVTYTSQEDLVEKKCLAKKYTHLSCDKKVFCQPWQRICETCVCKLPHYQPQPGNTPGATCNNSPTFQYQQKQKELPCLGKPTKFLNNGTCTAEFKSVSLKGNSTDSEGIVEVEKLVQDQKTMFCKSSWSMREANVACLDLGFFQQGADTQRFRFSLSNISTELCHVCRGLETSLEACTFTKRTMGMQYQDFADVVCYQTQKADSPMDDFQCVNGKYISQMKAACGIDNCGDQSDELCCKACQKGFGHCKGVCIPSQYQCNGEVDCITGDEVGCAGFASVAQETEILTADMDAEERRIKSSLPPKLSGCGVKNRMHIRKRIVGGKRAQLGKMKQISLDIFKGLHRVYAILFECIIK"
misc_feature 193..393
  /gene="IF"
  /note="FIMAC; Region: factor I membrane attack complex"
  /db_xref="CDD:smart00057"
misc_feature 418..714
  /gene="IF"
  /note="SRCR; Region: Scavenger receptor cysteine-rich domain. These domains are disulphide rich extracellular domains. These domains are found in several extracellular receptors and may be involved in protein-protein interactions"
  /db_xref="CDD:pfam00530"
misc_feature 838..954
  /gene="IF"
  /note="ldl_recept_a; Region: Low-density lipoprotein receptor domain class A"
  /db_xref="CDD:pfam00057"
misc_difference 967
  /gene="IF"
  /note="'G' in cDNA is 'A' in the human genome; amino acid difference: 'A' in cDNA, 'T' in the human genome. The chimpanzee genome agrees with the cDNA sequence, suggesting that this difference is unlikely to be due to an artifact; Differences found between this sequence and the human reference genome (build 35) are described in misc_difference features below and these differences were also compared to chimpanzee genomic sequences available as of 09/15/2004 00:00:00"
misc_difference 1135
  /gene="IF"
  /note="'T' in cDNA is 'C' in the human genome; no amino acid change. The chimpanzee genome agrees with the cDNA sequence, suggesting that this difference is unlikely to be due to an artifact; Differences found between this sequence and the human reference genome (build 35) are described in misc_difference features below and these differences were also compared to chimpanzee genomic..."
1 aatattcaaa agaatacctg gagtggaaaa gatttttcag cagagacaaa gacccccgaac
61 acctccaaac tgaagcttct tcatgtttttc ctgttatattc tgtccttcca cttaaggttt
121 tgcaaggtca cttatacatc tcaagagggat ctgtggagga aaaagtgcct agcaaaaaaaaa
181 tatactccac tctccgtgca taaagtcttc tgccagccat ggcagagatg cattgagggc
241 acctgtgttt gttaaactacc gtatcagtgc ccaagaagatg gcactgtcagt gtgtgcaacct
301 aacaggagaa gcttcccaac atactgtcaa caaaaaggtt tggaatgtct tcatccagg
361 acaaaagtttt taaataacgg aacatgcaca gcggaaggaa agtttaggtgt ttcttgaag
421 catggaataa cagattcagag ggaataggtt gaagtaaacc ttgtggacca agataagaca
481 atgttcatat gcaaaagcag ctggagcatg agggaagcca acgttgccctg ccttgacctt
541 gggttttaac aaggtgctga tactcataaga aggtttaagtgt tctctgtatct ctctataaat
601 tccactgaat gtctcatatgt gcattgcggag ggattagaga ccagtttggc tgaatgtact
661 tttaataaga gaagaactat ggtttaccag gatttgcgtg atgtggttgg ttatacacag
721 aagcagatt ctccaatgga tgaacttcttt cagtggtgtga atggggaaatata cattttctcg
781 atggagacgt gtgtatgtat caatgtgtgt ggaagccaaa gtgatgaacgt gtgtgttaaa
841 gcgtgcaag gcagaagcgtt ccattgcaa tctgggttatt gcattccaa ccagtatcaa
901 tgcaatgtgtg agggtgactg cttaccagg gaagatgaag ttggctgtgc agggggggca
961 tctgtggtct aagaagaaaa agaatatttg actgctgaca tggatgcaga aagaagacgg
1021 ataaaatcat tatattcata actatctgtt ggagttaaaa acagaatgca cattcgaagg
1081 aacagagtgt tgggagggaa gcgagccaa ctggggaaaaa tgaagcaaat cttacttgaat
1141 attttttaag gtctccacag agtttatgccc atatggaat ttgtttgtat aattttcaca
1201 taatatatttt ggtgaagccaa aaaaaaaaaaaaaaaaaa aaaaaaaaaaa
<table>
<thead>
<tr>
<th>number of species</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specie1</td>
<td>RWLFSTNHKD IGTLYFIFGI WSGMVGTALS LLIRAELG-Q SGSLIGD--F</td>
</tr>
<tr>
<td>Specie2</td>
<td>RWFYSTNHKD IGTLYFIFGA WAGMVGTALS MLIRAELG-Q PGSLIGD---</td>
</tr>
<tr>
<td>Specie3</td>
<td>RWLYSTNHKD IGTLYLIFGA WAGMVGTALS LLIRAELGAQ PGSLIGD---</td>
</tr>
<tr>
<td>Specie4</td>
<td>KWSGATNHKD IGTLYLIAAGA WAGFVGAAMS VLIRLELG-Q CGSFLGNNN--</td>
</tr>
</tbody>
</table>

- gap

Phylip Format
Example of alignment
Example of deleterious single base mutation: Sickle Cell Anemia

Due to 1 swapping an A for a T, causing inserted amino acid to be valine instead of glutamine in hemoglobin
Healthy Individual

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

>gi|4504349|ref|NP_000509.1| beta globin [Homo sapiens]
MVHLTP EKSAVTALWGTKVNVDEVGGEALGRLLVVYPWTQRFFEFSDLSTPDAVMGPKVKAHGGKVVG
AFSDGLAHLDNLKGTFLATLSELHCDDKHLVDPEFRLGNVLVCVLKhFGKEFTPVPVQAAYQKVAGVAN
ALAHKYH
Diseased Individual

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

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

MVHLTPVEKSAVTALWGKVNVDEVGGGEALGRLLVVPWTQRFFESFGDLSTPDAMGNPKVAHGKKVLG
AFSDGLAHLDNLKGTFLATLSELHCDKLHVDPENFRLLGNVLVCVALHFFGKEFTPVPQAAYQKVVAVGAN
ALAHKYH
Big Alignments

Mouse and Human Genetic Similarities

Mouse chromosomes

Human chromosomes

Courtesy Lisa Stubbs
Oak Ridge National Laboratory
From foldit www.frogheart.ca -

Thermus thermophilus
large subunit ribosomal RNA
The Nussinov Folding Algorithm

Example:   **GGGAAAUCC**

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

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

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

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

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

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

From Timothy L. Bailey
Nussinov Folding Algorithm:
After scores for subsequences of length 2

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

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

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

After scores for subsequences of length 4

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

Two optimal substructures for same subsequence

From Timothy L. Bailey
Nussinov Folding Algorithm

After scores for subsequences of length 5

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

\[ \text{max}_{i < k < j} \left[ \gamma(i, k) + \gamma(k + 1, j) \right] \]
From Timothy L. Bailey

Nussinov Folding Algorithm
After scores for subsequences of length 6

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

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

After scores for subsequences of length 7

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

\[
\text{max}_{i<k<j}[\gamma(i, k) + \gamma(k+1, j)]
\]

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

After scores for subsequences of length 8

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

From Timothy L. Bailey
Nussinov Folding Algorithm

After scores for subsequences of length 9

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

From Timothy L. Bailey

$$
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 3 \\
0 & 0 & 1 & 1 & 1 & 2 & 3 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
$$
Algorithm: Nussinov RNA folding, fill stage

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

Recursion: starting with all subsequences of length 2, to length \( L \):
\[
\gamma(i, j) = \max \begin{cases} 
\gamma(i + 1, j), \\
\gamma(i, j - 1), \\
\gamma(i + 1, j - 1) + \delta(i, j), \\
\max_{i < k < j} [\gamma(i, k) + \gamma(k + 1, j)].
\end{cases}
\]
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- $G \cdot C$ and $G \circ U$ (circled)
- $G \circ C$ and $A \cdot U$ (circled)
- $C$ (circled)

Legend:

- $G$ (green)
- $C$ (cyan)
- $A$ (blue)
- $U$ (red)
- Circled connections indicate interactions or connections within the grid.

The grid and diagram represent a biological or chemical interaction pattern, possibly involving nucleotides (A, T, G, C) and their interactions. The numbers and symbols indicate the strength or type of interaction at each position.
<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>G</th>
<th>C</th>
<th>C</th>
<th>A</th>
<th>G</th>
<th>U</th>
<th>U</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
BLAST: Seeding

Query sequence: PQGEFG

Word 1: PQG
Word 2: QGE
Word 3: GEF
Word 4: EFG

BLOSUM

T > 10

PGQ (18) -> PAQ (12)
PGQ (18) -> PGA (11)
...

PGQ (18) -> ...
Human-Mouse genome homology
BLAST finds a “hit” and then extends

Seed match = hit
Original BLAST: Example

- \( w = 4 \)
- Exact keyword match of GGTC
- Extend diagonals with mismatches until score is under 50%
- Output result GTAAGGTCC
  GTTAGGTCC

From lectures by Serafim Batzoglou (Stanford)
• Original BLAST exact keyword search, THEN:
• Extend with gaps around ends of exact match until score < threshold
• Output result
  GTAAGGTCCAGT
  GTTAGGTC–AGT

From lectures by Serafim Batzoglou (Stanford)
Step 1-pairwise alignments

Compare each sequence with each other and calculate a Similarity matrix.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>.87</td>
<td>.59</td>
</tr>
<tr>
<td>B</td>
<td>.87</td>
<td>-</td>
<td>.60</td>
</tr>
<tr>
<td>C</td>
<td>.59</td>
<td>.60</td>
<td>-</td>
</tr>
</tbody>
</table>

Each number represents the number of exact matches divided by the sequence length (ignoring gaps). Thus, the higher the number the more closely related the two sequences are.

In this similarity (distance) matrix sequence A is 87% identical to sequence B.
Step 2-Create Guide Tree

Use the Similarity Matrix to create a Guide Tree to determine the “order” of the sequences.

Different sequences

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>.87</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>.59</td>
<td>-</td>
<td>.60</td>
</tr>
<tr>
<td>C</td>
<td>.60</td>
<td>.59</td>
<td>-</td>
</tr>
</tbody>
</table>

Guide Tree

Branch length proportional to estimated divergence between A and B (0.13)
Step 3-Progressive Alignment

Align A and B first. Then add sequence C to the previous alignment. In the closely aligned sequences and gaps are given a heavier weight than more divergent sequences.
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>HAHO</th>
<th>HBHO</th>
<th>MYWHP</th>
<th>P1LHB</th>
<th>LGHB</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAHU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HBHU</td>
<td>21.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAHO</td>
<td></td>
<td>32.9</td>
<td>19.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HBHO</td>
<td>20.7</td>
<td></td>
<td>39.0</td>
<td>20.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MYWHP</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>P1LHB</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>LGHB</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>
</tbody>
</table>
• Cluster the sequences by similarity to create a guide tree
• Branch length is proportional to estimated divergence between the two sequences
This article is about the computer scientist. For the Church of Ireland (Anglican) bishop, see Michael Burrows (bishop).

Michael Burrows, FRS (b. 1958) is a British computer scientist and the creator of the Burrows–Wheeler transform.

Born in Britain, he now lives in the United States, although remaining a British citizen.

Burrows did his first degree in Electronic Engineering with Computer Science at University College London and then completed his PhD in the University of Cambridge Computer Laboratory at the University of Cambridge, where he was a member of Churchill College.

Upon leaving Cambridge, he worked at the Systems Research Center (SRC) at Digital Equipment Corporation (DEC) where, with Louis Monier, he was one of the two main creators of AltaVista.¹

Following Compaq’s acquisition of DEC, Burrows worked briefly for Microsoft.² Shortly thereafter he went to Google.³

After his early work at Cambridge University, where he researched micro-kernels and basic matters of security, he went on to enlarge upon that work as systems were deployed at large scale on the Internet.

During his employment at Google, Burrows has studied concurrency and synchronisation, and for programming in the large—especially in respect the C++ language.—[citation needed]

Burrows was elected a Fellow of the Royal Society in May 2013.⁴
Bowtie 2 is an ultrafast and memory-efficient tool for aligning sequencing reads to long reference sequences. It is particularly good at aligning reads of about 50 up to 100s or 1,000s of characters, and particularly good at aligning to relatively long (e.g. mammalian) genomes. Bowtie 2 indexes the genome with an FM Index to keep its memory footprint small: for the human genome, its memory footprint is typically around 3.2 GB. Bowtie 2 supports gapped, local, and paired-end alignment modes.

**Version 2.1.0 - February 21, 2013**

- Improved multithreading support so that Bowtie 2 now uses native Windows threads when compiled on Windows and uses a faster mutex. Threading performance should improve on all platforms.
- Improved support for building 64-bit binaries for Windows x64 platforms.
- Bowtie 2 uses a lightweight mutex by default.
- Test option --no-spin is no longer available. However bowtie2 can always be recompiled with EXTRA_FLAGS="-DNO_SPINLOCK" in order to drop the default spinlock usage.

**Version 2.0.6 - January 27, 2013**

- Fixed issue whereby spurious output would be written in --no-unal mode.
- Fixed issue whereby multiple input files combined with --crossref would cause truncated output and a memory spike.
- Fixed spinlock datatype for Win64 API (LLP64 data model) which made it crash when compiled under Windows 7 x64.
- Fixed bowtie2 wrapper to handle filename/path operations in a more platform independent manner.
- Added pthread as a default library option under cygwin, and pthreadGC for MinGW.
- Fixed some minor issues that made MinGW compilation fail.

**Version 2.0.5 - January 4, 2013**
The BWT

\[ T = \text{mississippi}\# \]

\[
\begin{array}{c}
\text{mississippi}\# \\
\text{i} \text{mississippi}\#m \\
\text{i} \text{mississippi}\#mi \\
\text{i} \text{mississippi}\#mis \\
\text{i} \text{mississippi}\#miss \\
\text{i} \text{mississippi}\#missi \\
\text{i} \text{mississippi}\#missis \\
\text{i} \text{mississippi}\#mississ \\
\text{i} \text{mississippi}\#mississi \\
\text{i} \text{mississippi}\#mississip \\
\text{i} \text{mississippi}\#mississippi \\
\text{i} \text{mississippi}\#missississippi \\
\text{i} \text{mississippi}\#mississippi# \\
\end{array}
\]

Sort the rows

F

L = BWT(T)

BWT sorts the characters by their post-context
More Burrows-Wheeler

Input
SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES

Burrows-Wheeler Output
TEXYDST.E.IXIXIXXXSSMPPS..E.S.EUSFXDIIIOIIIT

Repeated characters mean that it is easier to compress

From Dr Konrad Paszkiewicz
Bowtie/Soap2 example

Reference

BWVT( Reference )

Query:
AATGATACGGCGACCACCAGATCTA

From Dr Konrad Paszkiewicz
Bowtie/Soap2 example

Reference

BWT(Reference)

Query:
AATGATACGGCGACCACCGAGATCTA

From Dr Konrad Paszkiewicz
Bowtie/Soap2 example

From Dr Konrad Paszkiewicz
Bowtie/Soap2 example

Reference

BWT( Reference )

Query:
AATGATACGGCGACCACCGAGATCTA

From Dr Konrad Paszkiewicz
Bowtie/Soap2 example

Reference

BWT( Reference )

Query:
AATGATACGGCGAC CACCGAGATCTA

From Dr Konrad Paszkiewicz
Bowtie/Soap2 example

Reference

BWT( Reference )

Query: AATGATACGCGAGACCACCGAGATCTA

From Dr Konrad Paszkiewicz
Bowtie/Soap2 example

Reference

BWT(Reference)

Query:
AATGATACGGCGACCACCCGAGATCTA

From Dr Konrad Paszkiewicz
Bowtie/Soap2 example

Reference

BWT(Reference)

Query: AATGTACGGCGACCAACCGAGATCTA

From Dr Konrad Paszkiewicz
Bowtie/Soap2 example

From Dr Konrad Paszkiewicz
A Euler path is a path that crosses every edge exactly once without repeating, if it ends at the initial vertex then it is a Euler cycle.

A Hamiltonian path passes through each vertex (note not each edge), exactly once, if it ends at the initial vertex then it is a Hamiltonian cycle (or circuit).

In a Euler path you might pass through a vertex more than once.

In a Hamiltonian path you may not pass though all edges.

An Eulerian circuit traverses every edge in a graph exactly once, but may repeat vertices, while a Hamiltonian circuit visits each vertex in a graph exactly once but may repeat edges.

de Bruijn graph is a graph representing overlaps between kmers
Example of an Eulerian graph (left); possible solution trail on the right, starting bottom left corner.
de Bruijn graph assemblers

- Oases,
- ABySS,
- SOAPdenovo,
- Velvet, MetaVelvet,
- MetaIDBA,
- Trinity
Velvet: Algorithms for de novo short read assembly using de Bruijn graphs

Daniel R. Zerbino and Ewan Birney

Abstract

We have developed a new set of algorithms, collectively called "Velvet," to manipulate de Bruijn graphs for genomic sequence assembly. A de Bruijn graph is a compact representation based on short words (k-mers) that is ideal for high coverage, very short read (25–50 bp) data sets. Applying Velvet to very short reads and paired-ends information only, one can produce contigs of significant length, up to 50-kb N50 length in simulations of prokaryotic data and 3-kb N50 on simulated mammalian BACs. When applied to real Solexa data sets without read pairs, Velvet generated contigs of ~8 kb in a prokaryote and 2 kb in a mammalian BAC, in close agreement with our simulated results without read-pair information. Velvet represents a new approach to assembly that can leverage very short reads in combination with read pairs to produce useful assemblies.
Short read de novo assembler using de Bruijn graphs [http://www.ebi.ac.uk/~zerbino/velvet/](http://www.ebi.ac.uk/~zerbino/velvet/)

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>contrib</td>
<td>Updated Manoj's contrib</td>
<td>5 months</td>
</tr>
<tr>
<td>data</td>
<td>the public release tgz of velvet contains a one line difference in th...</td>
<td>2 years</td>
</tr>
<tr>
<td>debian</td>
<td>updated debian dir</td>
<td>2 years</td>
</tr>
<tr>
<td>doc</td>
<td>Corrected manual</td>
<td>4 months</td>
</tr>
<tr>
<td>src</td>
<td>isCreateBinary() should not be inline</td>
<td>14 days</td>
</tr>
<tr>
<td>tests</td>
<td>Suggest using &quot;#/usr/bin/orv bash&quot; instead of &quot;#/bin/bash&quot; to accom...</td>
<td>a year</td>
</tr>
<tr>
<td>third-party</td>
<td>Velvet 0.7.39</td>
<td>4 years</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Add a .gitignore file</td>
<td>a year</td>
</tr>
</tbody>
</table>
def de_bruijn(k, n):
    """
    De Bruijn sequence for alphabet size k and subsequences of length n.
    """
    a = [0] * k * n
    sequence = []
    def db(t, p):
        if t > n:
            if n % p == 0:
                for j in range(1, p + 1):
                    sequence.append(a[j])
            else:
                a[t] = a[t - p]
                db(t + 1, p)
                for j in range(a[t - p] + 1, k):
                    a[t] = j
                    db(t + 1, t)
        db(1, 1)
        return sequence
    print(de_bruijn(2, 3))

which prints

[0, 0, 0, 1, 0, 1, 1, 1]
Parsimony Example

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

- four sequences, three possible unrooted trees
Maximum Parsimony Example

1  G  G  A
2  G  G  G
3  A  C  A
4  A  C  G

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

- Is a substitution
Sankoff’s Algorithm

- Check children’s every vertex and determine the minimum between them
- An example

\[
\begin{array}{c|cccc}
\delta & A & T & G & C \\
\hline
A & 0 & 3 & 4 & 9 \\
T & 3 & 0 & 2 & 4 \\
G & 4 & 2 & 0 & 4 \\
C & 9 & 4 & 4 & 0 \\
\end{array}
\]
Sankoff Algorithm: Dynamic Programming

• Calculate and keep track of a score for every possible label at each vertex
  – \( s_t(v) = \) 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:
  – \( s_t(parent) = \min_i \{ s_i(\text{left child}) + \delta_{i,t} \} + \min_j \{ s_j(\text{right child}) + \delta_{j,t} \} \)
Sankoff Algorithm (cont.)

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

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

\[
    s_A(v) = 0 + \min_j \{ s_j(w) + \delta_{j, A} \}
\]
Sankoff Algorithm (cont.)

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

\[ s_A(v) = 0 + 9 = 9 \]
Sankoff Algorithm (cont.)

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

Repeat for T, G, and C
Sankoff Algorithm (cont.)

Repeat for right subtree
Sankoff Algorithm (cont.)

Repeat for root
Sankoff Algorithm (cont.)

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 assigns each vertex with optimal character.

9 is derived from $7 + 2$

So left child is T,

And right child is T
Sankoff Algorithm (cont.)

And the tree is thus labeled...
Here are 375 phylogeny packages and 50 free servers, all that I know about. It is an attempt to be completely comprehensive. I have not made any attempt to exclude programs that do not meet some standard of quality or importance. Updates to these pages are made roughly weekly. Here is a "waiting list" of new programs waiting to have their full entries constructed. Many of the programs in these pages are available on the web, and some of the older ones are also available from ftp server machines.
Example of protein network

Example of Markov clustering
Hidden Markov Models

From CSCE555 Bioinformatics
From CSCE555 Bioinformatics

Probability of a Sequence of Events
**Definition:** A hidden Markov model (HMM)

- **Alphabet** \( \Sigma = \{ b_1, b_2, \ldots, b_M \} \)
- **Set of states** \( Q = \{ 1, \ldots, K \} \)
- **Transition probabilities** between any two states

\[
a_{ij} = \text{transition prob from state } i \text{ to state } j
\]

\[
a_{i1} + \ldots + a_{iK} = 1, \text{ for all states } i = 1\ldots K
\]

- **Start probabilities** \( a_{0i} \)

\[
a_{01} + \ldots + a_{0K} = 1
\]

- **Emission probabilities** within each state

\[
e_{i}(b) = P\left( x_i = b \mid \pi_i = k \right)
\]

\[
e_{i}(b_1) + \ldots + e_{i}(b_M) = 1, \text{ for all states } i = 1\ldots K
\]
Example: The Dishonest Casino

A casino has two dice:

- **Fair die**
  \[ P(1) = P(2) = P(3) = P(5) = P(6) = \frac{1}{6} \]

- **Loaded die**
  \[ P(1) = P(2) = P(3) = P(5) = \frac{1}{10} \]
  \[ P(6) = \frac{1}{2} \]

Casino player switches back-&-forth between fair and loaded die once every 20 turns

**Game:**
1. You bet $1
2. You roll (always with a fair die)
3. Casino player rolls (maybe with fair die, maybe with loaded die)
4. Highest number wins $2
Question # 1 – Evaluation

GIVEN

A sequence of rolls by the casino player

124552646214614613613666166466163661636616361651561511514612
3562344

QUESTION

How likely is this sequence, given our model of how the casino works?

This is the EVALUATION problem in HMMs
Question # 2 – Decoding

GIVEN

A sequence of rolls by the casino player

124552646214614613613666166466163661636616361651561511514612356234

4

QUESTION

What portion of the sequence was generated with the fair die, and what portion with the loaded die?

This is the **DECODING** question in HMMs
Question # 3 – Learning

GIVEN

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

QUESTION

How “loaded” is the loaded die? How “fair” is the fair die? How often does the casino player change from fair to loaded, and back?

This is the LEARNING question in HMMs
The three main questions on HMMs

1. Evaluation
   GIVEN a HMM M, and a sequence x,
   FIND Prob[ x | M ]

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

3. Learning
   GIVEN a HMM M, with unspecified transition/emission probs.,
   and a sequence x,
   FIND parameters \( \theta = (e_i(.), a_{ij}) \) that maximize \( P[ x | \theta ] \)
Example: the dishonest casino

Let the sequence of rolls be:

\[ x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4 \]

Then, what is the likelihood of

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

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

\[
\frac{1}{2} \times P(1 \mid \text{Fair}) \times P(\text{Fair} \mid \text{Fair}) \times P(2 \mid \text{Fair}) \times P(\text{Fair} \mid \text{Fair}) \times \ldots \times P(4 \mid \text{Fair}) = \\
\frac{1}{2} \times (1/6)^{10} \times (0.95)^9 = 0.00000000521158647211 = 0.5 \times 10^{-9}
\]
Example: the dishonest casino
So, the likelihood the die is fair in all this run
is just $0.521 \times 10^{-9}$

OK, but what is the likelihood of
= Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?

$\frac{1}{2} \times P(1 \mid \text{Loaded}) \times P(\text{Loaded, Loaded}) \times \ldots \times P(4 \mid \text{Loaded}) = $

$\frac{1}{2} \times (1/10)^8 \times (1/2)^2 \times (0.95)^9 = 0.00000000078781176215 = 7.9 \times 10^{-10}$

Therefore, it is after all 6.59 times more likely that the die is fair all the way, than that it is loaded all the way.
Example: the dishonest casino

Let the sequence of rolls be:

\[ x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6 \]

Now, what is the likelihood \( \pi = F, F, \ldots, F \)?

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

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

\[
\frac{1}{2} \times (1/10)^4 \times (1/2)^6 \times (0.95)^9 = .000000049238235134735 = 0.5 \times 10^{-7}
\]

So, it is 100 times more likely the die is loaded
Example

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

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

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

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

6 nucleotides “123456” parsed as F, contribute $0.95^6 \times (1/6)^6$

$= 1.6 \times 10^{-5}$

parsed as L, contribute $0.95^6 \times (1/2)^1 \times (1/10)^5 = 0.4 \times 10^{-5}$

“162636” parsed as F, contribute $0.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$

parsed as L, contribute $0.95^6 \times (1/2)^3 \times (1/10)^3 = 9.0 \times 10^{-5}$