Computational Biology

Lecture 3
Sequencing

• As before, DNA is cut into small (\( \approx 0.4\text{KB} \)) fragments and a clone library is formed.

• Biological experiments allow to read a certain number of these short fragments per experiment.

• Entire genome (\( \approx 4\text{GB long} \)) must be assembled from the knowledge of these short fragments.
Shortest Superstring

• The simplest naive approximation of DNA sequencing, ignoring unavoidable experimental errors, is the following:

• Shortest Superstring Problem: Given a set of strings $s_1, \ldots, s_n$, find the shortest string $s$ such that each $s_i$ appears as a substring of $s$.

• This problem is NP-hard.
Shortest Superstring

• Set of strings:
  \{000,001,010,011,100,101,011,111\}

• A trivial superstring:
  000 001 010 011 100 101 011 111

• A shortest superstring:
  0 0 0 1 1 1 0 1 0 0
Sequencing by Hybridization

• Although DNA sequencing is a fast and efficient procedure now, it was time consuming and hard 10 years ago.

• In 1988, 4 groups of biologist independently and simultaneously suggested a new approach called Sequencing by Hybridization (SBH).

• Build a DNA chip containing thousands of short DNA fragments (probes) working like the chip’s memory.

• Each probe will reveal some information about an unknown DNA fragment.

• All the pieces of information combined would solve the DNA sequencing.

• Of course, in 1988, no one believed that such a thing could work! Now, building DNA arrays with thousands of probes has become an industry.
SBH

- Given a DNA fragment with an unknown sequence, a DNA array would provide its ℓ-tuple composition, i.e. information about all substrings of length ℓ contained in this fragment.

- **SBH Problem**: Reconstruct a string by its ℓ-tuple decomposition.

- Although conventional DNA sequencing and SBH are different approaches, computationally they are similar. SBH is a special instance of the Shortest Superstring problem when \(s_1 \ldots s_n\) represent all substrings of a fixed length.

- While the general Shortest Superstring problem in NP-hard, SBH can be solved efficiently.
Finding CG-islands

- The most infrequent dinucleotide in many genomes is CG (CG has tendency to mutate to TG).

- However, CG appears relatively frequently around genes in areas called CG-islands.

- How to define and find CG-islands in a genome?

- This is similar to the following analogy of the Casino: the dealer uses two coins: biased and unbiased. He switches coins with probability $p$. Given a sequence of coin tosses, can you find out when the biased coin was used?

- Why is that a good analogy? Because as we go along the genome, we can switch between two states: CG-island and non CG-island. Each state has different probability for the occurrence of CG. Given the genome, can you tell when you are in a CG island?
Similarity Search

• After sequencing, biologist have no idea about the function of the newly sequenced gene.

• Hoping to find a clue, they compare it with previously sequenced genes with known functionality.

• **Edit distance**: number of operations needed to transform one string into another, where operations are insertion of a symbol, deletion of a symbol, and substitution of a symbol.

• Since mutations in DNA can be represented by the above operations, the edit distance is a natural measure of similarity between DNA fragments.

• Variations to the basic edit distance above are possible and lead to alignment algorithms.
Sequence Alignment
Similar Sequences

• These two look very much alike
  GACGGGATTAG
  GATCGGAATAG

• Aligning them one above the other
  GA−CGGATTAG
  GATCGGAATAG
Alignment

- **Alignment**: Insertion of gaps in arbitrary locations along the sequences so that they end up with the same size.

- No gap in one sequence should be aligned to a gap in the other.

- We want the *best* alignment, but what is best?
Simple Scoring

• Two identical characters receive a score of $+ m$ (match)

• Two different characters receive a score of $- s$ (mismatch)

• A character and a gap receive a score of $- d$ (gap)

• $\text{score} = (\#\text{matches}).m - (\#\text{mismatches}).s - (\#\text{gaps}).d$
Example

- $m = 1$
- $s = 1$
- $d = 2$

$$\text{GA} - \text{CGGATTTAG}$$
$$\text{GA} \text{T} \text{CGGAAATAG}$$

score $= +1(9) -1(1) -2(1) = 6$

Why do we penalize gaps more?
Insertions and Deletions are less likely than substitutions
More General Scoring

The scoring scheme can be more general

- Given two sequences x and y, aligning $x_i$ and $y_j$ could add a score of $s(x_i,y_j)$. Therefore, we have a scoring matrix.

- [later] Gap penalty is not linear, once you have a gap, it is likely to have another one so we should penalize the start of the gap more
Greedy Algorithm

• To obtain the best alignment, try all possible alignments and find the best one
  – Exponentially many alignments!
  – How many? (homework)

• Greedy would result in a very slow algorithm
Dynamic Programming

• Solving an instance of the problem by taking advantage of already computed solution for smaller instances of the problem.

• To find optimal alignment for sequences $x$ and $y$, compute optimal alignments for prefixes of $x$ and $y$. 
Alignment is Additive

• The score of aligning

\[ x_1 \ldots x_m \]
\[ y_1 \ldots y_n \]

is additive (with our particular scoring scheme)

• If the alignment is

\[ x_1 \ldots x_i \quad x_{i+1} \ldots x_m \]
\[ y_1 \ldots y_j \quad y_{j+1} \ldots y_n \]

then the score is:

\[ \text{score}(x_1 \ldots x_i, y_1 \ldots y_j) + \text{score}(x_{i+1} \ldots x_m, y_{j+1} \ldots y_n) \]
Optimal structure

Optimal solution => optimal solution of sub-problems

optimal

\[
\begin{array}{ccc}
\text{GA-CG} & | & \text{GATTAG} \\
\text{GATCG} & | & \text{GAATAG} \\
\end{array}
\]

cut optimal alignment anywhere
Dynamic programming

• Assume we want to align
  \[ x_1 \ldots x_m \]
  \[ y_1 \ldots y_n \]

• Let \( A(i,j) \) be the score of optimally aligning the two prefixes
  \[ x_1 \ldots x_i \]
  \[ y_1 \ldots y_j \]
Dynamic Programming (cont.)

Three possible cases for aligning $x_1..x_i$ and $y_1..y_j$

1. \[
\begin{array}{c}
  \text{X}_1..\text{X}_{i-1} \\
  \text{Y}_1..\text{Y}_{j-1}
\end{array}
\quad \begin{array}{c}
  x_i \\
  y_j
\end{array}
\]

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

2. \[
\begin{array}{c}
  \text{X}_1..\text{X}_{i-1} \\
  \text{Y}_1..\text{Y}_j
\end{array}
\quad \begin{array}{c}
  x_i \\
  -
\end{array}
\]

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

3. \[
\begin{array}{c}
  \text{X}_1..\text{X}_i \\
  \text{Y}_1..\text{Y}_{j-1}
\end{array}
\quad \begin{array}{c}
  - \\
  y_j
\end{array}
\]

\[A(i,j) = A(i, j-1) - d\]
Dynamic Programming (cont.)

**Inductive step:**

\[ A(i, j - 1), A(i - 1, j), A(i - 1, j - 1) \] are correct

Then,

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

Where

\[ s(x_i, y_j) = \begin{cases} 
m, & \text{if } x_i = y_j; \\
- s, & \text{if not}
\end{cases} \]
Illustration

\[ A(i-1,j-1) \quad A(i-1,j) \]
\[ A(i,j-1) \quad A(i,j) \]

\( A(m,n) \) will be the optimal score
What Else?

• Base case

- \( A(0,0) = 0 \)
- \( A(i,0) = -d.i \) \( i = 1 \ldots m \)
- \( A(0,j) = -d.j \) \( j = 1 \ldots n \)
## AAAC and AGC

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>A</td>
<td>-4</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>-6</td>
<td>-3</td>
<td>-2</td>
</tr>
<tr>
<td>C</td>
<td>-8</td>
<td>-5</td>
<td>-4</td>
</tr>
</tbody>
</table>
Obtaining Actual Alignment

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>A</td>
<td>-4</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>-6</td>
<td>-3</td>
<td>-2</td>
</tr>
<tr>
<td>C</td>
<td>-8</td>
<td>-5</td>
<td>-4</td>
</tr>
</tbody>
</table>

Saad Mneimneh
http://engr.smu.edu/~saad/
### Obtaining Actual Alignment

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>A</td>
<td>-4</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>-6</td>
<td>-3</td>
<td>-2</td>
</tr>
<tr>
<td>C</td>
<td>-8</td>
<td>-5</td>
<td>-4</td>
</tr>
</tbody>
</table>

**AAAC**

**AG–C**

[Image of DNA structure]

[Website link: http://engr.smu.edu/~saad/]

Saad Mneimneh
Needleman-Wunsch Algorithm

1. Initialization
   \[ A(0, 0) = 0 \]
   \[ A(i, 0) = -i.d \text{ for } i = 1 \ldots m \]
   \[ A(0, j) = -j.d \text{ for } j = 1 \ldots n \]

2. Main Iteration (Aligning prefixes)
   for each \( i = 1 \ldots m \)
   for each \( j = 1 \ldots n \)
   \[ A(i, j) = \max \begin{cases} 
   A(i-1, j-1) + s(x_i, y_j) & \text{[case 1]} \\
   A(i-1, j) - d & \text{[case 2]} \\
   A(i, j-1) - d & \text{[case 3]} 
   \end{cases} \]

   \[ Ptr(i, j) = \begin{cases} 
   \text{Diag} & \text{[case 1]} \\
   \text{Up} & \text{[case 2]} \\
   \text{Left} & \text{[case 3]} 
   \end{cases} \]

3. Termination
   \( A(m, n) \) is the optimal score, and
   from \( Ptr(m, n) \) can trace back optimal alignment.
Complexity

• Time
  – $O(mn)$

• Space
  – $O(mn)$