Sequence Alignment
Overview

• DNA and mutations
• Edit distance and some properties
• Alignment v.s. edit distance
Sequence is cheap, everything else is expensive

It’s pretty cheap to sequence DNA at this point; hundreds of genomes have been sequenced, and another thousand or so are in the process.

Sequence is much cheaper – In 2008, a few dollars per reaction (a few hundred bps)

But studying molecules’ shapes involves expensive equipment: microscopes, NMR spectrometers, X-ray crystallography, ...
DNA
For this course

- DNA is double-helical, with two complementary strands.
- Complementary bases:
  - Adenine (A) – Thymine (T)
  - Cytosine (C) – Guanine (G)
- What is the reverse complement of AAGGTAGC?
- Two types:
  - Purine: A and G
  - Pyrimidine: T and C
- Two substitution types:
  - Transition: A ↔ G and T ↔ C
  - Transversion: otherwise.
  - Transition occurs more frequently. 2/3 of SNPs are transitions.
Single Nucleotide Polymorphisms

• Single base variation between members of a species.
• SNPs, which make up about 90% of all human genetic variation, occur every 100 to 300 bases along the 3–billion–base human genome.
• Major risk for genetic disease.
Compare DNA sequences

• The most often used distance on strings in computer science is Hamming distance.
  • AGTTTAATCA
  • ||| ||| |
  • AGTATAACGA

• This makes some sense on comparing DNA sequences in some cases. But there are other mutations
  • Substitution AC\textcolor{red}{AGT} \rightarrow AC\textcolor{red}{GGT}
  • Insertion/deletion (indel) AC\textcolor{red}{AGT} \rightarrow ACGT
  • Inversion ACA……GT \rightarrow AG……ACT
  • Translocation AC……AG…TAA \rightarrow AG…TC……AAA
  • Duplication

• We only consider the first two mutations for now.
  • There are algorithms for the other mutations…
Edit Distance

• How to compare two current sequences without knowing the mutation history?
  • E.g. CGATA and GGCCCATTA
• How “far” away they are from each other?
• The most commonly used distance is edit distance: the minimum number of edit operations it takes to convert one to another.
  • Edit operations often contain substitutions and indels
  • But there are extensions to this basic model.
• $d(\text{ATGCATTTA, ATGTACTTTTC})$
  • ATGCATTTTA
  • ATGTACTTTTC
Edit distance v.s. mutation history

• Edit distance simpler than reconstructing the history
  • Orders of the edit operations do not matter.
  • If two events overlap or even cancel each other in the evolution, they cannot be seen at edit distance.

• It is a distance metric.
  • Identity: $d(x,y) = 0$ iff $x = y$
  • Symmetry: $d(x,y) = d(y,x)$
  • Triangular Inequality: $d(x,z) \leq d(x,y) + d(y,z)$

• Write clear proof
  • when you’re searching for a proof, it’s much intuition driven. But when the proof is to be written, intuition is not enough.
Alignment

• Hard to visually show the edit distance:
  • E.g. C→T@4, insert C@6, delete@9

• Alignment is much nicer:
  • ATGCA−TTTA
    ||| | || |
    ATGTACTT−A

• Align the two sequences by inserting spaces, so that they are most similar column−wisely.

• What does “similar” mean?
• Usually we need a “scoring function” or a “score function”.
• E.g. Match = 1, mismatch = 0, indel = 0.
Alignment v.s. Edit Distance

• By a properly defined score scheme, finding the optimal alignment equivalent to edit distance computing
  • match =
  • mismatch =
  • indel =

• Prove it!
“Optimal” alignment

• The word “optimal” alignment is somewhat misleading. Ideally we want to find the “real” alignment of the sequences according to the real evolution instead.

• But we cannot!

• So we are forced to find the so-called “optimal” alignment.

• This applies to most, if not all, bioinformatics problems. The “optimal” solution is not necessary the correct solution. It all depends on how good the score function is.

• The identity scoring scheme is not a very accurate one.
  • E.g., transitions and transversions have the same score.
  • Along this alignment topic, we will refine the score functions.
Scoring sequence alignment

How to score an alignment?

Simplest scoring scheme:
• +1 = match
• −1 = mismatch
• −1 = indel

This is called “linear gap penalty” because the cost of a gap (consecutive indels) is proportional to its length. (We could have each gap position cost $g$, for some negative constant $g$.)

Let’s see some examples
Two sample alignments

AATGCGA−TTTT
  ||  ||  ||  ||
G−TG--ACTTTTC

6 matches: +6
2 mismatches: −2
4 indels: −4 Total score: 0

Our measure makes the first one win.
Alignment with DP

• The question is how alignment can be computed with a computer?

• Dynamic Programming
  • Requires the subsolution of an optimal solution is also optimal.

• AATGC-GA−TTTT
  |   ||    |   |||
  A−TG--ACTTTT

• AATGC-GA−TTTT
  |   ||    |   |||
  A−TG--ACTTTT
Suppose we are to align $S[1..i]$ and $T[1..j]$. Consider the last column of the alignment. Three cases arise:

- **Case 1:** $S[1..i-1] \ S[i] \ S[1..i-1] \ S[i] \ S[1..i] \ -$  
  $T[1..j-1] \ T[j] \ T[1..j] \ -$  
  $T[1..j-1] \ T[j]$

- **Case 2:** $S[1..i-1] \ S[i] \ S[1..i-1] \ S[i] \ S[1..i] \ -$  
  $T[1..j] \ -$  
  $T[1..j]$  

- **Case 3:** In either case, the sub-alignment without the last column is an optimal one.
Recursive definition

• Denote the optimal alignment score of $S[1..i]$, $T[1..j]$ by $DP[i,j]$. Then $DP[m,n]$ is the optimal alignment score.

• Let $f(*,*)$ be the score between two letters.

• Case 1: $s[i]$ v.s. $t[j]$  
  • $DP[i,j] = DP[i-1, j-1] + f(s[i], t[j])$;

• Case 2: $s[i]$ v.s. $-$  
  • $DP[i,j] = DP[i-1, j] + \text{indel}$;

• Case 3: $t[j]$ v.s. $-$  
  • $DP[i,j] = DP[i, j-1] + \text{indel}$;

• Therefore…  
  $$DP[i,j] = \max \begin{cases} 
DP[i-1, j-1] + f(s[i], t[j]); \\
DP[i-1, j] + \text{indel}; \\
DP[i, j-1] + \text{indel}; 
\end{cases}$$
Algorithm

DP[0,0] = 0;
for i from 1 to m
    DP[i,0] = i * indel;
for j from 1 to n
    DP[0,j] = j * indel;
for i from 1 to m
    for j from 1 to n
        DP[i,j] = max
            DP[i-1, j-1] + f(s[i], t[j]);
            DP[i-1, j] + indel;
            DP[i, j-1] + indel;

Output DP[m,n];
## Figure

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

CATTTG--
--ATTGAA
Question: What if write a recursive function?

DP(int i, int j) {
  x = DP(i-1, j-1) + f(s[i], t[j]);
  y = DP(i-1, j) + indel;
  z = DP(i, j-1) + indel;
  return max(x, y, z);
}

What made the difference?
Getting the actual alignment – backtracking

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
</table>
| A   | 0 | -1| -2| -3| -4| -5| **No need to physically record the green arrows. Why?**
| T   | -1| -1| 0 | -1| -2| -3|
| T   | -2| -2| -1| 1 | 0 | -1|
| T   | -3| -3| -2| 0 | 2 | 1 |
| G   | -4| -4| -3| -1| 1 | 3 |
| A   | -5| -5| -2| -2| 0 | 2 |

CATTTG--

--ATTGA
Pseudo Code for Backtracking

While i > 0 or j > 0:
• Figure out which of the three terms gave rise to $M_{i,j}$.
• Move to the right place (reduce i, reduce j, or reduce both), and write down the configuration of the current column.

Floating point rounding error!

S=T=empty string;
While i > 0 or j > 0
  if $M_{i,j} = M_{i-1,j} + \text{indel}$ then
    $S = s[i] + S$;
    $T = '-' + T$;
    i--;
  else if $M_{i,j} = M_{i,j-1} + \text{indel}$ then
    $S = '-' + S$;
    $T = t[j] + T$;
    j--;
  else
    $S = s[i] + S$;
    $T = t[j] + T$;
    i--; j--;
Solving Rounding Error

While \( i > 0 \) or \( j > 0 \):

- Figure out which of the three terms gave rise to \( M_{i,j} \) by picking the largest.
  - \( M_{i-1,j} + \text{indel} \)
  - \( M_{i,j-1} + \text{indel} \)
  - \( M_{i-1,j-1} + f(s[i], t[j]) \)

- Move to the right place (reduce \( i \), reduce \( j \), or reduce both), and write down the configuration of the current column.

Array out of boundary error!

S = T = empty string;

While \( i > 0 \) or \( j > 0 \)

\[
x_1 = M_{i-1,j} + \text{indel};
\]
\[
x_2 = M_{i,j-1} + \text{indel};
\]
\[
x_3 = M_{i-1,j-1} + f(s[i], t[j]);
\]

max = \( \max(x_1, x_2, x_3) \);

if \( x_1 == \text{max} \) then

\[
S = s[i] + S;
\]
\[
T = '-' + T;
\]
\[
i--;
\]

else if \( x_2 == \text{max} \) then

\[
S = '-' + S;
\]
\[
T = t[j] + T;
\]
\[
j--;
\]

else

\[
S = s[i] + S;
\]
\[
T = t[j] + T;
\]
\[
i--; j--;
\]
Time Complexity

The dynamic programming algorithm runs in O(nm) time: Each step requires only 3 checks to other points in the matrix.

How about the backtracking?
Space Complexity

We also need $O(nm)$ space, to store the matrix.

If we only want to know the score of the optimal alignment, we can do that in linear space.

How?

Can you reconstruct the alignment in linear space?
Alignments are scored

• Need to **score** alignments.
• Note, right away: we might be **wrong**.
• That is, the alignment that has highest **score** may not be the one that actually matches evolutionary history.
• So you should never trust that an alignment must be right. It just optimizes the score.
• When we move to multiple alignments, things get worse: no guarantee of the optimal score, even.
Do scores have meaning?

Every evolutionary history gives rise to an alignment.

[Not 1–1: number of possible histories much larger than the number of meaningful alignments of two sequences!]

Example:

True alignment:

\[
\text{AGATTCC} \quad \text{G–A–--TC}
\]
Improving the score

- Some simple improvements that do not affect the algorithm.
  - Give indel, mismatch, match different weights according to how rarely they occur during evolution.
  - Give different weights to transitions and transversions.
  - In genetics, a **transition** is a mutation changing a purine to another purine nucleotide (A <-> G) or a pyrimidine to another pyrimidine nucleotide (C <-> T). Approximately two out of every three SNPs are transitions.

- Some other improvements affect the algorithm. E.g. gap penalty.
Why are gaps scores linear?

- Consecutive gaps no better than separated ones.
- These histories get the same score:

- because they give these two alignments:
  - CCTTGT  CCTTGT
  - CCT--T  CC--T--T
- both of which score +2.
That seems somehow wrong.

Evolutionary history on the left seems “simpler” than the other.

→ Indels are relatively rare. Indel a segment of \( k \) consecutive bases is much easier than \( k \) scattered indels.

→ But linear gap scores give us no way of choosing between these two options.

Can we re-design our scoring framework to make higher scoring alignments represent simpler or more likely histories?
Arbitrary gap penalty

- Consecutive insertions or deletions are called a gap. Suppose the gap penalty of a length $k$ gap is $g(k)$ instead of the simple $ck$.
- Assume $g(x)+g(y) \leq g(x+y)$. (not necessary, but...)
- How to compute optimal alignment under this definition of score function?

AATGCGA-TTTT
|   |   |   |   |   |
A-TG--ACTTTT

score = $8 + 2g(1) + g(2)$

AATGCGA-TTTT
|   |   |   |   |   |
A-TG--ACTTTT

subsolution still optimal

AATGCGA-TTTT
|   |   |   |   |   |
A-TG--ACTTTT

subsolution not necessarily optimal

Let’s try construct a counterexample.
Arbitrary Gap Penalty

• We use $D[i,j]$ to denote the optimal alignment score of $s[1..i]$ and $t[1..j]$.

• $D[i,j] = \max$ of the following three cases:
  • $D[i-1,j-1]+f(s[i],t[j])$. (s[i] v.s. t[j])
  • $\max_{1 \leq k \leq i} D[i-k,j]+g(k)$
  • $\max_{1 \leq k \leq j} D[i,j-k]+g(k)$

• Attention: when we do dynamic programming, the matrix itself has a physical meaning. Its definition is this physical meaning. The recurrence relation is the algorithm to compute it, but is not the definition.
Time Complexity

- Cubic time complexity.
- In bioinformatics, the hardest work is perhaps choosing the right scoring function so that it both
  - approximates the real biology
  - can easily be computed
- Now let’s simplify the $g(k)$ a little. We basically want a function that grows slower than linear.
- $g(k) = a + b \cdot k$
  - $a = \text{gap open penalty}$
  - $b = \text{gap extension penalty}$
Affine Gap Penalty

With **affine gap costs**, the score of an alignment equals:

+1 for every match

−1 for every mismatch

−5 for every gap **open**

−1 for every gap **extension**.

(Of course, all of these can be arbitrary values, as we’ll see next week.)

How to actually find the optimal alignment?
Affine gap penalty example

- ATAGG--AAG
- || || || |
- ATTGGCAATG
- 6 match, 2 mismatch, 1 gap open, 2 gap extension, score = ?
- ATAGG-AA-G
- || || || |
- ATTGGCAATG
Why old algorithm fails?

- ATAGG--AAG
- || ||  |
- ATTGGCAATG

- Sub-solution of optimal is not optimal
- ATAGG--AA--G
- || || || |
- ATTGGCAATG

- Two optimal sub-solutions do not give the optimal.

- Because of the score of the “global” solution may not equal to the simple sum of the two sub solutions.
How to solve this problem?

• Consider the last column of an alignment again:

\[
\begin{array}{ccc}
\text{AT–GG–} & \text{ATGG--} & \text{ATAGGC–} \\
| | | | & | | | & | | | \\
\text{ATTGGGC} & \text{ATTGGGC} & \text{ATTGGG–A}
\end{array}
\]

• When the last column is an indel, the added cost depends on the previous column.
  • If previous column has a gap opened already, then
    • \( D[i,j] = D[i,j-1] + \text{gapext} \)
  • Else
    • \( D[i,j] = D[i,j-1] + \text{gapopen} + \text{gapext} \)

• How do we know the previous column’s configuration?

• Because by induction we know the optimal solution for \( D[i,j-1] \), can we simple look at it and use the configuration?
How to solve the problem?

• The sub-solution of the optimal solution may not be optimal. So cannot do that.
  – Exercise: Find a counter example to show that the above algorithm fail.

  \[
  \begin{align*}
  \text{ATAGG} & \quad \text{ATAGG}^{-} & \quad \text{ATAGGC} \\
  | & | & | & | & | & | \\
  \text{ATTGG} & \quad \text{ATTGGGC} & \quad \text{ATTGGC}^{-}
  \end{align*}
  \]

• Instead, we compute the optimal solution by forcing the last column to be one of the three configurations.
Recursive definition

Let $D_0[i,j]$ be the alignment score providing that $s[i]$ matches $t[j]$.
Let $D_1[i,j]$ be the alignment score providing that $s[i]$ matches $\cdot$.
Let $D_2[i,j]$ be the alignment score providing that $t[j]$ matches $\cdot$.

\[
\begin{align*}
\text{ATAGG} & \quad \text{ATAGGC} & \quad \text{ATAGG-} \\
\mid & \quad \mid & \quad \mid \\
\text{ATTGG} & \quad \text{ATTGG-} & \quad \text{ATTGGC}
\end{align*}
\]
Recurrence Relation

\[ D_0[i,j] = f(s[i], t[j]) + \max \begin{cases} D_0[i-1, j-1]; \\ D_1[i-1, j-1]; \\ D_2[i-1, j-1]; \end{cases} \]

\[ D_1[i,j] = b + \max \begin{cases} D_0[i-1, j] + a; \\ D_1[i-1, j]; \\ D_2[i-1, j] + a; \end{cases} \]

\[ D_2[i,j] = b + \max \begin{cases} D_0[i, j-1] + a; \\ D_1[i, j-1] + a; \\ D_2[i, j-1]; \end{cases} \]
Algorithm

• No difference to the simple DP but now uses three arrays.
• Backtracking should be very careful!
• Still $O(nm)$ time. Approximately 3 times slower.
• This is okay because the model is more expressive.
• Much faster than the general gap penalty.
Two sequences always arise from a common ancestor.
• Since that ancestor lived, there have been a long number of descendants, leading up to the present time.
• A full evolutionary history would detail the mutations that happened over the course of history.
• We don’t have a time machine. The next best thing: alignments.
Characterize which positions in the two sequences arose from the common ancestor. Between these, “indel” mutations.
Review

DP algorithm for alignment

- Matrix entry: score of best alignment of $S(1...i)$ to $T(1...j)$.
- Can compute matrix entries in constant time $\Rightarrow O(nm)$ runtime.
- Can backtrack through matrix to find optimal alignment.
- If only score is needed, then linear space.

Scoring function important

- Some do not change DP (better scoring scheme)
- Some change (gap penalty)
- General gap penalty cubic time.
- Affined gap penalty still quadratic time.