Gap Penalties

CMSC 423
General Gap Penalties

Now, the cost of a run of $k$ gaps is $\text{gap} \times k$.

It might be more realistic to support general gap penalty, so that the score of a run of $k$ gaps is $\text{gap}(k) < \text{gap} \times k$.

Then, the optimization will prefer to group gaps together.

These have the same score, but the second one is often more plausible.

A single insertion of “GAAT” into the first string could change it into the second.
General Gap Penalties

Previous DP no longer works with general gap penalties because the score of the last character depends on details of the previous alignment:

\[
\begin{align*}
\text{AAAGAATTCA} & \quad \text{vs.} \quad \text{AAAGAATTCA} \\
\text{A-A-A-T-CA} & \quad \text{vs.} \quad \text{AAA----TCA}
\end{align*}
\]

Instead, we need to “know” how long a final run of gaps is in order to give a score to the last subproblem.
Three Matrices

We now keep 3 different matrices:

\[ M[i,j] = \text{score of best alignment of } x[1..i] \text{ and } y[1..j] \text{ ending with a character-match or mismatch.} \]

\[ X[i,j] = \text{score of best alignment of } x[1..i] \text{ and } y[1..j] \text{ ending with a space in } X. \]

\[ Y[i,j] = \text{score of best alignment of } x[1..i] \text{ and } y[1..j] \text{ ending with a space in } Y. \]

\[
M[i,j] = \text{match}(i,j) + \max \left\{ M[i-1,j-1], X[i-1,j-1], Y[i-1,j-1] \right\}
\]

\[
X[i,j] = \max \left\{ M[i,j-k] - \text{gap}(k) \text{ for } 1 \leq k \leq j, Y[i,j-k] - \text{gap}(k) \text{ for } 1 \leq k \leq j \right\}
\]

\[
Y[i,j] = \max \left\{ M[i-k,j] - \text{gap}(k) \text{ for } 1 \leq k \leq i, X[i-k,j] - \text{gap}(k) \text{ for } 1 \leq k \leq i \right\}
\]
The M Matrix

We now keep 3 different matrices:

- \( M[i,j] = \text{score of best alignment of } x[1..i] \text{ and } y[1..j] \text{ ending with a character-match or mismatch.} \)
- \( X[i,j] = \text{score of best alignment of } x[1..i] \text{ and } y[1..j] \text{ ending with a space in } X. \)
- \( Y[i,j] = \text{score of best alignment of } x[1..i] \text{ and } y[1..j] \text{ ending with a space in } Y. \)

By definition, alignment ends in a match.

\[
M[i,j] = \text{match}(i,j) + \max \left\{ M[i-1,j-1], X[i-1,j-1], Y[i-1,j-1] \right\}
\]

Any kind of alignment is allowed before the match.
The X (and Y) matrices

\[ X[i, j] = \max \begin{cases} M[i, j - k] - \text{gap}(k) & \text{for } 1 \leq k \leq j \\ Y[i, j - k] - \text{gap}(k) & \text{for } 1 \leq k \leq j \end{cases} \]

\( k \) decides how long to make the gap.

We have to make the whole gap at once in order to know how to score it.
The $X$ (and $Y$) matrices

$$X[i, j] = \max \begin{cases} M[i, j - k] - \text{gap}(k) & \text{for } 1 \leq k \leq j \\ Y[i, j - k] - \text{gap}(k) & \text{for } 1 \leq k \leq j \end{cases}$$

$k$ decides how long to make the gap.

We have to make the whole gap at once in order to know how to score it.

This case is automatically handled.
Running Time for Gap Penalties

\[ M[i, j] = \text{match}(i, j) + \max \begin{cases} M[i - 1, j - 1] \\ X[i - 1, j - 1] \\ Y[i - 1, j - 1] \end{cases} \]

\[ X[i, j] = \max \begin{cases} M[i, j - k] - \text{gap}(k) & \text{for } 1 \leq k \leq j \\ Y[i, j - k] - \text{gap}(k) & \text{for } 1 \leq k \leq j \end{cases} \]

\[ Y[i, j] = \max \begin{cases} M[i - k, j] - \text{gap}(k) & \text{for } 1 \leq k \leq i \\ X[i - k, j] - \text{gap}(k) & \text{for } 1 \leq k \leq i \end{cases} \]

Final score is \( \max \{M[n, m], X[n, m], Y[n, m]\} \).

How do you do the traceback?

Runtime:

- Assume \(|X| = |Y| = n\) for simplicity: 3n^2 subproblems
- 2n^2 subproblems take \(O(n)\) time to solve (because we have to try all \(k\))
  \(\Rightarrow O(n^3)\) total time
Affine Gap Penalties

- $O(n^3)$ for general gap penalties is usually too slow...

- We can still encourage spaces to group together using a special case of general penalties called *affine gap penalties*:
  
  \[ \text{gap\_start} = \text{the cost of starting a gap} \]
  
  \[ \text{gap\_extend} = \text{the cost of extending a gap by one more space} \]

- Same idea of using 3 matrices, but now we don’t need to search over all gap lengths, we just have to know whether we are starting a new gap or not.
Affine Gap Penalties

\[ M[i, j] = \text{match}(i, j) + \max \left\{ M[i-1, j-1], X[i-1, j-1], Y[i-1, j-1] \right\} \]

\[ X[i, j] = \max \left\{ \text{gap_start} + \text{gap_extend} + M[i, j-1], \text{gap_extend} + X[i, j-1], \text{gap_start} + \text{gap_extend} + Y[i, j-1] \right\} \]

\[ Y[i, j] = \max \left\{ \text{gap_start} + \text{gap_extend} + M[i-1, j], \text{gap_start} + \text{gap_extend} + X[i-1, j], \text{gap_extend} + Y[i-1, j] \right\} \]

match between x and y

gap in x

gap in y

If previous alignment ends in match, this is a new gap
Affine Base Cases

- $M[0, i] =$ “score of best alignment between 0 characters of $x$ and $i$ characters of $y$ that ends in a match” $= -\infty$ because no such alignment can exist.

- $X[0, i] =$ “score of best alignment between 0 characters of $x$ and $i$ characters of $y$ that ends in a gap in $x$” $= \text{gap}_\text{start} + i \times \text{gap}_\text{extend}$ because this alignment looks like:

  

  

  ───────
  YYYYYYYYY

- $X[i, 0] =$ “score of best alignment between $i$ characters of $x$ and 0 characters of $y$ that ends in a gap in $X$” $= -\infty$ because this alignment is not allowed:

  

  ───────
  xxxxxxxx-

- $M[i, 0] = M[0, i]$ and $Y[0, i]$ and $Y[i, 0]$ are computed using the same logic as $X[i, 0]$ and $X[0, i]$.
Affine Gap Runtime

- $3n^2$ subproblems
- Each one takes constant time
- Total runtime $O(n^2)$, back to the run time of the basic running time.
Why do you “need” 3 matrices?

- Alternative **WRONG** algorithm:

  \[
  M[i][j] = \max( \\
  M[i-1][j-1] + \text{cost}(x[i], y[i]), \\
  M[i-1][j] + \text{gap} + (\text{gap\_start if } \text{Arrow[i-1][j] != \leftarrow}), \\
  M[j][i-1] + \text{gap} + (\text{gap\_start if } \text{Arrow[i][j-1] != \downarrow}) \\
  )
  \]

**WRONG Intuition:** we only need to know whether we are starting a gap or extending a gap.

The arrows coming out of each subproblem tell us how the best alignment ends, so we can use them to decide if we are starting a new gap.

**PROBLEM:** The best alignment for strings \(x[1..i]\) and \(y[1..j]\) doesn’t have to be used in the best alignment between \(x[1..i+1]\) and \(y[1..j+1]\)
Why 3 Matrices: Example

match = 10, mismatch = -2, gap = -7, gap_start = -15

\[
\text{OPT}(4, 3) = \text{optimal score} = 30 - 15 - 7 = 8
\]

\[
\text{WRONG}(5, 3) = 30 - 15 - 7 - 15 - 7 = -14
\]

\[
\text{OPT}(5, 3) = 20 - 2 - 15 - 14 = -11
\]

this is why we need to keep the X and Y matrices around. they tell us the score of ending with a gap in one of the sequences.
Side Note: Lower Bounds

- Suppose the lengths of x and y are n.

- Clearly, need at least $\Omega(n)$ time to find their global alignment (have to read the strings!)

- The DP algorithms show global alignment can be done in $O(n^2)$ time.

- A trick called the “Four Russians Speedup” can make a similar dynamic programming algorithm run in $O(n^2 / \log n)$ time.
  - We probably won’t talk about the Four Russian’s Speedup.

- Open questions: Can we do better? Can we prove that we can’t do better? No one knows...
Recap

• Local alignment: extra “0” case.

• General gap penalties require 3 matrices and $O(n^3)$ time.
• Affine gap penalties require 3 matrices, but only $O(n^2)$ time.
What you should know by now...

• Dynamic programming framework

• Global & local sequence alignment algorithms with basic gap penalties

• Alignment with general gap penalties

• Alignment with affine gap penalties

• Longest common subsequence

• Subset Sum