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

AAAGAATTCA
A–A–A–T–CA

vs.

AAAGAATTCA
AAA----TCA

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

AAAGAACC
AAA----

vs.

AAAGAATTC
AAA----

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:

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

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

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

$$Y[i, j] = \max \begin{cases} X[i - k, j] - \text{gap}(k) \\ M[i - k, j] - \text{gap}(k) \end{cases}$$
The X (and Y) matrices

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

\( k \) 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.
The M Matrix

We now keep 3 different matrices:

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

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

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

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

Gaps start and end in the \( M \) matrix.
Running Time for Gap Penalties

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

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

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

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*:
  
  \[
  \text{gap}_\text{start} = \text{the cost of starting a gap} \\
  \text{gap}_\text{extend} = \text{the cost of extending a gap by one more space}
  \]

- 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.

  \[
  \text{gap}(k) = -\left(\sigma + (k - 1) \cdot \epsilon\right)
  \]
Affine gap algorithm as a finite state machine
Affine Gap Penalties

\[ M[i, j] = \max \begin{cases} X[i, j] & \text{gap closing} \\ M[i - 1, j - 1] + \text{SCORE}(x[i], y[j]) & \\ Y[i, j] \end{cases} \]

\[ X[i, j] = \max \begin{cases} X[i, j - 1] - \epsilon & \text{gap extension} \\ M[i - 1, j - 1] - \sigma & \text{gap opening} \end{cases} \]

\[ Y[i, j] = \max \begin{cases} Y[i - 1, j] - \epsilon & \\ M[i - 1, j] - \sigma \end{cases} \]
Affine Gap Runtime

- $3mn$ subproblems
- Each one takes constant time
- Total runtime $O(mn)$:
  - back to the run time of the basic running time.

Traceback

- Arrows now can point between matrices.
- The possible arrows are given, as usual, by the recurrence.
- E.g. What arrows are possible leaving a cell in the M matrix?
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.
Global Alignment in Linear Space


- Recall: Dynamic programming algorithms discussed so have $O(nm)$ time and space complexity

- Key idea:
  - We can get the optimal alignment score in space $O(n)$.
  - Can we reconstruct the optimal alignment in space $O(n)$?
Global Alignment in Linear Space


- Recall: Dynamic programming algorithms discussed so have $O(nm)$ time and space complexity

- Key idea:
  - We can get the optimal alignment score in space $O(n)$.
  - Can we reconstruct the optimal alignment in space $O(n)$?
    - Use recursion (divide and conquer) to do reconstruction.
Score:
ATCAA
A–CGA
= Score:
ATC
A–C
+ Score:
AA
GA

Assuming we know that optimal alignment goes through this node
Generally:

\[
\text{SCORE}(x_{0n}, y_{0m}) = \max_t \left[ \text{SCORE}(x_{0t}, y_{0 \frac{m}{2}}) + \text{SCORE}(x_{tn}, y_{\frac{m}{2}m}) \right]
\]

\(x_{ij}\): substring starting at position \(i\) ending at position \(j\)
We know how to calculate first term, what about second term?

\[ s_{n,m} = \max_t \left[ s_t, \frac{m}{2} \right] + \text{SCORE}(x_{tn}, y_{\frac{m}{2}m}) \]

\( x_{ij} \): substring starting at position \( i \) ending at position \( j \)
We know how to calculate first term, what about second term?

Score is invariant to string reversal:

$$\text{SCORE}(x_{ij}, y_{kl}) = \text{SCORE}(x_{ji}, y_{lk})$$
Reversed!

\[ s_{n,m} = \max_t s_{t, \frac{m}{2}} + s_{t, \frac{m}{2}} \]
Reversed!

Last backtrack pointer on reversed score gives us ‘middle edge’

\[ s_{n,m} = \max_t s_t, \frac{m}{2} + s^r_t, \frac{m}{2} \]
Analysis

• Space: O(n) for two columns required to compute score

• Time: O(nm) to compute all scores (there is some O(n) double counting)

• After finding ‘middle edge’, we have two O(nm/4) problems:
  • solve each in linear space
  • solve each in O(nm/4) time
  • so O(nm/2) time

• Overall we have O(nm + nm/2 + nm/4 + nm/8 + …) = O(nm)
## LCS Example

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

```
G A G C A A T T
0 0 0 0 0 0 0 0 0
A 0 0 1 1 1
C 0 0 0 1 2
T 0 0 0 1 2
T 0 0 0 1 2
A 0 0 1 1 2
A 0 0 1 1 2
T 0 0 0 1 2
T 0 0 0 1 2
```

```
T T A A C G A G
0 0 0 0 0
T 0 1 1 1 1
T 0 1 2 2 2
A 0 1 2 3 3
A 0 1 2 3 4
T 0 1 2 3 4
T 0 1 2 3 4
C 0 1 2 3 4
A 0 1 2 3 4
```
LCS Example