

Gap Penalties

CMSC 423

General Gap Penalties

AAAGAATTCA
A-A-A-T-CA

vs.

AAAGAATTCA
AAA-----TCA

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.

- Now, the cost of a run of k gaps is $GAP \times k$
- A solution to the problem above is to support general gap penalty, so that the score of a run of k gaps is $gap(k) < GAP \times k$.
- Then, the optimization will prefer to group gaps together.

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:

AAAGAAC|
AAA-----|

vs.

AAAGAAAT|C
AAA-----|

Instead, we need to “know” how the previous alignment ends in order to give a score to the last subproblem.

Three Matrices

We now keep 3 different matrices:

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

$X[i,j]$ = score of best alignment of $x[1..i]$ and $y[1..j]$ ending with a **space in X**.

$Y[i,j]$ = score of best alignment of $x[1..i]$ and $y[1..j]$ ending with a **space in Y**.

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

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

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

The M Matrix

We now keep 3 different matrices:

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

$X[i,j]$ = score of best alignment of $x[1..i]$ and $y[1..j]$ ending with a **space in X**.

$Y[i,j]$ = score of best alignment of $x[1..i]$ and $y[1..j]$ ending with a **space in Y**.

By definition, alignment ends in a match.

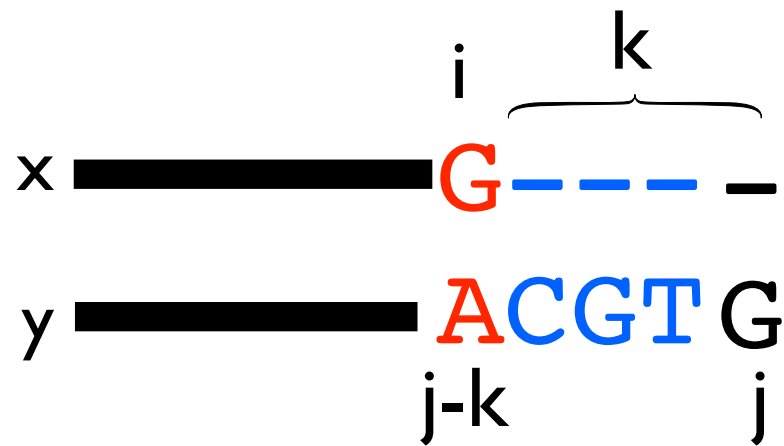
$$M[i, j] = \text{match}(i, j) + \max \begin{cases} M[i-1, j-1] \\ X[i-1, j-1] \\ Y[i-1, j-1] \end{cases}$$

Any kind of alignment is allowed before the match.

————— A
————— G



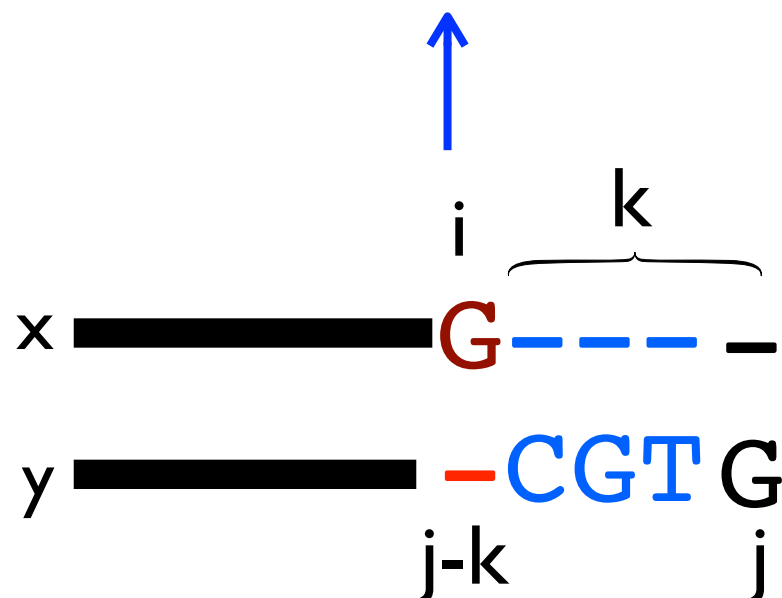
The X (and Y) matrices



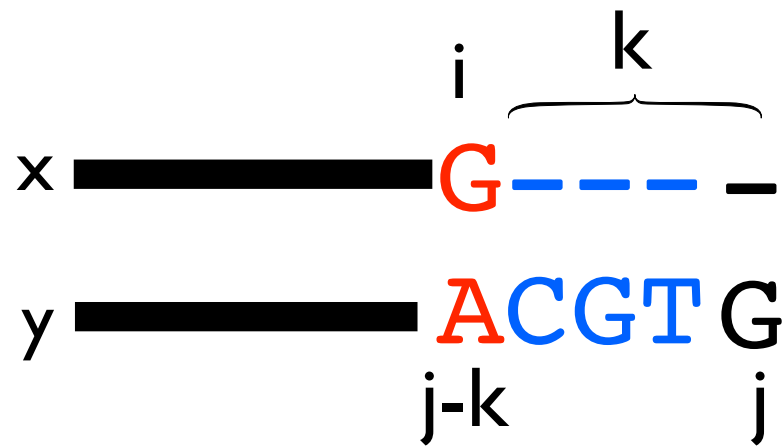
k decides how long to make the gap.

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

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



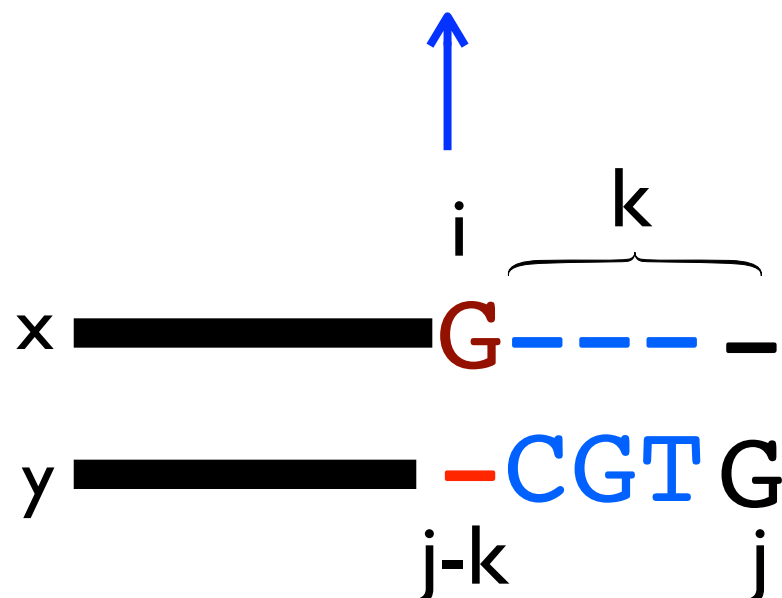
The X (and Y) matrices



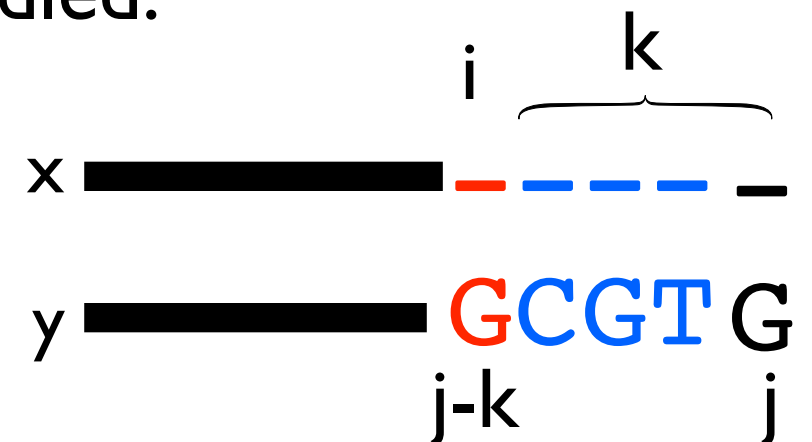
k decides how long to make the gap.

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

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



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*:
 - gap_start = the cost of starting a gap
 - gap_extend = 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

$M[i, j]$ = match between x and y

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

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

$X[i, j]$ = gap in x

$$X[i, j] = \max \begin{cases} \text{gap_start} + \text{gap_extend} + M[i, j-1] \\ \text{gap_extend} + X[i, j-1] \\ \text{gap_start} + \text{gap_extend} + Y[i, j-1] \end{cases}$$

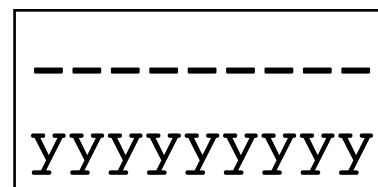
$Y[i, j]$ = gap in y

$$Y[i, j] = \max \begin{cases} \text{gap_start} + \text{gap_extend} + M[i-1, j] \\ \text{gap_start} + \text{gap_extend} + X[i-1, j] \\ \text{gap_extend} + Y[i-1, j] \end{cases}$$

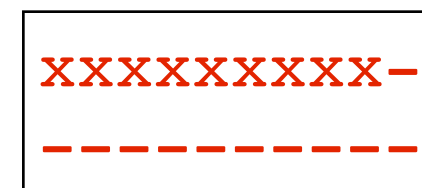
Affine Base Cases

- $M[0, i]$ = “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” = $\text{gap_start} + i \times \text{gap_extend}$ because this alignment looks like:



- $X[i, 0]$ = “score of best alignment between i characters of x and 0 characters of y that ends in a gap in X” = $-\infty$



← not allowed

- $M[i, 0] = M[0, i]$ 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?

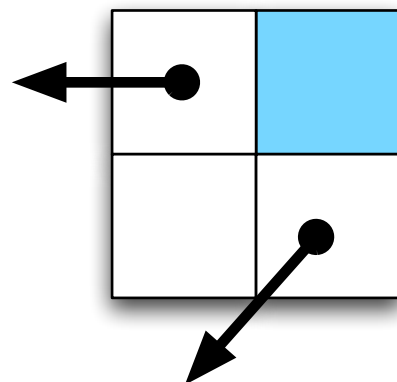
- Alternative **WRONG** algorithm:

```
M[i][j] = max(  
    M[i-1][j-1] + cost(x[i], y[i]),  
    M[i-1][j] + gap + (gap_start if Arrow[i-1][j] != ← ),  
    M[j][i-1] + gap + (gap_start if Arrow[i][j-1] != ↓ )  
)
```

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.

The best alignment
up to this cell ends
in a gap.



The best alignment
up to this cell ends
in a match.

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

CART
CA-T

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

CARTS
CA-T-

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

CARTS
CAT--

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



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 won’t talk about the Four Russian’s Speedup, but it’s in your book in Sections 7.3 & 7.4.
- Open questions: Can we do better? Can we prove that we can’t do better? No one knows...

Recap

- Semiglobal alignment: 0 initial columns or take maximums over last row or column.
- 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...

- Global & local sequence alignment algorithms with basic gap penalties
- Alignment with general gap penalties
- Alignment with affine gap penalties
- Longest common subsequence
- Dynamic programming framework