General gap penalty function

• Using a general $\gamma$ gives an $O(mn^2 + nm^2)$ algorithm.

• Can we still achieve our old $O(mn)$ time bound?

What was the problem?

• For each additional gap, we have a different additional score.

• We had to accommodate for every possible gap length.

• Restrict the possibilities. Use an affine gap function to approximate the general one.
Affine gap penalty function

\[ \gamma(x) = e + d(x - 1) \quad x \geq 1. \]

How can this help

- Any gap length greater than one is penalized linearly.
- Need to distinguish only between 1 and more than 1.
- \( d = 1 \)
  - penalize by \( 1 \).
- \( d > 1 \)
  - penalize linearly by increments of \( d \).
- Use more than one matrix to detect that, depends on where we stopped last time.

Looking for additivity

- Break the alignment into 3 kinds of blocks:
  - Two aligned characters
  - Consecutive gaps aligned with \( x \)
  - Consecutive gaps aligned with \( y \)

\[
A|A|C|---|A|A|\text{T|C|G|G|T|A|C|G|G|C|}\n\]

- Score is additive across block boundaries, this is true regardless of the gap penalty function.
Methodology

- Instead of reasoning on the last column in an alignment, we reason on the last block.
- For each pair \( (i, j) \), keep the best score for \( x_1...x_i \) and \( y_1...y_j \) that end with a particular block.
- We need three matrices
  - \( A \) for block type 1
  - \( B \) for block type 2, gaps aligned with \( x \)
  - \( C \) for block type 3, gaps aligned with \( y \)

Modified Needleman-Wunsch for affine gaps

\[
\begin{align*}
A(i, j) &= \sigma(i, j) + \max \begin{cases} 
A(i-1, j-1) \quad & A(i, j) \text{ is the start of block of type 1, so it aligns } x_i \text{ with } y_j \text{ no matter what.} \\
B(i-1, j-1) \quad & B \text{ for block type 2, only need to vary } i. \text{ First gap penalized } -d, \\
C(i-1, j-1) \quad & \text{i.e. after block type 1 and 3, otherwise } -d.
\end{cases} \\
B(i, j) &= \max \begin{cases} 
A(i, j-1) - e \quad & C \text{ for block type 3, only need to vary } j. \text{ First gap penalized } -e, \\
B(i, j-1) - d \quad & \text{i.e. after block type 1 and 2, otherwise } -d.
\end{cases} \\
C(i, j) &= \max \begin{cases} 
A(i-1, j) - e \quad & \\
B(i-1, j) - d \quad & \end{cases}
\end{align*}
\]

Initialization

\[
\begin{align*}
A(0, 0) &= 0 \\
A(i, 0) &= ? - \infty \text{ not block type 1} \\
A(0, j) &= ? - \infty \\
B(0, 0) &= ? - \infty \text{ not block type 2} \\
B(i, 0) &= -e - d(i - 1) \quad 1 \leq i \leq m \\
C(0, 0) &= ? - e - d(j - 1) \quad 1 \leq j \leq n \\
C(i, 0) &= ? - \infty \text{ not block type 3}
\end{align*}
\]
Simplification

\[
\begin{align*}
A(i, j) &= s(i, j) + \max \left\{ A(i-1, j-1), B(i-1, j-1) \right\} \\
B(i, j) &= \max \left\{ A(i-1, j) - e, B(i-1, j) - d, C(i-1, j) \right\} \\
C(i, j) &= \max \left\{ A(i, j-1) - e, B(i, j-1) - d, C(i, j-1) \right\}
\end{align*}
\]

Multiple sequence alignment

- Align \( k \) sequences in the best way
- What is the score (assume additive)?
  - In each column, we have \( k \) characters.
  - Scoring function takes \( k \) arguments.
  - Need \( O(2^k) \) entries (at least gap non-gap)!  
- Practical scoring function: Sum of Pairs (SP-score)
  - Let \( s(x, y) \) be the score of the induced alignment for sequences \( x \) and \( y \), i.e., the score of the alignment obtained by isolating \( x \) and \( y \) and ignoring columns with only gaps.
  - SP-score = \( \sum_{x \neq y} s(x, y) \)

SP-score function

- Example: three sequences \( x, y, \) and \( z \)
  - \( SP = s(x, y) + s(x, z) + s(y, z) \)
- Nice properties
  - This is independent of the order of characters in a column
  - It rewards similarities and penalizes differences.
- Assume additive
  - \( s(x, y) = \Sigma s(x, y), \quad s(-) = 0 \)
Example

- \( w \) = ATG, \( x \) = ATG, \( y \) = A, \( z \) = T

\[
\begin{array}{c|c|c}
ATG & ATG & \vspace{1ex} \\
A & - & A \\
\vspace{1ex} & T & - \\
\end{array}
\]

- Score = 3 - 3 - 3 - 3 - 4 = -13

- The induced alignment between a pair of sequences is not necessarily an optimal one, e.g. \( y \) and \( z \).

Dynamic Programming

- \( k \) sequences of length \( n_i \) each.

- \( k \) dimensional array \( A \) of length \( n_i + 1 \) in each direction.

- \( A(i, \ldots, j) \) holds the score of the optimal alignment involving \( x_i[1\ldots i] \), \( \ldots \), \( x_j[1\ldots j] \).

- A now requires \( O(n^k) \) space.

Dynamic Programming (cont.)

- Now each entry in \( A \) depends on \( 2^n - 1 \) entries (note: \( k = 2 \rightarrow 3 \) (why)?)
  - e.g. \( A(i, j, \ldots, j) \) depends on:
    - \( A(i, j, \ldots, j) = 001 \)
    - \( A(i, j, \ldots, j) = 010 \)
    - \( A(i, j, \ldots, j) = 011 \)
    - \( A(i, j, \ldots, j) = 100 \)
    - \( A(i, j, \ldots, j) = 101 \)
    - \( A(i, j, \ldots, j) = 110 \)
    - \( A(i, j, \ldots, j) = 111 \)

- Computing the SP-score in each case requires \( O(k^2) \) time.

- Total running time is \( O(k^2n^k) \).
Heuristic: Star Alignment

- Star alignment is a special case of tree alignments.
- What is a tree alignment?
- Given a tree with \( k \) nodes representing \( k \) sequences, a multiple alignment of the \( k \) sequences consistent with the tree is such that the induced alignment between \( x_i \) and \( x_j \) is optimal if there is an edge \((x_i, x_j)\).

Example

A multiple alignment consistent with the tree:

- \( s_1 \): A X X A Z
- \( s_2 \): A X Z A Z
- \( s_3 \): A X X A Z
- \( s_4 \): A Y X A Z
- \( s_5 \): A y X X Z

Tree alignment

- Given a tree, is it always possible to obtain a multiple alignment consistent with the tree?
  - YES
- How?
Algorithm: step 1

- Pick $x_i$ and $x_j$ such that $(x_i, x_j)$ is an edge and align them optimally.
- set $X = \{x_i, x_j\}$, the set of aligned sequences.

Algorithm: step 2

- Pick $x_i \notin X$ and $x_j \in X$ such that $(x_i, x_j)$ is an edge.
- Align $x_i$ and $x_j$ optimally.
- **Once a gap always a gap**: For each gap added to $x_j$ in this alignment, add a corresponding gap to sequences in $X$. For each gap already in $x_j$ add a corresponding gap in $x_i$ (if needed).
- $X = X \cup \{x_i\}$

Algorithm: step 3

- Repeat step 2 until all sequences are in $X$. 
Example revisited

Star Alignment

- Special case where tree is a star
- Which sequence should be the center of the star?
- The sequence $x_i$ such that $M = \sum_{j \in \mathcal{I}} \text{OPT}(x, x_j)$ is maximized.

Star alignment algorithm

- Pick $x$ to maximize $M = \sum_{j \in \mathcal{I}} \text{OPT}(x, x_j)$
- Find the optimal alignments between $x$ and all $x_i$.
- Join the alignments using once a gap always a gap technique.
- Running time = $O(\mathcal{K}^2 \mathcal{P})$ for alignments + $O(\mathcal{K}L)$ for gap updates, where $L$ is the length of the alignment and each time a sequence is joined, at most $L$ sequences of length at most $L$ must be updated $\Rightarrow O(k,kL) = O(kL)$
Example

$x_1 = ATGECATTT$
$x_2 = ATGGECCATT$
$x_3 = ATCCAAATTT$
$x_4 = ATCTTCCTTT$
$x_5 = ACTGACC$

$x_5$ maximizes $N$

$x_1, ATGECATTT$
$x_2, ATGGECCATT$
$x_3, ATCCAAATTT$
$x_4, ATCTTCCTTT$
$x_5, ACTGACC$