Affine Gap Penalty Function

BCB 597
Fall, 2007
Multiple Base Insertion/Deletion Mutation Event
Constant Gap Penalty

• Each gap character is penalized a constant amount: $\gamma$
• Not a good gap penalty for our mutation event model
• What if we had a more general gap penalty function?
General Gap Penalty Function

- Gap penalty function: $W(g)$
- Each entry in the dynamic programming table now takes $O(n)$ time to fill.

\[
\begin{align*}
S[0,0] &= 0 & \text{• } O(n^3) \text{ running time} \\
S[i,0] &= W(i) & \text{• } \text{Cannot directly apply space saving} \\
S[0,j] &= W(j) & \text{• } O(nm) \text{ space} \\
S[i,j] &= \max \left\{ S[i-1,j-1] + \delta(i-1, j-1), \max_{k=1}^{i} \left( S[i-k,j] + W(k) \right), \max_{k=1}^{j} \left( S[i,j-k] + W(k) \right) \right\}
\end{align*}
\]
Affine Gap Penalty

• $W(i) = g + hi$ (for $i \geq 1$)
• $g$: gap opening penalty
• $h$: gap extension penalty
• The ratio between $g$ and $h$ affects how much we view the size of the gap.
  – Small $g$, Large $h$: size more important
  – Large $g$, Small $h$: size less important
Remember our “What if?”

\[ E(\mathbb{A}_A, \mathbb{B}_A) = E(\mathbb{A}_A[0,i], \mathbb{B}_A[0,i]) + E(\mathbb{A}_A[i+1,L], \mathbb{B}_A[i+1,L]) \]

\[
\begin{align*}
E(\text{ACTAC-CTG}) &= E(\text{ACTAC}) + E(\text{CTG}) \\
E(\text{AC-AGACTA}) &= E(\text{AC-AG}) + E(\text{ACTA}) 
\end{align*}
\]
Alignment Decomposition
No Longer Behaves

\[
\begin{align*}
\alpha &= 1, \beta = -1, g = -3, h = -1 \\
\text{However, this was a sufficient condition, not a necessary condition!}
\end{align*}
\]
Amortize the cost of opening

• Consider filling out the dynamic programming table
• The cost of $g$ is eventually spread out over the length of the gap once the gap is closed.
• While gap opening might initially cause the alignment score to be less than a mismatch.
• However, the one time loss might eventually become “worth it.”
• We cannot predict ahead to know how much the gap opening will cost per gap character, or we would directly know if the gap opening will be worth it.

\[
\begin{align*}
\text{AGCTAAGGAA} \\
\text{AGCTAAGGAC} \\
> \\
\text{AGCTAAGGA} - \\
\text{AGCTAAGGAC}
\end{align*}
\]

\[
\begin{align*}
\text{AGCTAAGGAAA} \\
\text{AGCTAAGGACT} \\
> \\
\text{AGCTAAGGA} -- \\
\text{AGCTAAGGACT}
\end{align*}
\]

\[
\begin{align*}
\text{AGCTAAGGAAAA} \\
\text{AGCTAAGGACTT} \\
< \\
\text{AGCTAAGGA} --- \\
\text{AGCTAAGGACTT}
\end{align*}
\]
Three Dynamic Programming Tables

• Instead of looking to the future, we simply store the best possible alignments that end in gaps, in case they might eventually become “worth it.”

• Table $G_A$: Stores the best alignment, under the restriction that the last character in $A_A$ be aligned with a gap.

• Table $G_B$: Stores the best alignment score, under the restriction that the last character in $B_A$ be aligned with a gap.

• Table $S$: Stores the best alignment score with no restrictions.
Three Recursions

\[ G_A[0, j] = -\infty \]
\[ G_B[i, 0] = -\infty \]
\[ S[0, j] = g + jh \]
\[ S[i, 0] = g + ih \]
\[ S[0, 0] = 0 \]

\[ G_A[i, j] = \max \begin{cases} G_A[i - 1, j] + h \\
S[i - 1, j] + g + h \end{cases} \]
\[ G_B[i, j] = \max \begin{cases} G_B[i, j - 1] + h \\
S[i, j - 1] + g + h \end{cases} \]
\[ S[i, j] = \max \begin{cases} S[i - 1, j - 1] + \delta(i - 1, j - 1) \\
G_A[i, j] \\
G_B[i, j] \end{cases} \]
Traceback

- The path can now travel
- Start in table $S$
- If $S[i,j] = G_A[i,j]$ or $G_B[i,j]$, jump to the corresponding table, without making a “move.”
- Paths through $G_A$ must move up
- Paths through $G_B$ must move left
Semiglobal Alignment

\[ G_A[0, j] = -\infty \]
\[ G_B[i, 0] = -\infty \]
\[ S[0, j] = 0 \]
\[ S[i, 0] = 0 \]
\[ S[0, 0] = 0 \]

\[ G_B[i, j] = \max \begin{cases} G_A[i - 1, j] + h \\ S[i - 1, j] + g + h \end{cases} \]

\[ S[i, j] = \max \begin{cases} S[i - 1, j - 1] + \delta(i - 1, j - 1) \\ G_A[i, j] \\ G_B[i, j] \end{cases} \]
Local Alignment

- The key to Local Alignment is that the aligning region must still start and end with a matching character.
- This means that a local alignment path must start and end in table S.
- Thus finding local alignment is very similar to local alignment for constant gap penalty.
Local Alignment

\[
G_A[i, j] = \max \begin{cases} 
G_A[i-1, j] + h \\
S[i-1, j] + g + h 
\end{cases}
\]

\[
G_B[i, j] = \max \begin{cases} 
G_B[i, j-1] + h \\
S[i, j-1] + g + h 
\end{cases}
\]

\[
S[i, j] = \max \begin{cases} 
S[i-1, j-1] + \delta(i-1, j-1) \\
G_A[i, j] \\
G_B[i, j] \\
0 
\end{cases}
\]

- \( G_A[0, j] = -\infty \)
- \( G_B[i, 0] = -\infty \)
- \( S[0, j] = 0 \)
- \( S[i, 0] = 0 \)
- \( S[0, 0] = 0 \)
Applying Space Saving To Affine Gap Alignment
Finding the Intersection

\[
\max_j \begin{cases} 
S[i, j] + S[i + 1, j] + g + h & j \rightarrow j \\
G_A[i, j] + G_A[i + 1, j] - g + h & j \rightarrow j \\
S[i, j] + G_A[i + 1, j] + h & j \rightarrow j \\
G_A[i, j] + S[i + 1, j] + h & j \rightarrow j \\
S[i, j] + S[i + 1, j + 1] + \delta(i, j) & j \rightarrow j + 1 
\end{cases}
\]
Diagonal Intersection
Vertical Intersection