Dynamic Programming Method for Analyzing Biomolecular Sequences

Tao Jiang
Department of Computer Science
University of California - Riverside
(Type set by Kun-Mao Chao)

E-mail: jiang@cs.ucr.edu
http://www.cs.ucr.edu/~jiang

Outline
• The paradigm of dynamic programming
• Sequence alignment – a general framework for comparing sequences in bioinformatics
• Dynamic programming algorithms for sequence alignment
• Techniques for improving the efficiency of the algorithms
• Multiple sequence alignment

Dynamic Programming
• Dynamic programming is an algorithmic method for solving optimization problems with a compositional/recursive cost structure.
• Richard Bellman was one of the principal founders of this approach.

Two key ingredients
• Two key ingredients for an optimization problem to be suitable for a dynamic programming solution:
  1. optimal substructures
  2. overlapping subinstances
  Each substructure is optimal. (principle of optimality)
  (Otherwise, a divide-and-conquer approach is the choice.)
Three basic components

- The development of a dynamic programming algorithm has three basic components:
  - A recurrence relation (for defining the value/cost of an optimal solution);
  - A tabular computation (for computing the value of an optimal solution);
  - A backtracing procedure (for delivering an optimal solution).

Fibonacci numbers

The Fibonacci numbers are defined by the following recurrence:

\[
F_0 = 0 \\
F_1 = 1 \\
F_i = F_{i-1} + F_{i-2} \quad \text{for } i > 1.
\]

How to compute \(F_{10}\)?

Tabular computation

- Tabular computation can avoid redundant computation steps.

<table>
<thead>
<tr>
<th></th>
<th>(F_0)</th>
<th>(F_1)</th>
<th>(F_2)</th>
<th>(F_3)</th>
<th>(F_4)</th>
<th>(F_5)</th>
<th>(F_6)</th>
<th>(F_7)</th>
<th>(F_8)</th>
<th>(F_9)</th>
<th>(F_{10})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>21</td>
<td>34</td>
<td>55</td>
</tr>
</tbody>
</table>
Maximum sum interval

• Given a sequence of real numbers $a_1, a_2, \ldots, a_n$, find a consecutive subsequence with the maximum sum.

For each position, we can compute the maximum-sum interval starting at that position in $O(n)$ time. Therefore, a naive algorithm runs in $O(n^2)$ time.

$O$-notation: an asymptotic upper bound

• $f(n) = O(g(n))$ iff there exist two positive constant $c$ and $n_0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

For example, $5n + 108 = O(n)$ and $2n = O(n \log n)$.

How functions grow?

<table>
<thead>
<tr>
<th>Function</th>
<th>30s</th>
<th>$92n \log n$</th>
<th>$26n^2$</th>
<th>0.68$n^3$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.0 sec</td>
<td>0.001 sec</td>
<td>0.26 sec</td>
<td>0.48 sec</td>
<td>$4 \times 10^6$ sec</td>
</tr>
<tr>
<td>100,000</td>
<td>1.0 min</td>
<td>2.6 min</td>
<td>3.0 days</td>
<td>22 yr</td>
<td></td>
</tr>
</tbody>
</table>

(Assume one million operations per second.)

For large data sets, algorithms with a complexity greater than $O(n \log n)$ are often impractical!

Maximum sum interval

• Define $S(i)$ to be the maximum sum of the intervals ending at position $i$.

$$S(i) \leftarrow a_i + \max \left\{ S(i-1), 0 \right\}$$

If $S(i-1) < 0$, concatenating $a_i$ with its previous interval gives less sum than $a_i$ itself.
Maximum sum interval (tabular computation)

\[
\begin{array}{ccccccccccccccc}
    & 9 & -3 & 1 & 7 & -15 & 2 & 3 & -4 & 2 & -7 & 3 & -2 & 8 & 4 & -8 \\
S(i) & 9 & 8 & 6 & 7 & 14 & -1 & 2 & 5 & -1 & 3 & -4 & 8 & 4 & 12 & 10 & 7 \\
\end{array}
\]

The maximum sum

\[\text{S}(6) = 6 \cdot 2 \cdot 8 \cdot 4\]

Running time: \(O(n)\).

Maximum sum interval (backtracing)

\[
\begin{array}{ccccccccccccccc}
    & 9 & -3 & 1 & 7 & -15 & 2 & 3 & -4 & 2 & -7 & 3 & -2 & 8 & 4 & -8 \\
S(i) & 9 & 8 & 6 & 7 & 14 & -1 & 2 & 5 & -1 & 3 & -4 & 8 & 4 & 12 & 10 & 7 \\
\end{array}
\]

The maximum-sum interval: 6 -2 8 4.

Defining scores for alignment columns

- **infocon** [Stojanovic et al., 1999]
  - Each column is assigned a score that measures its information content, based on the frequencies of the letters both within the column and within the alignment.

\[
\begin{array}{cccccccc}
    & \text{GATCAT} & = & \text{GGA} \\
\text{GFAACTTGGAA} & \text{GACATAGTA} \\
\end{array}
\]

Defining scores (cont’d)

- **phylogen** [Stojanovic et al., 1999]
  - Columns are scored based on the evolutionary relationships among the sequences implied by a supplied phylogenetic tree.

\[
\begin{array}{cccccccc}
    \text{Score} = 1 & \text{Score} = 2 \\
\end{array}
\]
Two fundamental problems we solved (joint work with Lin and Chao)

• Given a sequence of real numbers of length $n$ and an upper bound $U$, find a consecutive subsequence of length at most $U$ with the maximum sum --- an $O(n)$-time algorithm.

\[ U = 3 \]
\[ 9 -3 1 7 -15 2 -3 -2 -7 6 -2 \]

Two fundamental problems we solved (joint work with Lin and Chao)

• Given a sequence of real numbers of length $n$ and a lower bound $L$, find a consecutive subsequence of length at least $L$ with the maximum average --- an $O(n \log L)$-time algorithm. This has been improved to $O(n)$ by others later.

\[ L = 4 \]
\[ 3 2 14 6 6 2 \underline{10} 2 6 6 10 2 1 \]

Another example

Given a sequence as follows:
2, 6.6, 6.6, 3, 7, 6, 7, 2
and $L = 2$, the highest-average interval is the squared area, which has the average value $20/3$.

2, 6.6, 6.6, 3, $\underline{6, 7}$

GC-rich regions

• Our method can be used to locate a region of length at least $L$ with the highest C+G ratio in $O(n \log L)$ time.

```
ATGACTCGAGCTCGTCA
0010101100101010
```

Search for an interval of length at least $L$ with the highest average.

```
ATGACTCGAGCTCGTCA
0010101100101010
```
Length-unconstrained version

- Maximum average interval

3 2 6 6 2 10 2 6 6 14 2 1

The maximum element is the answer. It can be done in $O(n)$ time.

A naive algorithm

- A simple shift algorithm can compute the highest-average interval of a fixed length in $O(n)$ time

- Try $L, L+1, L+2, \ldots, n$. In total, $O(n^2)$.

A pigeonhole principle

- Notice that the length of an optimal interval is bounded by $2L$, we immediately have an $O(nL)$-time algorithm.

We can bisect a region of length $\geq 2L$ into two segments, where each of them is of length $\geq L$.

Future Development

- Best $k$ (nonintersecting) subsequences?
- Max-average with both upper and lower length bounds
- General (gapped) local alignment with length upper bound.
- Measurement of goodness?
Longest increasing subsequence (LIS)

- The longest increasing subsequence is to find a longest increasing subsequence of a given sequence of distinct integers $a_1, a_2, \ldots, a_n$.

  e.g. $9 \ 2 \ 5 \ 3 \ 7 \ 11 \ 8 \ 10 \ 13 \ 6$

  \begin{align*}
  2 & \ 3 \ 7 \ \downarrow \text{are increasing subsequences.} \\
  5 & \ 7 \ 10 \ 13 \ \downarrow \\
  9 & \ 7 \ 11 \ \downarrow \\
  3 & \ 5 \ 11 \ 13 \ \downarrow
\end{align*}

  We want to find a longest one.

  \begin{align*}
  9 & \ 2 \ 5 \ 3 \ 7 \ 11 \ 8 \ 10 \ 13 \ 6 \\
  L[i] & = 1 + \max_{j=0..i-1} \{L[j] \mid a_j < a_i\}
\end{align*}

  (use a dummy $a_0 = \text{minimum}$, and $L[0] = 0$)

  \begin{align*}
  9 & \ 2 \ 5 \ 3 \ 7 \ 11 \ 8 \ 10 \ 13 \ 6 \\
  L[i] & \begin{cases}
    1 & i = 0 \\
    1 & i = 1 \\
    2 & i = 2 \\
    2 & i = 3 \\
    3 & i = 4 \\
    4 & i = 5 \\
    4 & i = 6 \\
    5 & i = 7 \\
    6 & i = 8 \\
    3 & i = 9
  \end{cases}
\end{align*}

The maximum length

The subsequence 2, 3, 7, 8, 10, 13 is a longest increasing subsequence.

This method runs in $O(n^2)$ time.

A naive approach for LIS

- Let $L[i]$ be the length of a longest increasing subsequence ending at position $i$.

  \[
  L[i] = 1 + \max_{j=0..i-1} \{L[j] \mid a_j < a_i\}
  \]

  (use a dummy $a_0 = \text{minimum}$, and $L[0] = 0$)

  \begin{align*}
  9 & \ 2 \ 5 \ 3 \ 7 \ 11 \ 8 \ 10 \ 13 \ 6 \\
  L[i] & \begin{cases}
    1 & i = 0 \\
    1 & i = 1 \\
    2 & i = 2 \\
    2 & i = 3 \\
    3 & i = 4 \\
    4 & i = 5 \\
    4 & i = 6 \\
    5 & i = 7 \\
    6 & i = 8 \\
    3 & i = 9
  \end{cases}
\end{align*}

The subsequence 2, 3, 7, 8, 10, 13 is a longest increasing subsequence.

This method runs in $O(n^2)$ time.

Binary Search

- Given an ordered sequence $x_1, x_2, \ldots, x_n$, where $x_1 < x_2 < \ldots < x_n$, and a number $y$, a binary search finds the largest $x_i$ such that $x_i < y$ in $O(\log n)$ time.

  \[
  a < \frac{a + b}{2} < \frac{a + c}{2} < \frac{b + c}{2} < \frac{b + d}{2} < \cdots
  \]
Binary Search

• How many steps would a binary search reduce the problem size to 1?
  
  \[ n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow n/16 \rightarrow \ldots \rightarrow 1 \]

  How many steps? \( O(\log n) \) steps.

  \[ n/2^s = 1 \]

  \[ \Rightarrow s = \log_2 n \]

An \( O(n \log n) \) method for LIS

• Define \( \text{BestEnd}[k] \) to be the smallest end number of an increasing subsequence of length \( k \).

  \[
  \begin{array}{cccccccc}
  9 & 2 & 5 & 3 & 7 & 11 & 8 & 10 & 13 & 6 \\
  2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
  3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
  7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\
  8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
  10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\
  13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 \\
  \end{array}
  \]

  For each position, we perform a binary search to update \( \text{BestEnd} \). Therefore, the running time is \( O(n \log n) \).

Longest Common Subsequence (LCS)

• A subsequence of a sequence \( S \) is obtained by deleting zero or more symbols from \( S \). For example, the following are all subsequences of “president”: pred, sdn, predent.

• The longest common subsequence problem is to find a maximum length common subsequence between two sequences.
LCS

For instance,
Sequence 1: president
Sequence 2: providence
Its LCS is priden.

Another example:
Sequence 1: algorithm
Sequence 2: alignment
One of its LCS is algm.

How to compute LCS?

- Let A = a1a2...am and B = b1b2...bn.
- len(i, j): the length of an LCS between a1a2...ai and b1b2...bj
- With proper initializations, len(i, j) can be computed as follows.

```
len(0, 0) = 0
len(i, 0) = len(i-1, 0) + 1
len(0, j) = len(0, j-1) + 1
len(i, j) = max(len(i-1, j-1), len(i-1, j), len(i, j-1))
```

procedure LCS-Length(A, B)
1. for i ← 0 to m do len(0, i) = 0
2. for j ← 1 to n do len(0, j) = 0
3. for i ← 1 to m do
4. for j ← 1 to n do
5. if aᵢ = bⱼ then
   len(i, j) = len(i-1, j-1) + 1
6. else if len(i-1, j) ≥ len(i, j-1) then
   len(i, j) = len(i-1, j)
7. else
   len(i, j) = len(i, j-1)
8. prev(i, j) ← "
9. return len and prev
The backtracing algorithm

procedure Output-LCS(A, prev, i, j)
if \( i = 0 \) or \( j = 0 \) then return
if \( \text{prev}(i, j) = "\) then
\[ \text{print} \] \( a_i \)
Output-LCS(A, prev, i-1, j)
else if \( \text{prev}(i, j) = \) then
Output-LCS(A, prev, i, j-1)
else
Output-LCS(A, prev, i-1, j-1)

The backtracing algorithm

procedure Output-LCS(A, prev, i, j)
if \( i = 0 \) or \( j = 0 \) then return
if \( \text{prev}(i, j) = "\) then
\[ \text{print} \] \( a_i \)
Output-LCS(A, prev, i-1, j-1)
else if \( \text{prev}(i, j) = \) then
Output-LCS(A, prev, i-1, j)
else
Output-LCS(A, prev, i, j-1)

The backtracing algorithm

procedure Output-LCS(A, prev, i, j)
if \( i = 0 \) or \( j = 0 \) then return
if \( \text{prev}(i, j) = "\) then
\[ \text{print} \] \( a_i \)
Output-LCS(A, prev, i-1, j)
else if \( \text{prev}(i, j) = \) then
Output-LCS(A, prev, i, j-1)
else
Output-LCS(A, prev, i-1, j-1)

Dot Matrix

Sequence A: CTTAACT
Sequence B: CGGATCAT

Output: priden
Pairwise Alignment

Sequence A: CTTAACT
Sequence B: CGGATCAT

An alignment of A and B:

```
C     G     G     A     T     C     A     T
C     T     T     A     A     C     T
```

A simple scoring scheme

- Match: +8 \( w(x, y) = 8 \), if \( x = y \)
- Mismatch: -5 \( w(x, y) = -5 \), if \( x \neq y \)
- Each gap symbol: -3 \( w(-, x) = w(x, -) = -3 \) (i.e. space)

```
C     G     G     A     T     C     A     T
C     T     T     A     A     C     T
+8   -3   -3   -8   -5   +8   -3   -3   +8 = +12
```

Alignment (or Edit) Graph

A simple scoring scheme
Scoring Matrices

- Amino acid substitution matrices
  - PAM
  - BLOSUM

- DNA substitution matrices
  - DNA is less conserved than protein sequences
  - Less effective to compare coding regions at nucleotide level

PAM

- Point Accepted Mutation (Dayhoff, et al.)
  - 1 PAM = PAM₃ = 1% average change of all amino acid positions
    - After 100 PAMs of evolution, not every residue will have changed
      - some residues may have mutated several times
      - some residues may have returned to their original state
      - some residues may not changed at all

PAMₓ

- PAMₓ = PAMₓ
  - E.g. PAM₂₅₀ = PAM₁²₅₀

PAM₂₅₀ is a widely used scoring matrix.

|       | Ala | Arg | Asn | Asp | Cys | Gln | Glu | Gly | His | Ile | Leu | Lys | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ala</td>
<td>1</td>
<td>1</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>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Arg</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Asn</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Asp</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Cys</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</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>
<td></td>
</tr>
<tr>
<td>Gln</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Glu</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Gly</td>
<td>12</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>His</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Ile</td>
<td>8</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Leu</td>
<td>12</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Lys</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

BLOSUM

- Blooms Substitution Matrix
  - Scores derived from observations of the frequencies of substitutions in blocks of local alignments in related proteins
  - Matrix name indicates evolutionary distance
    - BLOSUM₆₂ was created using sequences sharing no more than 62% identity
The Blosum50 Scoring Matrix

An optimal alignment
-- an alignment of maximum score

- Let $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$.
- $S_{i,j}$: the score of an optimal alignment between $a_1a_2...a_i$ and $b_1b_2...b_j$
- With proper initializations, $S_{i,j}$ can be computed as follows:

\[
S_{i,j} = \max \left\{ S_{i-1,j} + w(a_i -), S_{i,j-1} + w(-, b_j), S_{i-1,j-1} + w(a_i, b_j) \right\}
\]

Computing $S_{i,j}$

Initialization

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-3</td>
<td>-6</td>
<td>-9</td>
<td>-12</td>
<td>-15</td>
<td>-18</td>
<td>21</td>
<td>-24</td>
</tr>
<tr>
<td>C</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Global Alignment vs. Local Alignment

- global alignment:

- local alignment:
An optimal local alignment

- $S_{i,j}$: the score of an optimal local alignment ending at $a_i$ and $b_j$
- With proper initializations, $S_{i,j}$ can be computed as follows.

$$s_{i,j} = \max \begin{cases} \ 0 \\ s_{i-1,j} + w(a_i, -) \\ s_{i,j-1} + w(-, b_j) \\ s_{i-1,j-1} + w(a_i, b_j) \end{cases}$$

local alignment

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</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>G</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

Match: 8
Mismatch: -5
Gap symbol: -3

local alignment

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</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>G</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

$A-C-T$

A T C A T

$8+3+5+8+8 = 18$

Match: 8
Mismatch: -5
Gap symbol: -3

$A - C - T$

A T C A T

$8+3+5+8+8 = 18$
Affine gap penalties

- Match: +8 \( w(x, y) = 8 \), if \( x = y \)
- Mismatch: -5 \( w(x, y) = -5 \), if \( x \neq y \)
- Each gap symbol: -3 \( w(-, x) = -3 \), \( w(x, -) = -3 \)
- E.g. each gap is charged an extra gap-open penalty: -4.
- In general, a gap of length \( k \) should have penalty \( g(k) \)

\[
\begin{array}{cccccccc}
C & - & - & T & T & A & A & C & T \\
C & G & G & A & T & C & A & - & - \\
+8 & -3 & -3 & -3 & +8 & -5 & +8 & -3 & -3 \\
\text{alignment score: } 12 - 4 - 4 = 4
\end{array}
\]

Affine gap penalties

- A gap of length \( k \) is penalized \( x + k \cdot y \).
- Let \( D(i, j) \) denote the maximum score of any alignment between \( a_1, a_2, \ldots, a_i \) and \( b_1, b_2, \ldots, b_j \) ending with a deletion.
- Let \( I(i, j) \) denote the maximum score of any alignment between \( a_1, a_2, \ldots, a_i \) and \( b_1, b_2, \ldots, b_j \) ending with an insertion.
- Let \( S(i, j) \) denote the maximum score of any alignment between \( a_1, a_2, \ldots, a_i \) and \( b_1, b_2, \ldots, b_j \).

\[
\begin{align*}
D(i, j) &= \max (D(i-1, j) - y) \\
I(i, j) &= \max (I(i, j-1) - x - y) \\
S(i, j) &= \max (S(i-1, j-1) + w(a_i, b_j), D(i, j), I(i, j))
\end{align*}
\]
**Affine gap penalties**

(Gotoh’s algorithm)

\[
\begin{align*}
&\begin{array}{c}
D \\
I \\
S
\end{array} \\
&\begin{array}{c}
D \\
I \\
S
\end{array} \\
&w(a_i, b_j) - y
\end{align*}
\]

\[
\begin{align*}
&\begin{array}{c}
D \\
I \\
S
\end{array} \\
&\begin{array}{c}
D \\
I \\
S
\end{array} \\
&-x - y
\end{align*}
\]

\[
\begin{align*}
&\begin{array}{c}
D \\
I \\
S
\end{array} \\
&\begin{array}{c}
D \\
I \\
S
\end{array} \\
&-y
\end{align*}
\]

---

**k best local alignments**

- **Smith-Waterman**
  (Smith and Waterman, 1981; Waterman and Eggert, 1987)
  - linear-space version: sim (Huang and Miller, 1991)
  - linear-space variants: sim2 (Chao et al., 1995), sim3 (Chao et al., 1997)
- **FASTA**
  (Wilbur and Lipman, 1983; Lipman and Pearson, 1985)
  - linear-space band alignment (Chao et al., 1992)
- **BLAST**
  (Altschul et al., 1990; Altschul et al., 1997)
  - restricted affine gap penalties (Chao, 1999)

BLAST and FASTA are key genomic database search tools.

---

**FASTA**

1) Find runs of identities, and identify regions with the highest density of identities.
2) Re-score using PAM matrix, and keep top scoring segments.
3) Eliminate segments that are unlikely to be part of the alignment.
4) Optimize the alignment in a band.

Its running time is \(O(n)\).
FASTA

Step 1: Find runs of identities, and identify regions with the highest density of identities.

FASTA

Step 2: Re-score using PAM matrix, and keep top scoring segments.

FASTA

Step 3: Eliminate segments that are unlikely to be part of the alignment.

FASTA

Step 4: Optimize the alignment in a band.
BLAST

1) Build the hash table for sequence A (the database sequence).
2) Scan sequence B for hits.
3) Extend hits.

Also O(n) time.

Step 1: Build the hash table for sequence A. (3-tuple example)
For DNA sequences:
Seq. A = AGATCGAT
12345678
AAA
AAC
AGA   1
ATC   3
CGA   5
GAT   2   6
TCG   4
TTT

For protein sequences:
Seq. A = ELVIS
Add xyz to the hash table if Score(xyz, ELV) \geq T;
Add xyz to the hash table if Score(xyz, LVIS) \geq T;
Add xyz to the hash table if Score(xyz, VIS) \geq T;

Step 2: Scan sequence B for hits.

Step 3: Extend hits.

Terminate if the score of the extension fades away.

BLAST 2.0 saves the time spent in extension, and considers gapped alignments.
Remarks

• Filtering is based on the observation that a good alignment usually includes short identical or very similar fragments.
• The idea of filtration was used in both FASTA and BLAST to achieve high speed.

Linear space ideas
Hirschberg, 1975; Myers and Miller, 1988
(i) scores can be computed in (O(n)) space
(ii) divide-and-conquer

\[
S(a_1 \ldots a_{m/2}, b_1 \ldots b_j) + S(a_{m/2+1} \ldots a_m, b_{j+1} \ldots b_n)
\]
maximized

Two subproblems
½ original problem size

Four subproblems
¼ original problem size
Time and Space Complexity

- Space: $O(m + n)$
- Time: $O(mn)(1 + \frac{1}{2} + \frac{1}{4} + \ldots) = O(mn)$

Band Alignment

(K. Chao, W. Pearson, and W. Miller)

Sequence A

Sequence B

Band Alignment in Linear Space

The remaining subproblems are no longer only half of the original problem. In worst case, this could cause an additional $\log n$ factor in time.
Multiple sequence alignment (MSA)

- The multiple sequence alignment problem is to simultaneously align more than two sequences.

Seq1: GCTC  GC-TC
Seq2: AC    A---C
Seq3: GATC  G-ATC

How to score an MSA?

- *Sum-of-Pairs (SP-score)*

\[
\text{Score} = \left\{ \begin{array}{c}
\text{Score}^{(GC-TC)} + \\
\text{Score}^{(A---C)} + \\
\text{Score}^{(G-ATC)}
\end{array} \right. 
\]

MSA for three sequences

- an \(O(n^3)\) algorithm

General MSA

- For \(k\) sequences of length \(n\): \(O(n^4)\)
- NP-Complete (Wang and Jiang)
- The exact multiple alignment algorithms for many sequences are not feasible.
- Some approximation algorithms are given.
  \((e.g., 2^{-l/k} \text{ for any fixed } l \text{ by Bafna et al.})\)
Progressive alignment
• A heuristic approach proposed by Feng and Doolittle.
• It iteratively merges the most similar pairs.
• “Once a gap, always a gap”

The time for progressive alignment in most cases is roughly the order of the time for computing all pairwise alignment, i.e., $O(k^2n^2)$.

Concluding remarks
• Three essential components of the dynamic programming approach:
  – the recurrence relation
  – the tabular computation
  – the backtracing
• The dynamic-programming approach has been used in a vast number of computational problems in bioinformatics.