More Alignments
Myoglobin Genes of Mouse and Human

>NM_013593.3 Mus musculus myoglobin (Mb), transcript variant 2, mRNA
TCGGGAACTGTTTTAGAAACAGAAGACATCATCTTCAACATCCAGAGGACTGTCATCCTTGTCCCTGTGGGTGAGGGAAACAAACACTTGGCTTCAATGTCCCAGGAGAAAGACCCAATTGCTCATCCAGCCCACGTGGCCTCCAGAAGCCACCATGGGGCTCAGTGATGGGGAGTGGCAGCTGGTGCTGAATGTCTGGGGGAAGGTGGAGGCACCTTGCTGGCCATGGACAGGAAGTCCTCATCGGTCTGTTTAAGACTCACCCTGAGACCCTGGATAAGTTTGACAAGTTCAAGAACTTGAAGTCAGAGGAAGATATGAAGGGCTCAGAGGACCTGAAGAAGCATGGTTGCACCGTGCTCACAGCCCTGGGTACCATCCTGAAGAAGAAGGGACAACATGCTGCCGAGATCCAGCCTCAGCCCAATCACAGGCAACCAAGCCACCAAGCACAAGATCCCGGTCAAGTACCTGGAGTTTATCTCAGAAATTATCATGGAAGTCCTGAAGAAGAGACATTCCGGGGACTTTGGAGCAGATGCTCAGGGCGCCATGAGCAAGGCCCTGGAGCTCTTCCGGAATGACATTGCCGCCAAGTACAAGGAGCTAGGCTTCCAGGGCTGAGCCATGGGCTCCCACTGTCCAGCCCACCAAGCTGGGACCCAGTGTTGTGTAGCAAGTAGCGTGTGCAGTGTTCTAGGTTAGCAAGAACAGAAGAGGGGAGCATAGTGTGGCATCCACCCACACCCCTGGGGACAGGGCTCTGGGCAGTGTTACCCTGGAGCCCAGAGGTGCAAAGTGGCCTTCGTCAGCTCTGCCGGGTCATGCTCAGGTCTCCTAAGTCCCAGTCCATTTTCTTCTGGTTTTGGGAAAATCTCTTTTCCACTGTCACATTTGACCCCAAATCCAAGTCACTGACTAGCAGACCCTGACCTTTGGGCGAGATGGAGGGTTGCTTAGAGGGAGTGGAGGGTGAAAACGGGGCGGTGAGCATCAAGTCTCCCACTGCTCAGCTTCCCGTTGACCCACCTTGTCTCAATAAAATATCCTGCGAGTCCTCAAAAAAAAAAAAA

>NM_005368.3 Homo sapiens myoglobin (MB), transcript variant 1, mRNA
AAACCCCAAGCTGTGTGGGGGCGACAGGCAACCCACAGTGGACCACTCTGTCTTTTTGTCTTCTTCTTCATGCGCCCATGGGGCTCAGCGACGGGGAATGGCAGTTGGTGCTGAACGTCTGGGGGAAGGTGGAGGCCTGACATCCCAGGCCATGGGCAGGAAGTCCTCATCAGGCTCTTTAAGGGTCACCCAGAGACTCTGGAGAAGTTTGACAAGTTCAAGCACCTGAAGTCAGAGGACGAGATGAAGGCGTCTGAGGACTTAAAGAAGCATGGTGCCACCGTGCTCACCGCCCTGGGTGGCATCCTTAAGAAGAAGGGGCATCATGAGGCAGAGATTAAGCCCCTGGCACAGTCGCATGCCACCAAGCACAAGATCCCCGTGAAGTACCTGGAGTTCATCTCGGAATGCATCATCCAGGTTCTGCGAGCAGCAAGCATCCCGGGGACTTTGGTGCTGATGCCCAGGGGGCCATGAACAAGGCCCTGGAGCTGTTCCGGAAGGACATGGCCTCCAACTACAAGGAGCTGGGCTTCCAGGGCTAGGCCCCTGCCGCTCCCACCCCCACCCATCTGGGCCCCGGGGTTCAAGAGAGAGAGGGGTCGTGATCTCTGGTCTGAGCCATATAGAAGTTTGTCTCTGAGTGCTCCTGTGGTGTCGAAGAGAGGAGGGTCGCTGAAGGTAAAAAGCATGGTGCTTCTCCCGTCCGGTGCATGCTCAGGTCTCCTAAGTCCCAGTCCATTTTCTTCTGGTTTTGGGAAAATCTCTTTTCCACTGTCACATTTGACCCCAAATCCAAGTCACTGACTAGCAGACCCTGACCTTTGGGCGAGATGGAGGGTTGCTTAGAGGGAGTGGAGGGTGAAAACGGGGCGGTGAGCATCAAGTCTCCCACTGCTCAGCTTCCCGTTGACCCACCTTGTCTCAATAAAATATCCTGCGAGTCCTCAAAAAAAAAAAAA

Try align them at EMBL-EBI
Mouse v.s. Human

- Human and mouse share big blocks on their genomes.
- Figure shows relation between chromosome X of mouse and human.
- Each colored block is relatively conserved, but different in orders and orientations.
- Seven inversions are required to put them in the correct order and orientation. This is called “sorting by reversals”.

![Diagram showing the relation between mouse and human chromosome X](image)
Mouse, Human, Chimpanzee

Mouse to Human

Chimpanzee to Human
Local Alignment

- Conserved regions are “local” to the genome/chromosome. But previous alignment is “global”.
- We need a model to define “local” similarity.
Local Alignment

- Given: two sequences S and T
- Find: substrings of S and T that maximizes the alignment score.

- AATTAG—CCGATGAC
- || | |||
- TGGAGGCTGATATA

- I.e., The indels at the beginning and end of the two strings are free.
Local Alignment

- Local alignment score is at least 0.
- The model only makes sense for alignment but not edit distance nor LCS.
- Question: Is the optimal local alignment a local part of an optimal “global” alignment?
Warm-up: Prefix alignment

What if we want to find the highest-scoring alignment between two prefixes of the two sequences.

- CATTC
- ATTGA

Match=1
Mismatch=-1
Indel=-1
Warm-up: “suffix alignment”

• Suppose we only get the “free” gap at the prefixes of the alignment.
  • AATTAG--CCGAT
  • || || ||
  • TGGAGGCTGAT

• That is, we choose two suffixes, and align them together optimally.
Last column

- Let $D[i,j]$ denote the optimal “suffix alignment” alignment score of $s[1..i], t[1..j]$.
- Consider the last column of this optimal “suffix” alignment. Four cases arise:

  Case 1: $s[i]$ v.s. $t[j]$
  Case 2: $s[i]$ v.s. $-$
  Case 3: $t[j]$ v.s. $-$
  Case 4: an empty alignment

- Case 4 is the only new case comparing to the basic alignment.
DP algorithm for “suffix alignment”

\[
D[i,j] = \max \begin{cases} 
D[i-1, j-1] + f(s[i], t[j]); \\
D[i-1, j] + f(s[i],-); \\
D[i, j-1] + f(-, s[j]); \\
0 
\end{cases}
\]
## Suffix Alignment Example

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

**Match** = 1  
**Mismatch** = -1  
**Indel** = -1
Local Alignment

• The optimal local alignment is $\max_{i,j} D[i,j]$
  • Suppose the optimal local alignment aligns $S[i'..i]$ and $T[j'..j]$ together, then it is the best “suffix alignment” of $S[1..i]$ and $T[1..j]$.
  • Moreover, any local alignment is a “suffix alignment” of two prefixes, and vice versa.
  • ……
• Filling the DP table is the same as suffix alignment.
• But then find $(i,j)$ to maximize $D[i,j]$, and backtrack from there.
Local Alignment Example

\[
\begin{array}{cccccc}
& C & A & T & T & C \\
C & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 0 & 1 & 0 & 0 \\
T & 0 & 0 & 0 & 2 & 1 \\
T & 0 & 0 & 0 & 1 & 3 \\
G & 0 & 0 & 0 & 0 & 2 \\
A & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

Match=1
Mismatch=-1
Indel=-1
A Little History

• The algorithm was first proposed by Temple Smith and Michael Waterman in 1981. It works for both linear and affined gap penalty.
• It is known popularly as the Smith-Waterman algorithm.
• The global alignment algorithm was called the Needleman-Wunsch algorithm, which was published in 1970.
### Affine Gap Local Alignment

\[ D_0[i,j] = f(s[i], t[j]) + \max \begin{cases} D_0[i-1, j-1]; \\ D_1[i-1, j-1]; \\ D_2[i-1, j-1]; \\ 0 \end{cases} \]

\[ D_2[i,j] = \text{gapext} + \max \begin{cases} D_0[i, j-1] + \text{gapopen}; \\ D_1[i, j-1]; \\ D_2[i, j-1] + \text{gapopen}; \\ 0 \end{cases} \]

\[ D_1[i,j] = \text{gapext} + \max \begin{cases} D_0[i, j-1] + \text{gapopen}; \\ D_1[i, j-1]; \\ D_2[i, j-1] + \text{gapopen}; \\ 0 \end{cases} \]

- Algorithm is as before, except that score is now lower bounded by 0.
- Afterward, find maximum element in all 3 tables, and backtrack until reaching a 0.
An important side note

It’s fairly straightforward to not just find one local alignment of S and T, but many of them.

How?

Time complexity?

This is crucially important when several sub-regions of S and T are evolutionarily conserved.
Local Alignment

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>C</th>
<th>C</th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</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>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Fit Alignment

• Given sequence S and T. Find a global alignment between S and a substring of T, maximizing the alignment score.

\[
\begin{array}{c}
\text{S} \\
\text{T}
\end{array}
\]

• Deleting the prefix of T is free, deleting the suffix of T is free.
• How?
• Time Complexity?
Linear Space Alignment

• Why linear space?
  • Computer RAM used to be very expensive in 80s.
  • “Prediction: The cost for 128 kilobytes of memory will fall below U$100 in the near future.”
  • Even today, keeping everything in the L2 cache or within one page of the RAM will speed up the computation.

• We have learned the linear space if only alignment score, instead of the alignment, is required.

• Let’s now develop a linear space alignment. We focus on global alignment model first.

• We will learn how to get slightly more than the alignment score…
Linear Space (global) Alignment

• We want to compute one piece of information
  • the j such that S[1..m/2] aligns with T[1..j] in the optimal alignment.

\[
\begin{align*}
  S & \quad m/2 \\
  T & \quad j
\end{align*}
\]

• This is only slightly more than the alignment score.
  • Can this be done in linear space?
  • So what?
Essentially, $j$ is such that

$$\text{score}(S[1..m/2], T[1..j]) + \text{score}(S[m/2+1..m], T[j+1,n])$$

is maximized.

We can reverse $S$ and $T$ so that the second part of the formula is also an alignment of prefixes.
Divide

\[ \text{score}(S[1..m/2], T[1..j]) + \]
\[ \text{score}(S[m/2+1..m], T[j+1..n]) \]
Algorithm

• Use the previous idea to compute j such that the optimal alignment can be divided into two parts:
  • S[1..m/2] v.s. T[1..j]
  • S[m/2+1..m] v.s. T[j+1..n]

• Then we use the same algorithm to recursively compute the optimal alignments of these two parts.

• Return the concatenation of these two optimal alignments.
Time Complexity

- $T(m,n) \leq mn + T(m/2,j) + T(m/2,n-j) \leq 2mn$

A Linear Space Algorithm for Computing Maximal Common Subsequences

D.S. Hirschberg
Princeton University

The problem of finding a longest common subsequence of two strings has been solved in quadratic time and space. An algorithm is presented which will solve this problem in quadratic time and in linear space.

Key Words and Phrases: subsequence, longest common subsequence, string correction, editing
CR Categories: 3.63, 3.73, 3.79, 4.22, 5.25
Linear Space Affine Gap Penalty

CABIOS

Optimal alignments in linear space

Eugene W. Myers¹,² and Webb Miller²

Abstract

Space, not time, is often the limiting factor when computing optimal sequence alignments, and a number of recent papers in the biology literature have proposed space-saving strategies. However, a 1975 computer science paper by Hirschberg presented a method that is superior to the new proposals, both in theory and in practice. The goal of this paper is to give Hirschberg’s idea the visibility it deserves by developing a linear-space version of Gotoh’s algorithm, which accommodates affine gap penalties. A portable C-software package implementing this algorithm is available on the BIONET free of charge.

where $\sigma_{\text{max}} = \max_{a,b} \sigma(a,b)$ (Smith et al., 1981). Thus, to produce an alignment that maximizes the similarity score, first apply these transformations and then run the program described in this paper with the resulting $w$, $g$, and $h$. If the minimum conversion score is $C$, then the corresponding maximum alignment score is $\frac{1}{2}(M + N)\sigma_{\text{max}} - C$.

Gotoh (1982) gave an algorithm that solves such problems in $O(MN)$ time. If only the minimum cost is desired, then it is easy to implement the algorithm in $O(N)$ space, where $N$ can be taken as the shorter sequence length. If one also desires a set of operations attaining the minimum cost, then straightforward implementations need $O(MN)$ space. In practice, this space

“The goal of this paper is to give Hirschberg’s idea the visibility it deserves by developing a linear-space version of Gotoh’s algorithm.”
Question

How to do local alignment in linear space?
How to do affined gap penalty in linear space?