**COS 597c: Topics Computational Molecular Biology**
Lecture 3: September 27, 1999
Lecturer: Mona Singh
Scribe: Erich Schmidt [1]

# Sequence Comparison: Local and Multiple Alignments

## Recap: Global Alignment Algorithm

In the last lecture, we covered global sequence alignment. Here, we want to align two sequences $s$ and $t$, with $|s| = m$ and $|t| = n$. We use dynamic programming to optimally align all the prefixes of $s$ with all the prefixes of $t$. We use a matrix $sim$ (with $m + 1$ rows and $n + 1$ columns) to keep track of the optimal scores of alignments of prefixes. That is, in the matrix $sim$, entry $sim(i, j)$ is the score of the optimal alignment of the $i$th prefix of $s$ with the $j$th prefix of $t$. For the simple similarity function we considered (match = 1, mismatch =-1, and gap = -2), to calculate $sim(i, j)$, we find the maximum of the three numbers $sim(i-1, j) - 2$, $sim(i, j-1) - 2$, and $sim(i-1, j-1) \pm 1$. We choose $+1$ in the last term if $s(i) = t(i)$; otherwise we choose $-1$. We also record which matrix entries the maximum derived from by using one or more arrows.

To construct an optimal alignment, we examine the entry $sim(m, n)$ and follow a direct path back through the arrows to the origin of the matrix. The direction of the arrows pointing out of $sim(i, j)$ indicate the type of alignment at that position. A diagonal arrow indicates that $s(i)$ is aligned with $t(j)$; a horizontal arrow indicates that $t(j)$ is aligned with a gap; and a vertical arrow indicates that $s(i)$ is aligned with a gap.

This algorithm uses time $O(mn)$ and space $O(mn)$. There is also an improvement which uses $O(m + n)$ space.

## Local Alignments

**Motivation:** Proteins are built of modules that can be mixed and matched. Thus,

[1]Portions of this lecture are adapted from one given by Bonnie Berger at MIT. Scribe notes adapted from notes taken by Antonio Ramirez, also at MIT.

the global similarity may be poor for some sequences, but we may want to look for local regions of similarity that suggest shared structural or functional subunits.

A *local alignment* of strings $s$ and $t$ is an alignment of a substring[2] of $s$ with a substring of $t$.

**Problem:** Given two sequences ($s$ of length $m$ and $t$ of length $n$), find the highest scoring local alignment between them.

We can naively test all possible substrings against each other, yielding an algorithm with a running time of $O((nm)^3)$. Here we present an algorithm that runs in time $O(nm)$.

**Algorithm:**

**Definition:** A *suffix* is a substring containing the last character of a string (includes the empty string). (Example: the suffixes of the string `TAC` are: empty, `C`, `AC`, `TAC`.)

The local alignment algorithm algorithm (or Smith-Waterman algorithm) involves a very simple modification to the global alignment algorithm. We will again use a $(m+1) \times (n+1)$ matrix *sim*. However, this time, the $(i, j)$th entry of the matrix *sim* will hold the optimal alignment between a *suffix* of the $i$th prefix of $s$ and a *suffix* of the $j$th prefix of $t$.

The basic intuition is that a suffix of a prefix is a substring, and $sim(i, j)$ holds the best alignment score between substrings that end in $s(i)$ and $t(j)$. Suppose that we want the best alignment between a substring of $s$ ending in $s(i)$ and a substring of $t$ ending in $t(j)$. This means that we need to pick the best starting points for the strings, since their endpoints are determined. Note that $s(i)$ is not necessarily aligned with $t(j)$.

**Note:** there are no initial or trailing gaps in an optimal local alignment. Since introducing gaps causes the alignment score to drop, we can always delete the initial or trailing columns containing gaps and increase the score of the resulting alignment.

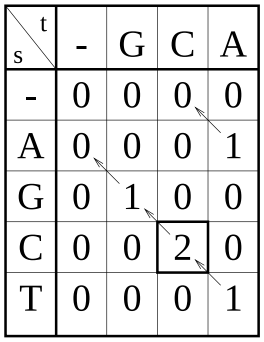
We will fill in the *sim* matrix. A small addition to the global alignment algorithm solves this problem. First we zero out the first row and column of *sim*, because there are no initial gaps in the best local alignment. Then we use the rule:

$$sim(i, j) = \max \begin{cases} sim(i-1, j-1) \pm 1 \\ sim(i-1, j) - 2 \\ sim(i, j-1) - 2 \\ 0 \end{cases} \tag{1}$$

[2]Recall that a substring must consist of contiguous characters.

where the sign for the first case is chosen, as before, according to whether $s(i)$ matches $t(j)$ or not. As we fill in the array, we remember when a zero came from the fourth case. Intuitively, this case says that the best match between suffixes might be to align empty suffixes, which yields a score of zero.

See Figure 1 for an example of the algorithm at work, in a local alignment of strings $s = \texttt{AGCT}$ and $t = \texttt{GCA}$.



| s \ t | - | G | C | A |
|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 |
| A | 0 | 0 | 0 | 1 |
| G | 0 | 1 | 0 | 0 |
| C | 0 | 0 | $\boxed{2}$ | 0 |
| T | 0 | 0 | 0 | 1 |

Figure 1: Local alignment example

To construct an optimal local alignment, we look for the maximum value in the matrix. Then we follow its arrows back until we hit a zero. In Figure 1, we choose the value 2 in entry $sim(3,2)$. This gives the local alignment

$s$: `GC`
$t$: `GC`

If we want further optimal alignments, we take another maximum value in the matrix, generally one that has not already been visited by a previous path. If we want the second best local alignment, we take the second highest value in the matrix that has not already been visited. These alignments are sometimes useful and are called *near-optimal alignments.* In our case, if we start from $sim(1,3)$, we obtain the local alignment.

$s$: `A`
$t$: `A`

This algorithm uses time $O(mn)$ and space $O(mn)$. Again, it is possible to improve it to use space $O(m+n)$.

**Idea for inductive proof.** Suppose we know the best alignment of a substring of $s$ ending in position $i$ with a substring of $t$ ending in position $j$. There are four cases:

- The alignment was gotten by aligning $s(i)$ with $t(j)$. What is left is a suffix ending in $s(i-1)$ and a suffix ending in $t(j-1)$. We already have the best alignment for that through the inductive hypothesis.
- The alignment was gotten by aligning $s(i)$ with a gap. What is left is to align a suffix ending in $s(i-1)$ with a suffix ending in $t(j)$. We already have the best for that through the inductive hypothesis.
- The alignment was gotten by aligning a gap with $t(j)$. This case is analogous to the previous case.
- It was gotten by aligning empty strings (suffixes can be empty!).

The base cases in the induction are the alignments of an empty string with any prefix. That is, $sim(i,0) = sim(0,j) = 0$ for all $i$ and $j$.

## Extensions to the Alignment Problem

Sequence comparisons in practice often use different gap penalties (as well as different substitution matrices). Using different gap penalties can require changes in our basic algorithm.

We often don't penalize opening or trailing gaps in a global alignment; this can be easily done with a slight modification to the Needleman-Wunsch algorithm (change the initial conditions and where to begin the traceback from in the similarity matrix).

Gap penalties are also important when we want to note the difference between $k$ interspersed spaces and $k$ contiguous spaces, as in the example below:

```
xabcaxbxcxabc
x---a-b-c-abc
```

vs.

```
xabcaxbxcxabc
xabc------abc
```

We model favoring the second alignment using *linear (affine) gap penalty*: the gap penalty is now expressed as a function

$$w(k) = p + qk$$

where $p$ is the gap open penalty, $q$ is the gap continuation penalty, and $k$ is the number of consecutive gaps in a row. In practice, the penalty $p$ is made much worse than the penalty $q$. The reason is that a break in a molecule for a deletion or insertion is hard to make. Once a break is made, though, it is relatively easy to make multiple insertions or deletions. This type of gap penalty function is used in computing most pairwise alignments.

With linear gap penalties, we can still find a solution in time $O(mn)$; the idea is to use *three* $(m+1) \times (n+1)$ arrays that keep track of what kind of gap we have and where it comes from (either in $s$ or $t$). See [2] or the Setubal and Meidanis textbook [3] for details.

## Multiple Sequence Alignments (MSA)

So far, we have concentrated on comparisons between a pair of sequences. However, we often are given several sequences that we have to align simultaneously.

**Motivation:** Multiple sequence alignments (or MSAs)

- provide stronger evidence than pairwise similarity for functional and structural inferences.
- are useful for detecting regions of variability or conservation (that might be missed in a pairwise alignment)
- are the first step in other computational procedures:
  - phylogenetic reconstruction
  - secondary structure prediction
  - building profiles (for use in locating other similar sequences)

So, given $k$ strings $s_1, s_2, \ldots s_k$, a *multiple sequence alignment* is obtained by inserting spaces in the sequences to make them the same length.

**Note:** no column can be all spaces.

**Example:** MSA of 4 sequences:

```
s1 : MQPILLLV
s2 : MLR-LL--
s3 : MK-ILLL-
s4 : MPPVLILV
```

**Question:** How to score an alignment?

There are many possibilities; one commonly studied scoring measure is *sum-of-pairs*. It is determined for each column in the alignment, and it is the sum of the scores of all pairs of symbols in the column. In the example above, the score of the fourth column is:

SP(`I`, `-`, `I`, `V`) = score(`I`, `-`) + score(`I`, `I`) + score(`I`, `V`) + score(`-`, `I`) + score(`-`, `V`) + score(`I`, `V`)

**Note:** score(`-`, `-`) = 0; gaps never occur in an entire column, but may occur pairwise.

**SP-MSA Problem:** Find the best MSA under the SP scoring system.

Is sum-of-pairs a good measure for scoring multiple sequence alignments? It is not clear, as sum-of-pairs may lead to aberations when scores from closely related sequences can overrun lower scores from distant but more informative sequences. Nevertheless, it is a widely used and studied scoring system.

**Note:** Optimum SP-MSA alignments may not give an optimal alignment for a given pair. For example,

```
s1 : AT
s2 : A-
s3 : -T
s4 : AT
s5 : AT
```

has a sum-of-pairs score of $-4$. The pairwise alignment of sequences $s_2$ and $s_3$ has score $-4$ in this multiple sequence alignment, but their optimum pairwise alignment would be:

```
s2 : A
s3 : T
```

with a score of $-1$. However, replacing this back into the MSA gives:

$s_1$ : `AT`
$s_2$ : `A-`
$s_3$ : `T-`
$s_4$ : `AT`
$s_5$ : `AT`

with a sum-of-pairs score of $-7$. Thus, it is clearly not the case that the best SP-MSA alignment can be found easily using pairwise alignments.

We will sketch the idea for finding the optimum MSA under the sum-of-pairs criterion.

Let's assume we have $k$ sequences $s_1, s_2 \dots s_k$ each of length $n$. Instead of a 2-dimensional array, we will have a $k$-dimensional array, with $(n+1)^k$ entries. How will we fill out this matrix? Again, let's consider the last column of the alignment and see what happens. For each sequence, either its last character is in the last column of the alignment, or it's not (and we have a gap instead). This gives $2^k - 1$ possibilities, since we can't have all gaps in a column in the alignment.

So to fill out the $(n+1)^k$ entries in the array, we need to take the maximum over $2^k - 1$ possibilities to compute each entry, and for each possibility we need to calculate the sum of the pairwise scores ($O(k^2)$ time, as there are $k(k-1)/2$ pairwise scores). So the total running time is: $O(2^k n^k k^2)$.

The complexity of this algorithm is exponential in the number of input sequences, thus can't be used in practice for more than a few sequences. In fact, SP-MSA has been shown to be NP-complete, and in the next lecture we'll discuss heuristics for multiple sequence alignment that are used in practice.

## References

[1] Smith, T. and Waterman, M. (1981) Identification of common molecular subsequences. *J. Mol. Biol.* 147:195–197.

[2] Gotoh, O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.* 162:705–708.

[3] Setubal, J. and Meidanis, J. *Introduction to Computational Molecular Biology.* PWS Publishing Company, 1997.