

Formal Comparison of Two Strings

Recall that our goal is to compare strings in a way that reveals any evidence of homology, and that highlights important biological similarities between the strings. That is, the comparison should allow us to exploit homology to deduce important biology. As we look at the technical details of comparison models and methods that have been developed in bioinformatics, we need always ask if the models and methods achieve that goal.

0.1 String alignment

Definition A (global) *alignment* of two strings S_1 and S_2 is obtained by first inserting chosen spaces (or dashes), either into or at the ends of S_1 and S_2 and then placing the two resulting strings one above the other so that every character or space in either string is opposite a unique character or a unique space in the other string. If that placement does put two spaces opposite each other, then the resulting alignment is called *valid*, and otherwise is called *invalid*.

In the remainder of these notes, when we talk about an alignment, we will assume we mean a valid alignment, even without explicitly using the term *valid*.

The term “global” emphasizes the fact that for each string, the *entire* string is involved in the alignment. This will be contrasted with local alignment to be discussed later. Notice that our use of the word “alignment” is now much more precise than its use in Parts I and II. There, alignment was used in the colloquial sense to indicate how one string is placed relative to the other, and spaces were not then allowed in either string.

As an example of a global alignment, consider the alignment of the strings *qacdbd* and *qawxb* shown below. In that alignment, character *c* is mismatched with *w*, both the *d*'s and the *x* are opposite spaces, and all other characters match their counterparts in the opposite string.

<i>q</i>	<i>a</i>	<i>c</i>	-	<i>d</i>	<i>b</i>	<i>d</i>
<i>q</i>	<i>a</i>	<i>w</i>	<i>x</i>	-	<i>b</i>	-

1 How many alignments are there?

In preparation for models and methods that specify and select a particular alignment from the set of all possible alignments, it is worth first asking how many alignments there are, as a function of string length. If the number is relatively small (more exactly, grows slowly as a function of string length), then we don't have to be too concerned about computational techniques, and more important, can have more exotic criteria for selecting "biologically good" alignments, because we can do more brute force enumeration of the set of all possible alignments. But if the number of alignments grows rapidly as a function of string length, then we might have to make

some modeling compromises and be more clever about how we compute the alignment we select.

To analyze the number of possible alignments, let us assume both strings S_1 and S_2 are of the same length, denoted n . The analysis will focus on a simpler question first: How many possible alignments are there if we put exactly k spaces into both S_1 and S_2 ? We will answer this question as a function of k .

The analysis of the simpler question goes like this: After putting in k spaces, the resulting elongated strings both have length $n + k$. Consider $n + k$ blank positions (drawn on a piece of paper say). An alignment of S_1 and S_2 containing exactly k spaces can be obtained by first choosing which of the $n + k$ positions will be filled by the n characters from string S_1 . Having made a choice of some particular n positions, we then need to select positions for the character from S_2 . However, since we cannot end up with a space opposite a space, we can only choose to place down the spaces of S_2 in those positions which have original characters of S_1 . That means there are only n possible places where the k spaces for S_2 can be placed. That is, the k spaces in S_2 must be placed opposite k of the n positions where actual characters of S_1 were placed. There are $\binom{n}{k}$ ways to choose where to put those k spaces. After the spaces have been chosen, we place in the n characters of S_2 in the n remaining places. Every possible alignment can be obtained in the above way. Hence there are

$$\binom{n+k}{n} \times \binom{n}{k}$$

ways to choose the positions in this way. Let us call this number $A(n, k)$. Recall that

$$\binom{x}{y} = \frac{x!}{y! \times (x-y)!}.$$

This is also called a *binomial coefficient*, and it is the exact number of ways to choose y distinct items from a set of x distinct items. Recall that $x!$ is the product of all the numbers from 1 up to (and including) x .

So we now have answered the simpler question, and have a formula for $A(n, k)$, the number of alignments when exactly k spaces are inserted into each string of length n . With it, we can write a formula for the total number of alignments of two strings of length n each. Since no two spaces in an alignment can be opposite each other, k is at most n . Therefore, the total number of possible alignments of two strings of length n each is

$$\sum_{k=0}^{k=n} A(n, k).$$

We call this $T(n)$.

Now, we don't know a simpler formula for $T(n)$, but the one we have can be used in a program to compute the values for $T(n)$, given a choice of n . A Perl program, called `countbadly.pl` will be available on the web. I call it `countbadly.pl` because it

isn't a good idea to directly use this formula. Rather, by examining how $A(n, k)$ relates to $A(n, k - 1)$ we see that

$$A(n, k) = A(n, k - 1) \times \frac{(n + k)(n - k + 1)}{k^2}.$$

Using that recursive (or iterative) formula for $A(n, k)$ allows a much better program to compute $T(n)$. It is called `countgoodly.pl` and it will be on the web shortly. Both of these programs run ahead of the Perl we have learned so far, but if you know C, then `countgoodly.pl` should be pretty clear. `countbadly.pl` uses subroutines and the syntax for passing values is a bit odd in Perl. But have a look anyway and we will explain them during the course.

Ok, so we can compute the exact number of alignments, and that computation shows pretty rapid growth. For example $T(4) = 321$, $T(10) = 8,097,453$ and $T(200) = 5.2 \times 10^{151}$. That certainly seems like fast growth, but we would like something more analytical. Here is a crude argument that shows the growth is at least exponential.

Certainly, $T(n) > A(n, n) = \binom{2n}{n} \times 1 > \frac{n^n}{n!}$. Now Stirling's approximation for $n!$ is $\sqrt{2\pi n} \frac{n^n}{e^n}$, where e is about 2.8. Using Stirling's approximation as an equality,

$$T(n) > \frac{n^n}{n!} = \frac{e^n}{\sqrt{2\pi n}}.$$

The numerator e^n shows exponential growth, while the denominator shows slow growth, $\sqrt{2\pi n}$, as a function of n . So $T(n)$ grows at least exponentially with n .

Hence, it will be impractical, except for very small n , to generate all possible alignments. How then will we be able to define and find a particular "good" alignment from the set of so many alignments? We next consider a formal definition of what a "good" alignment is.

Problem for those inclined: Derive the formula for the number of alignments when one string is of length n and the other string is of length m .

2 String similarity via alignment

One way of formalizing the relatedness of two strings is to measure their *similarity* via alignment, as formally defined below. This approach is chosen in most biological applications for technical reasons that should be clear later.

Definition: Let Σ be the alphabet used for strings S_1 and S_2 , and let Σ' be Σ with the added character “_” denoting a space. Then, for any two characters x, y in Σ' , $s(x, y)$ denotes the value (or *score*) obtained by aligning character x against character y .

In class, we started with a somewhat simpler situation where $s(x, y) = 1$ if x and y are the same characters (match), and equal to -1 if they are different (mismatch), or one is a space character. But the notes here are more general.

Definition: For a given alignment \mathcal{A} of S_1 and S_2 , let S'_1 and S'_2 denote the strings after the chosen insertion of spaces, and let l denote the (equal) length of the two strings S'_1 and S'_2 in \mathcal{A} . The *value* of alignment \mathcal{A} is defined as $\sum_{i=1}^l s(S'_1(i), S'_2(i))$.

That is, every position i in \mathcal{A} specifies a pair of opposing characters in the alphabet Σ' , and the value of \mathcal{A} is obtained by summing the value contributed by each pair.

For example let $\Sigma = \{a, b, c, d\}$ and let the pairwise scores be defined in the following matrix:

s	a	b	c	d	$-$
a	1	-1	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
$-$					0

Then the alignment

c	a	c	$-$	d	b	d
c	a	b	b	d	b	$-$

has a total value of $0+1-2+0+3+3-1 = 4$.

In string similarity problems, scoring matrices usually set $s(x, y)$ to be greater or equal to zero if characters x, y of Σ' match and less than zero if they mismatch. In class, we just had +1 for each match, -1 for each mismatch (or transformation), and -1 for each character opposite a space (an indel). With such a scoring scheme, one seeks an alignment with as *large* a value as possible. That alignment will emphasize matches (or similarities) between the two strings while penalizing mismatches or inserted spaces. Of course, the meaningfulness of the resulting alignment may depend heavily on the scoring scheme used and how match scores compare to mismatch and space scores. There are numerous character-pair scoring matrices that have been suggested for proteins and for DNA [3, 2, 6, 4, 5, 1], and no single scheme is right for all applications.

Definition: Given a pairwise scoring matrix over the alphabet Σ' , the *similarity* of two strings S_1, S_2 is defined as the value of the alignment \mathcal{A} of S_1 and S_2 which *maximizes* total alignment value. This is also called the *optimal alignment value* of S_1 and S_2 .

References

- [1] D. L. Brutlag, J. P. Dautricourt, S. Maulik, and J. Relph. Improved sensitivity of biological sequence database searches. *Comp. Apps. in the BioSciences*, 6:237–245, 1990.

- [2] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:345–352, 1978.
- [3] R. F. Doolittle. *Of Urfs and Orfs: A primer on how to analyze derived amino acid sequences*. University Science Books, Mill Valley, CA., 1986.
- [4] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. of the Nat. Academy of Science*, 89:10915–10919, 1992.
- [5] T.H. Jukes and C. R. Cantor. Evolution of protein molecules. In H.N. Munro, editor, *Mammalian Protein Metabolism*, pages 21–132. Academic Press, 1969.
- [6] R. Schwarz and M. Dayhoff. Matrices for detecting distant relationships. In M. Dayhoff, editor, *Atlas of protein sequences*, pages 353–358. National Biomedical Research Foundation, 1979.