

1 Local alignment: Finding substrings of high similarity

In many applications, two strings may not be highly similar in their entirety but contain *regions* that are highly similar. The task is to find and extract a pair of regions, one from each of the two given strings, which exhibit high similarity. This is called the *local alignment or local similarity* problem and is defined formally below.

Local alignment problem Given two strings S_1 and S_2 , find substrings α and β of S_1 and S_2 , respectively, whose similarity (optimal global alignment value), is maximum over all pairs of substrings from S_1 and S_2 . We use v^* to denote the value of an optimal solution to the local alignment problem.

For example, consider the strings $S_1 = pqraxabcstvq$ and $S_2 = xyabacsll$. If we give each match a value of 2, each mismatch a value of -2 , and each space a value of -1 , then the two substrings $\alpha = axabc$ and $\beta = abacs$ of S_1 and S_2 respectively have the following optimal (global) alignment

$$\begin{array}{cccccccc} a & x & a & b & - & c & s \\ a & x & - & b & a & c & s \end{array}$$

which has a value of 8. Further, over all choices of pairs of substrings, one from each of the two strings, those two substrings have maximum similarity (for the chosen scoring scheme). Hence for that scoring scheme, the optimal local alignment of S_1 and S_2 has value 8 and is defined by substrings $axabc$ and $abacs$.

It should be clear why local alignment is defined in terms of similarity, which *maximizes* an objective function, rather than in terms of edit distance, which *minimizes* an objective. When one seeks a pair of substrings to minimize distance, the optimal pairs would be exactly matching substrings under most natural scoring schemes. But the matching substrings might be just a single character long and would not identify a *region* of high similarity. A formulation such as local alignment, where matches contribute positively and mismatches and spaces contribute negatively, is more likely to find more meaningful regions of high similarity.

Why local alignment?

Global alignment of protein sequences is often meaningful when the two strings are members of the same protein family. For example, the protein *cytochrome c* has almost the same length in most organisms that produce it, and one expects to see a relationship between two cytochromes from any different species over the entire length of the two strings. The same is true of proteins in the *globin* family, such as

myoglobin and *hemoglobin*. In these cases, global alignment is meaningful. When trying to deduce evolutionary history by examining protein sequence similarities and differences, one usually compares proteins in the same sequence family, and so global alignment is typically meaningful and effective in those applications.

However, in many biological applications, local similarity (local alignment) is far more meaningful than global similarity (global alignment). This is particularly true when long stretches of anonymous DNA are compared, since only *some* internal sections of those strings may be related. When comparing protein sequences, local alignment is also critical because proteins from very different families are often made up of the same structural or functional subunits (called motifs or domains), and local alignment is appropriate in searching for these (unknown) subunits. Similarly, different proteins are often made from related motifs that form the inner core of the protein, but the motifs are separated by outside surface looping regions that can be quite different in different proteins.

A very interesting example of conserved domains comes from the proteins encoded by *homeobox* genes. Homeobox genes [?, ?] show up in a wide variety of species, from fruit flies to frogs to humans. These genes regulate high-level embryonic development, and a single mutation in these genes can transform one body part into another (one of the original mutation experiments causes fruit fly antenna to develop as legs, which doesn't seem to bother the fly very much). The protein sequences that these genes encode are very different in each species, except in one region called the *homeodomain*. The homeodomain consists of about sixty amino acids that form the part of the regulatory protein that binds to DNA. Oddly, homeodomains made by certain insect and mammalian genes are particularly similar, showing about 50 to 95 percent identity in alignments without spaces. Protein-to-DNA binding is central in how those proteins regulate embryo development and cell differentiation. So the amino acid sequence in the most biologically critical part of those proteins is highly conserved, while the other parts of the protein sequences show very little similarity. In cases such as these, local alignment is certainly a more appropriate way to compare protein sequences than is global alignment.

Local alignment in protein is additionally important because particular *isolated* characters of related proteins may be more highly conserved than the rest of the protein (for example, the amino acids at the *active site* of an enzyme or the amino acids in the *hydrophobic core* of a globular protein are the most highly conserved). Local alignment will more likely detect these conserved characters than will global alignment. A good example is the family of *serine proteases* where a few isolated, conserved amino acids characterize the family. Another example comes from the Helix-Turn-Helix motif which occurs frequently in proteins that regulate DNA transcription by binding to DNA. The tenth position of the Helix-Turn-Helix motif is very frequently occupied by the amino acid glycine, but the rest of the motif is more variable.

The following quote from C. Chothia [?] further emphasizes the biological impor-

tance of protein domains and hence of *local* string comparison.

Extant proteins have been produced from the original set not just by point mutations, insertions and deletions but also by combinations of genes to give chimeric proteins. This is particularly true of the very large proteins produced in the recent stages of evolution. Many of these are built of different combinations of protein domains that have been selected from a relatively small repertoire.

Doolittle [?] summarizes the point: “The underlying message is that one must be alert to regions of similarity even when they occur embedded in an overall background of dissimilarity”.

So, the dominant viewpoint today is that local alignment is the most appropriate type of alignment for comparing proteins from different protein families. While agreeing with this view viewpoint, it has also been pointed out [?, ?] that one often sees extensive global similarity in pairs of protein strings that are first recognized as being related by strong local similarity. There are also suggestions [?] that in some situations global alignment is more effective than local alignment in exposing important biological commonalities.

1.1 Computing local alignment

Why not look for regions of high similarity in two strings by first globally aligning those strings? A global alignment between two long strings will certainly be *influenced* by regions of high similarity, and an optimal global alignment might well align those corresponding regions with each other. But more often, local regions of high local similarity would get lost in the overall optimal global alignment. So to identify high local similarity it is more effective to search *explicitly* for local similarity.

We will show that if the lengths of strings S_1 and S_2 are n and m respectively, then the local alignment problem can be solved in $O(nm)$ time, the same time as for global alignment. This efficiency is surprising because there are $\Theta(n^2m^2)$ pairs of substrings, so even if a global alignment could be computed in *constant* time for each chosen pair, the time bound would be $\Theta(n^2m^2)$. In fact, if we naively use $O(kl)$ for the bound on the time to align strings of lengths k and l , then the resulting time bound for the local alignment problem would be $O(n^3m^3)$, instead of the $O(nm)$ bound that we will establish. The $O(nm)$ time bound was obtained by Temple Smith and Michael Waterman [?], using the algorithm we will describe below.

In the definition of local alignment given earlier, any scoring scheme was permitted for the global alignment of two chosen substrings. One slight restriction will help in computing local alignment. We assume that the global alignment of two *empty* strings has value zero. That assumption is used in order to allow the local alignment algorithm to choose two empty substrings for α and β . Before describing the solution

to the local alignment problem, it will be helpful to consider first a more restricted version of the problem.

Definition Given a pair of indices $i \leq n$ and $j \leq m$, the *local suffix alignment* problem is to find a (possibly empty) *suffix* α of $S_1[1..i]$ and a (possibly empty) *suffix* β of $S_2[1..j]$ such that $V(\alpha, \beta)$ is the maximum over all pairs of suffixes of $S_1[1..i]$ and $S_2[1..j]$. We use $v(i, j)$ to denote the value of the optimal local suffix alignment for the given index pair i, j .

For example, suppose the objective function counts 2 for each match and -1 for each mismatch or space. If $S_1 = abcxdex$ and $S_2 = xxxcde$, then $v(3, 4) = 2$ (the two c 's match); $v(4, 5) = 1$ (cx aligns with cd); $v(5, 5) = 3$ (x_d aligns with xcd); and $v(6, 6) = 5$ (x_de aligns with $xcde$).

Since the definition allows either or both of the suffixes to be empty, $v(i, j)$ is always greater or equal to zero.

The following theorem shows the relationship between the local alignment problem and the local suffix alignment problem. Recall that v^* is the value of the optimal local alignment for two strings of length n and m .

Theorem 1.1 $v^* = \max[v(i, j) : i \leq n, j \leq m]$.

Proof Certainly $v^* \geq \max[v(i, j) : i \leq n, j \leq m]$ because the optimal solution to the local suffix alignment problem for any i, j is a feasible solution to the local alignment problem. Conversely, let α, β be the substrings in an optimal solution to the local alignment problem and suppose α ends at position i^* and β ends at j^* . Then α, β also defines a local suffix alignment for index pair i^*, j^* , and so $v^* \leq v(i^*, j^*) \leq \max[v(i, j) : i \leq n, j \leq m]$, and both directions of the lemma are established. \square

Theorem 1.1 only specifies the value v^* , but its proof makes clear how to find substrings whose alignment have that value. In particular,

Theorem 1.2 *If i' and j' is an index pair maximizing $v(i, j)$ over all i, j pairs, then a pair of substrings solving the local suffix alignment problem for i', j' also solves the local alignment problem.*

So a solution to the local suffix alignment problem solves the local alignment problem. We now turn our attention to the problem of finding $\max[v(i, j) : i \leq n, j \leq m]$ and a pair of strings whose alignment has maximum value.

1.2 How to solve the local suffix alignment problem

First, $v(i, 0) = 0$ and $v(0, j) = 0$ for all i, j , since we can always choose an empty suffix.

Theorem 1.3 *For $i > 0$ and $j > 0$, the proper recurrence for $v(i, j)$ is*

$$v(i, j) = \max[0, v(i-1, j-1)+s(S_1(i), S_2(j)), v(i-1, j)+s(S_1(i), -), v(i, j-1)+s(-, S_2(j))].$$

Proof The argument is similar to the justifications of previous recurrence relations. Let α and β be the substrings of S_1 and S_2 whose global alignment establishes the optimal local alignment. Since α and β are permitted to be empty suffixes of $S_1[1..i]$ and $S_2[1..j]$, it is correct to include 0 as a *candidate* value for $v(i, j)$. However, if the optimal α is not empty, then character $S_1(i)$ must either be aligned with a space or with character $S_2(j)$. Similarly, if the optimal β is not empty then $S_2(j)$ is aligned with a space or with $S_1(i)$. So we justify the recurrence based on the way characters $S_1(i)$ and $S_2(j)$ may be aligned in the optimal local suffix alignment for i, j .

If $S_1(i)$ is aligned with $S_2(j)$ in the optimal local i, j suffix alignment, then those two characters contribute $s(S_1(i), S_2(j))$ to $v(i, j)$ and the remainder of $v(i, j)$ is determined by the local suffix alignment for indices $i - 1, j - 1$. That local suffix alignment must be optimal and so has value $v(i - 1, j - 1)$. Therefore, if $S_1(i)$ and $S_2(j)$ are aligned with each other, $v(i, j) = v(i - 1, j - 1) + s(S_1(i), S_2(j))$.

If $S_1(i)$ is aligned with a space, then by similar reasoning $v(i, j) = v(i - 1, j) + s(S_1(i), -)$, and if $S_2(j)$ is aligned with a space then $v(i, j) = v(i, j - 1) + s(-, S_2(j))$. All cases being exhausted, we have proven that $v(i, j)$ must either be zero or be equal to one of the three other terms in the recurrence.

On the other hand, for each of the four terms in the recurrence, there *is* a way to choose suffixes of $S_1[1..i]$ and $S_2[1..j]$ so that an alignment of those two suffixes has the value given by the associated term. Hence the optimal suffix alignment value is *at least* the maximum of the four terms in the recurrence. Having proved that $v(i, j)$ must be one of the four terms, and that it must be greater or equal to the maximum of the four terms, it follows that $v(i, j)$ must be *equal* to the maximum, proving the theorem. \square

The recurrences for local suffix alignment are almost identical to those for global alignment. The only difference is the inclusion of zero in the case of local suffix alignment. This makes intuitive sense. In both global alignment and local suffix alignment of prefixes $S_1[1..i]$ and $S_2[1..j]$ the end characters of any alignment are specified, but in the case of local suffix alignment, any number of initial characters can be ignored. The zero in the recurrence implements this, acting to “restart” the recurrence.

Given Theorem 1.2, the method to compute v^* is to compute the dynamic programming table for $v(i, j)$, and then find the largest value in *any* cell in the table, say in cell (i^*, j^*) . As usual, pointers are created while filling in the values of the table. After cell (i^*, j^*) is found, the substrings α and β which give the optimal local alignment of S_1 and S_2 are found by tracing back the pointers from cell (i^*, j^*) until an entry (i', j') is reached which has value zero. Then the optimal local alignment substrings are $\alpha = S_1[i'..i^*]$ and $\beta = S_2[j'..j^*]$.

Time analysis

Since it takes only four comparisons and three arithmetic operations per cell to compute $v(i, j)$, the entire table can be filled in in $O(nm)$ time. The search for v^* and the traceback clearly require only $O(nm)$ time as well, so we have established the following desired theorem.

Theorem 1.4 *For two strings S_1 and S_2 of lengths n and m , the local alignment problem can be solved in $O(nm)$ time, the same time as for global alignment.*

Recall that the pointers in the dynamic programming table for edit distance, global alignment and similarity encode *all* the optimal alignments. Similarly, the pointers in the dynamic programming table for local alignment encode the optimal local alignments as follows.

Theorem 1.5 *All optimal local alignments of two strings are represented in the dynamic programming table for $v(i, j)$ and can be found by tracing any pointers back from any cell with value v^* .*

We leave the proof as an exercise.

1.3 Three final comments on local alignment

Terminology for local and global alignment

In the biological literature, global alignment (similarity) is often referred to as a Needleman-Wunsch [?] alignment after the authors who first discussed global similarity. Local alignment is often referred to as a Smith-Waterman [?] alignment after the authors who introduced local alignment. There is however some confusion in the literature between Needleman-Wunsch and Smith-Waterman as *problem statements* and as *solution methods*. The original solution given by Needleman-Wunsch runs in cubic time and is rarely used. Hence “Needleman-Wunsch” usually refers to the global alignment *problem*. The Smith-Waterman method (detailed in Section 1.1) runs in quadratic time and is commonly used, so “Smith-Waterman” often refers to their specific solution as well as to the problem statement. But there are solution methods to the (Smith-Waterman) local alignment problem that differ from the Smith-Waterman solution and yet are sometimes also referred to as “Smith-Waterman”.

Using Smith-Waterman to find several regions of high similarity

Very often in biological applications it is not sufficient to find just a single pair of substrings of input strings S_1 and S_2 with the optimal local alignment. Rather, what

is required is to find all or “many” pairs of substrings that have similarity above some threshold. A specific application of this kind will be discussed in Section ?? and the general problem will be much more deeply in Section ?. Here we simply point out that in practice, the dynamic programming table used to solve the local suffix alignment problem is often used to find additional pairs of substrings with “high” similarity. The key observation is that for any cell (i, j) in the table, one can find a pair of substrings of S_1 and S_2 (by traceback) with similarity (global alignment value) of $v(i, j)$. So an easy way to look for a set of highly similar substrings is to find a set of cells in the table with a value above some set threshold. Not all similar substrings will be identified in this way, but this approach is common in practice.

The need for good scoring schemes

The utility of optimal local alignment is affected by the scoring scheme used. For example, if matches are scored as one, and mismatches and spaces as zero, then the optimal local alignment will be determined by the longest common *subsequence*. Conversely, if mismatches and spaces are given large negative scores, and each match is given a score of one, then the optimal local alignment will be the longest common *substring*. In most cases, neither of these is the local alignment of interest and some care is required to find an application-dependent scoring scheme that yields meaningful local alignments. For local alignment, the entries in the scoring matrix must have an average score that is negative. Otherwise the resulting “local” optimal alignment tends to be a global alignment. Recently, several authors have developed a rather elegant theory of what scoring schemes for local alignment mean in the context of database search, and how they should be derived. We will briefly discuss this theory in Section ??

References