Sequence Alignment

Dannie Durand

Pairwise Sequence Alignment

The goal of pairwise sequence alignment is to establish a correspondence between the elements in a pair of sequences that share a common property, such as common ancestry or a common structural or functional role. In computational biology, the sequences under consideration are typically nucleic acid or amino acid polymers. We will consider three variants of the pairwise sequence alignment problem: global alignment, semi-global alignment, and local alignment.

Global alignment is used to compare sequences in cases where we have reason to believe that the sequences are related along their entire length. If, for example, sequences s and t are two independent sequencing runs of the same PCR product, then they should differ only in positions where there are sequencing errors. In order to find those sequencing errors, we align all of sequence s with all of sequence t. Other applications of global alignment include finding mutations in closely related gene or protein sequences and identification of single nucleotide polymorphisms (SNPs).

Semi-global alignment is a variant of global alignment that allows for gaps at the beginning and/or the end of one of the sequences. Semi-global alignment should be used in cases where we believe that s and t are related along the entire length of the region where they overlap. For example, if s is a prokaryotic gene sequence and t is the spliced mRNA transcript produced when s is expressed, every base in t should correspond to some base in s. Semi-global alignment "jumps" over the 5' untranslated region in s without exacting a penalty, but forces an alignment along the entire length of t.

In contrast, local alignment addresses cases where we only expect to find isolated regions of similarity. One example is alignment of genomic DNA upstream from two co-expressed genes to find conserved regions that may correspond to transcription factor binding sites. Another application is identification of conserved domains[1] in two amino acid sequences that encode proteins that share one or more domains, but are otherwise unrelated.

Prior to introducing algorithms for these pairwise alignment problems, we introduce some notation.

Alphabet:

An *alphabet*, denoted by Σ, is a finite, unordered set of symbols; e.g.,
DNA: $\Sigma_D = \{A, C, G, T\}$

[1] A domain is a peptide sequence that encodes a protein module that will fold into its characteristic shape independent of the surrounding amino acid context and that is found in many different proteins.

RNA: $\Sigma_R = \{A, C, G, U\}$
Amino acids: $\Sigma_{AA} = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$

Sequences or Strings:

A *sequence* or *string*, s, is a finite succession of the symbols in Σ. We say that $s \in \Sigma^*$, where Σ^* is the set of all sequences over alphabet Σ, including the empty sequence, $\varnothing$. For example, $\Sigma_R^* = \{\varnothing, A, C, G, U, AA, AC, AG, AU, CA, CC, CG, CU, \ldots\}$. Given a sequence s of length m, we use $s[1]s[2]\cdots s[m]$ to denote the symbols in s. Sometimes, we will use $s_1 s_2 \cdots s_m$ for convenience.

Subsequences:

A *subsequence* of s is any sequence obtained by removing zero or more symbols from s. The sequences `CATA` and `CTG` are subsequences of `CATTAG`. `AATTCG` is not. A *proper subsequence* is a subsequence obtained by removing one or more symbols from s.

Substrings:

A *substring* of s is a subsequence of s consisting of consecutive symbols in s. Given a sequence, s, of length m, the substring that begins with $s[i]$ and ends with $s[j]$ is denoted $s[i \ldots j]$, $1 \leq i \leq j \leq m$. The sequence `CAT` is a substring of `CATTAG`. `CATA` is not.

A *prefix* of s is denoted $s[1 \ldots j]$, $j \leq m$.

A *suffix* of s is denoted $s[i \ldots m]$, $1 \leq i$.

Global pairwise alignment

Now that we have some notation to work with, we will introduce a formal definition of a global alignment. Next we will consider how to assign a numerical score to an alignment. The score represents an assessment of the quality of the alignment. Finally, we will introduce an efficient algorithm to find the alignment that is optimal with respect to a scoring function.

Let $\Sigma' = \Sigma \cup \{\_\}$ be the alphabet expanded to include a character to represent gaps. Given sequence $s \in \Sigma^*$ of length m and sequence $t \in \Sigma^*$ of length n, $\alpha = \{s', t'\}$ is an alignment of s and t if and only if

- $s', t' \in (\Sigma')^*$,
- $| s' | = | t' | = l$, where $\max(m, n) \leq l \leq m + n$,
- s is the subsequence obtained by removing '_' from s' and t is the subsequence obtained by removing _ from t',

- there is no value of i for which $s'[i] = t'[i] =$ '_'.

Our goal is to find the best global alignment; that is, we seek an alignment, $\alpha^*(s,t)$, that optimizes an objective criterion.

Scoring an alignment

Given sequences s and t and an alignment $\alpha(s,t) = \{s', t'\}$, it is convenient to assign a score to alignment α that quantifies how well α captures the relationship between s and t. This may be a minimization or a maximization criterion.

Distance scoring: An alignment can be scored using a distance-based metric. This is a minimization criterion: a lower distance indicates a better alignment. We define the distance score of an alignment $\alpha_k(s,t) = \{s', t'\}$ to be

$$\begin{aligned} D\big(\alpha_k(s,t)\big) &= D(s', t') \\ &= \sum_{i=1}^{l} d(s'[i], t'[i]), \end{aligned} \tag{1}$$

where $d(x,y)$ is the distance between a pair of symbols x and y and l is the length of the alignment. The optimal alignment is the alignment that minimizes the distance between s and t:

$$\alpha^*(s,t) = \underset{\alpha_k(s,t)}{\operatorname{argmin}} D\big(\alpha_k(s,t)\big).$$

If $d(x,y) = 1$ and $d(x,\_) = 1$, $\forall x, y$, then $D\big(\alpha^*(s,t)\big)$ corresponds to the minimum number of operations required to transform s into t, where the operations are substitution, insertion, and deletion. This is called the *edit distance*. If $d(x,y) > 1$ or $d(x,\_) > 1$ or both, then $D[\alpha^*(s,t)]$ is called the *weighted edit distance*.

The function specifying the distance between pairs of symbols must obey the following properties, for all x, y, and z:

1. $d(x,x) = 0$
2. $d(x,y) > 0$
3. $d(x,\_) > 0$
4. $d(x,y) = d(y,x)$
5. $d(x,z) \leq d(x,y) + d(y,z)$ (triangle inequality)

Several properties of the distance scoring function are worth noting. First, $D(s', t')$ is a metric; that is, it satisfies the triangle inequality.Second, the symmetric property, $d(x,y) = d(y,x)$, implies that there is no directionality in the scoring system. This is because, given a column with symbols x and y in a pairwise alignment, we have no way of knowing whether the ancestor was an x, followed

by an $x \to y$ substitution, or vice versa. It is also possible that the ancestor was neither x nor y and a series of substitutions gave rise to x in one sequence and to y in the other. Similarly, when a symbol, x, in sequence s is aligned with a gap in sequence t (or vice versa), there is no way to know whether x was inserted in s or deleted from t. For this reason, gaps are also called "indels."

Similarity scoring: Alignments can also be scored with similarity measures. These are maximization criteria: a higher score indicates a better alignment. The similarity score of $\alpha_k(s,t) = \{s', t'\}$ is

$$S(\alpha_k(s,t)) = \sum_{i=1}^{l} p(s'[i], t'[i]), \tag{2}$$

where $p(x,y)$ is a score that reflects the similarity of x and y and $p(x,\_)$ is the gap score. The optimal alignment is the alignment that maximizes the similarity between s and t:

$$\alpha^*(s,t) = \underset{\alpha_k(s,t)}{\operatorname{argmax}} S\big(\alpha_k(s,t)\big).$$

In general, amino acid alignments are scored with substitution matrices that assign a different similarity score to each pair of amino acid residues. Typically, pairs of amino acids with similar properties have higher scores than pairs with divergent properties. Examples of substitution matrices used to score alignments include the PAM and BLOSUM matrices. We will discuss how such substitution matrices are derived later in the semester. For now, we will consider a simple similarity scoring function that treats all symbols in Σ equally. This simple scoring function has just three values; a score for matching symbols (M), a score for a mismatch (m), and a gap score (g):

$$\begin{aligned} p(x,x) &= M, \\ p(x,y) &= m, \\ p(x,\_) &= g. \end{aligned} \tag{3}$$

In order to obtain alignments that make sense, we must impose several constraints on the values of M, m, and g. First, we require that $M > m$, since matches are preferred over mismatches. We further require that a substitution be preferred over two gaps (i.e., $m > 2g$). A scoring function with $m < 2g$ would exclude the possibility of an alignment with substitutions, because the score of any substitution could be improved by replacing it with two gaps.

Note that for both distance and similarity scoring, the score of an alignment is defined to be the sum of the scores for the individual positions in the alignment (Equations 1 and 2), which implies that each position in the alignment is independent of neighboring positions. This assumption is unrealistic: In real biomolecular sequences, there can be interactions between neighboring, or even distant, residues in the sequence. However, scoring functions that assume positional independence are widely used because they greatly simplify the calculation of alignment scores.

A dynamic programming algorithm to align a pair of sequences

We now have a formal definition of an alignment and a way of assigning a numerical score to any given alignment. How do we find the alignment with the optimal score? We could generate all possible alignments, score each one, and choose the alignment with the best score. However, the computational cost would be prohibitive, since the size of the space of all possible alignments of s and t is $O(2^{m+n})$. (Convince yourself this is the case.) Instead, dynamic programming can be used to find the optimal alignment efficiently.

The dynamic programs for sequence alignment compute a matrix a, where $a[i, j]$ is the score of the optimal alignment of the prefixes $s[1..i]$ and $t[1..j]$, that is, the prefixes of s and t that end at positions i and j, respectively. Dynamic programming algorithms for sequence alignment have four components:

- Initialization of the first row and column of $a[i, j]$.
- A recurrence relation for $a[i, j], i > 1, j > 1$.
- Determination of the score of the optimal alignment from the matrix $a[i, j]$.
- Trace back through the matrix to obtain the optimal alignment.

The details of each of these steps are what differentiate global, semi-global, and local alignment. In all cases, the algorithm proceeds by iteratively calculating the values of $a[i, j]$ from the values of $a[i-1, j], a[i, j-1]$, and $a[i-1, j-1]$. At each step, the indices of the entry in a that optimize the right hand side of the recurrence (Equations 4 and 5) are stored in a matrix, $\mathcal{T}$. These pointers are used to reconstruct the alignment that gave the optimal score. The algorithm continues until all entries in the matrix a have been assigned values.

The details of dynamic programming for global alignment are given below for both distance and similarity scoring functions. With distance scoring, all entries in a are non-negative, since $d(x, y) \geq 0, \forall\, x, y$. With a similarity scoring function, the entries in a may be positive or negative. This algorithm computes the scores of all pairs of prefixes in $O(m \cdot n)$ time. The trace back through the alignment matrix to obtain the optimal alignment requires $O(m + n)$ time. Note that there may be more than one optimal alignment.

Global alignment with distance scoring

Input:

Sequences s and t of lengths m and n, respectively.

Initialization:

$$\begin{aligned} a[0,j] &= a[0,j-1] + d(\_, t[j]) \qquad \text{(top row)} \\ a[i,0] &= a[i-1,0] + d(s[i], \_) \qquad \text{(left column)} \end{aligned}$$

Recurrence:

$$a[i,j] = \min \begin{cases} a[i-1,j] + d(s[i], \_) \\ a[i-1,j-1] + d(s[i], t[j]) \\ a[i,j-1] + d(\_, t[j]) \end{cases} \tag{4}$$

Store the indices of the entry in a that minimize the right hand side of Equation 4 in a traceback matrix, $\mathcal{T}$.

Trace back:

Follow the pointers from $\mathcal{T}[m,n]$ to $\mathcal{T}[0,0]$ to obtain the optimal alignment.

Output:

The optimal alignment score, $a[m,n]$.

The optimal global alignment of s and t with respect to distance function, D.

Global alignment with similarity scoring

The dynamic programming algorithm for global alignment with similarity scoring has the same general structure as the global alignment algorithm for distance scoring. However, the details of the initialization and recurrence differ.

Initialization:

$$\begin{aligned} a[0,i] &= a[i-1,0] + g \\ a[j,0] &= a[0,j-1] + g \end{aligned}$$

Recurrence relation:

$$a[i,j] = \max \begin{cases} a[i,j-1] + g \\ a[i-1,j-1] + p(i,j) \\ a[i-1,j] + g \end{cases} \tag{5}$$

Store the indices of the entry in a that minimize the right hand side of Equation 5 in a traceback matrix, $\mathcal{T}$.

Traceback:

From $\mathcal{T}[m, n]$ to $\mathcal{T}[0, 0]$ to obtain the optimal alignment.

Output:

The optimal alignment score, $a[m, n]$.

The optimal global alignment of s and t with respect to distance function, D.

Semi-global alignment

Global alignment seeks the best, full length alignment; that is, the best way to match up two sequences along their entire length. For some applications, it is desirable to relax this requirement and not penalize gaps at the beginning and/or end of an alignment. For example, for sequence assembly, we seek sequence fragments that overlap, that is we expect to be able to align the end of one fragment with the beginning of another. Very occasionally, we may find sequence fragments that start and end at the same position, but, in general, we expect some gaps at the beginning and end of the alignment. Another example is aligning cDNA's with genomic DNA to identify gene structure. The cDNA will be completely covered by the genomic DNA, so we expect gaps at both the beginning and end of the cDNA fragment.

Semi-global alignment is a modification of global alignment that allows the user to specify that gaps will be penalty-free at the beginning of one of the sequences and/or at the end of one of the sequences. Given sequences s and t, there are eight possible cases to consider:

- Gaps are penalty-free at the beginning of s; *e.g.*,

```
s:   _ _ D O
t:   R E D O
```

- Gaps are penalty-free at the beginning of t; *e.g.*,

```
s:   R E D O
t:   _ _ D O
```

- Gaps are penalty-free at the end of s; *e.g.*,

```
s:   D O _ _
t:   D O N E
```

- Gaps are penalty-free at the end of t; *e.g.*,

```
s:   D O N E
t:   D O _ _
```

- Gaps are penalty-free at the beginning and end of s; *e.g.*,

```
s:   _ _ D O _ _
t:   R E D O N E
```

- Gaps are penalty-free at the beginning and end of t; *e.g.*,

```
s:   R E D O N E
t:   _ _ D O _ _
```

- Gaps are penalty-free at the beginning of s and at the end of t; *e.g.*,

```
s:   _ _ D O N E
t:   R E D O _ _
```

- Gaps are penalty-free at the beginning of t and at the end of s; *e.g.*,

```
s:   R E D O _ _
t:   _ _ D O N E
```

In semi-global alignment, we do not allow gaps at the beginning of s and the beginning of t in the same alignment. Nor do we not allow gaps at the end of s and the end of t. Why not?

Like global alignment, the optimal semi-global alignment can be found using dynamic programming using either distance or similarity scoring. Below, we describe the modifications required to adapt the dynamic program for global pairwise alignment with similarity scoring to the semi-global alignment problem. Similar modifications can be made to obtain a semi-global alignment algorithm that uses distances.

Initialization:

To allow gaps at the beginning of s, set $a[0, j] = 0, \forall_j$; i.e., the first row is zero. The first column is initialized as in global alignment.

To allow gaps at the beginning of t, set $a[i, 0] = 0, \forall_i$; i.e., the first column is zero. The first row is initialized as in global alignment.

Recurrence relation:

Same as global.

Optimal alignment score:

To avoid trailing gap penalties at the end of t, we define the optimal score to be $\max_i a[i, n]$, the optimal score in the last column.

To avoid trailing gap penalties at the end of s, we define the optimal score to be $\max_j a[m, j]$, the optimal score in the bottom row.

Trace back:

To avoid trailing gap penalties at the end of t, trace back from $\mathcal{T}[i^*, n]$, where $i^* = \text{argmax}_i\, a[i, n]$. In other words, trace back from the cell(s) in the last column with optimal score.

To avoid trailing gap penalties at the end of s, trace back from $\mathcal{T}[m, j^*]$, where $j^* = \text{argmax}_j\, a[m, j]$. In other words, trace back from the cell(s) in the last row with optimal score.

Note that when the first row (or column) of the matrix is initialized to zero, the traceback will end in the first row (or column), but not necessarily in the cell $a[0, 0]$.

Like global alignment, either distance or similarity scoring can be used for semiglobal alignment. There may be more than one optimal semiglobal alignment.

Local pairwise alignment

Global and semiglobal alignment are used in cases where we expect that s and t are related from end to end. Semi-global allows for some gaps at the beginning or end of one sequence, but the underlying assumption is the same: s and t share a relationship within the entire aligned region. In contrast, local alignment is used in cases where s and t share one or more local regions that are related, but are not related from end to end.

The alignment of any substring $s_{h,i} = s[h \ldots i]$ of s and any substring $t_{j,k} = t[j \ldots k]$ of t is a local alignment of s and t. The optimal alignment of $s_{h,i}$ and $t_{j,k}$ is $\alpha^*(s_{h,i}, t_{j,k})$, the highest scoring global alignment of those substrings. The optimal local alignment of s and t is the highest scoring optimal alignment of all possible substrings of s and t, that is,

$$\alpha^*(s, t) = \underset{h,i,j,k}{\text{argmax}}\, S(\alpha^*(s_{h,i}, t_{j,k})),$$

where $1 \leq h \leq i \leq m$, and $1 \leq j \leq k \leq n$ and S is the similarity scoring function defined in Equation 2. Note that there may be more than one optimal local alignment. High scoring sub-optimal alignments may also be of interest.

For local alignment, the pairwise alignment dynamic programming algorithm must be modified to allow the alignment to start and stop anywhere in s and t. Unlike the dynamic programs for global alignment, the local alignment recurrence (Equation 6) has a fourth term that sets the score $a[i, j]$ to zero whenever adding a substitution or a gap to the alignment results in a negative score. This is what allows the local alignment algorithm to consider all possible starting positions in s and in t.

Initialization:

Set the first row and column to zero: $a[i, 0] = 0$ and $a[0, j] = 0$ for all i and j.

Recurrence:

$$a[i,j] = max \begin{cases} a[i, j-1] + g \\ a[i-1, j-1] + p(s[i],\ t[j]) \\ a[i, j-1] + g \\ 0 \end{cases} \tag{6}$$

Optimal alignment score:

The score of the optimal alignment is $\max_{i,j} a[i, j]$, where the maximum is taken over all i and all j.

Trace back:

Trace back starting at $\mathcal{T}[i^*, j^*]$, where $(i^*, j^*) = \text{argmax}_{i,j}\, a[i, j]$ is the cell corresponding to the maximum score. End the trace back at the first cell with value zero.

The point at which $a[i, j]$ drops below zero depends on the scoring function and critically determines what the resulting alignment will look like. For this reason, scoring functions for local alignment are subject to more stringent constraints than scoring functions for global and semi-global alignment.

In order to find biologically meaningful conserved regions, a scoring function for local pairwise alignment must satisfy the following requirements:

- $M > m > 2g$.
- The scoring function must be a similarity function. The local alignment that minimizes the edit distance (weighted or unweighted) is the empty alignment, which tells us nothing.
- There must be at least one pair of residues, x and y, for which the similarity score *p(x,y)* is positive. Otherwise, the optimal alignment is always the empty alignment.

- The expected alignment score of a pair of randomly generated sequences (i.e., sequences sampled from a background distribution) must be negative.

These rules apply to general scoring functions, where the score for each pair of residues can have a different numerical value. For our simple similarity function, we require a positive score for matches ($M > 0$) and a negative score for mismatches ($m < 0$) and gaps ($g < 0$).