Computational Molecular Biology

D. Durand

September 2, 2024

Contents

Chapter 1

Sequence Alignment

Pairwise sequence alignment seeks 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 in cases where we have reason to believe that the sequences are related along their entire length. If, for example, sequences s_1 and s_2 are two independent sequencing runs of the same PCR product, then they should differ only at those positions where there are sequencing errors. In order to find those sequencing errors, we align all of sequence s_1 with all of sequence s_2. 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 is used in situations where we believe that s_1 and s_2 are related along the entire length of the region where they overlap. For example, if s_1 is a segment of genomic DNA containing a prokaryotic gene and s_2 is the mRNA transcript produced when the gene in s_1 is expressed, every base in s_2 corresponds to a base in s_1, but not the reverse is not true. The bases immediately up- and downstream of the gene appear in the genomic DNA, but not in the transcript. Semi-global alignment "jumps" over those flanking regions in s_1 without exacting a penalty, but forces an alignment along the entire length of s_2.

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

Box 1: Notation for pairwise alignment

Alphabet:

An *alphabet*, denoted by Σ, is a finite, unordered set of symbols; e.g.,

DNA: $\Sigma_D = \{A, C, G, T\}$
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.

Σ^* denotes 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 n, we use $s[1]s[2]\cdots s[n]$ to denote the symbols in s.

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 n, the substring that begins with $s[i]$ and ends with $s[j]$ is denoted $s[i..j]$, $1 \leq i \leq j \leq n$. The sequence `CAT` is a substring of `CATTAG`. `CATA` is not.

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

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

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 in Box 1. In Section 1.1, we introduce a formal definition of a global alignment. There are many ways to align a given pair of sequences. We consider scoring functions that assess the quality of an alignment as a basis for quantitative comparison of different alignments. Finally, we provide an efficient algorithm to find the alignment that is optimal with respect to the scoring function. Local and semi-global alignments are discussed in Sections 1.3 and 1.2.

1.1 Global pairwise alignment

Given a pair of sequences, s_1 and s_2, the goal of global sequence alignment is to insert gaps in s_1 and s_2 such that residues that share a property of interest are found in the same column. Suppose that $s_1 = \texttt{CATCAC}$ and $s_2 = \texttt{CTAGC}$ are sequences of lengths 6 and 5, respectively. By introducing a gap after the first symbol in s_2, we obtain an alignment, $\alpha(s_1, s_2)$, of length $l = 6$ with three matches, two mismatches, and a gap:

```
CATCAC
C-TAGC
```

This is just one of many ways to align these sequences. A different alignment can be obtained by introducing one gap in s_1 and two gaps in s_2:

```
CATCA-C
C-T-AGC
```

This alignment is longer ($l = 7$) and has four matches, three gaps and no mismatches. The second alignment implies that the `A`'s in s_1 and s_2 share a relationship; the first alignment does not.

Stacking the sequences vertically provides a visually intuitive representation that places related residues in the same column. The same information can be represented in a more compact form by simply indicating the positions of the gaps in the two sequences. For example, the first alignment can also be represented as $\alpha(s_1, s_2) = \{s'_1, s'_2\}$, where $s'_1 = \texttt{CATCAC}$ and $s'_2 = \texttt{C-TAGC}$. Note that although s_1 and s_2 have different lengths, the alignment $\alpha(s_1, s_2)$, s'_1, and s'_2 are all of the same length ($l = 6$).

Because there are many different alignments of the same pair of sequences, we use superscripts to distinguish between them when more than one alignment is under consideration. Using this notation, the two alignments we introduced above are $\alpha^1(s_1, s_2) = \{s^1_1, s^1_2\}$,

[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.

where $s_1^1 =$ `CATCAC` and $s_2^1 =$ `C-TAGC` and $\alpha^2(s_1, s_2) = \{s_1^2, s_2^2\}$, where $s_1^2 =$ `CATCA-C` and $s_2^2 =$ `C-T-AGC`.

These ideas are the basis for the formal definition of a global sequence alignment. Given sequence $s_1 \in \Sigma^*$ of length n_1 and sequence $s_2 \in \Sigma^*$ of length n_2, $\alpha^\kappa(s_1, s_2) = \{s_1^\kappa, s_2^\kappa\}$ is a global alignment of s_1 and s_2 if and only if

- $s_1^\kappa, s_2^\kappa \in (\Sigma')^*$, where $\Sigma' = \Sigma \cup \{\text{`-'}\}$ is the alphabet expanded to include the gap symbol;
- $|s_1^\kappa| = |s_2^\kappa| = l^\kappa$, where $\max(n_1, n_2) \leq l^\kappa \leq n_1 + n_2$;
- s_1 is the subsequence obtained by removing '-' from s_1^κ and s_2 is the subsequence obtained by removing '-' from s_2^κ;
- there is no value of i for which $s_1^\kappa[i] = s_2^\kappa[i] =$ '-'.

The length of the alignment is bounded below by $\max(n_1, n_2)$ because the alignment cannot be shorter than the longer of the two sequences. The longest possible alignment is one where every symbol in s_1 is aligned with a gap in s_2 and vice versa. In this case the length of the alignment is $l^\kappa = n_1 + n_2$. A longer alignment is not possible without introducing a site with gaps in both sequences which is a violation of the fourth criterion. The position in the alignment is indexed by columns; column i is often called *site* i. When considering site i, $s_1^\kappa[i]$ and $s_2^\kappa[i]$ are the symbols in s_1 and s_2 that are aligned. If $s_2^\kappa[i] =$ `-`, we say that there is a gap in s_2 at site i.

1.1.1 Global sequence alignment with similarity scoring

Among the many possible ways to align s_1 and s_2, our goal is to find the global alignment that best captures the relationship between them. This is fundamentally a biological question. From a practical perspective, we use a mathematical approach: We introduce an objective criterion that quantifies how well α^κ captures the relationship between s_1 and s_2. We then seek the alignment that optimizes that criterion.

Given sequences s_1 and s_2 and an alignment $\alpha^\kappa(s_1, s_2) = \{s_1^\kappa, s_2^\kappa\}$, it is convenient to assign a score to alignment α^κ that quantifies how well α^κ captures the relationship between s_1 and s_2. One approach is to use a scoring function that reflects the similarity between the sequences. In this case, the alignment that maximizes the scoring function is considered to be the optimal alignment. Alternatively, we can introduce a function that quantifies the distance between the two sequences (for example, the number of changes needed to convert one sequence into the other) and seek the alignment that minimizes that distance.

In computational biology, similarity functions are more versatile and more widely used, so we will start with similarity scoring, where a higher score indicates a better alignment.

Given sequences s_1 and s_2 and an alignment $\alpha^\kappa(s_1, s_2) = \{s_1^\kappa, s_2^\kappa\}$, the similarity of $\alpha^\kappa(s_1, s_2)$ is defined to be

$$\mathcal{S}(\alpha^\kappa(s_1, s_2)) = \sum_{i=1}^{l^\kappa} S[s_1^\kappa[i], s_2^\kappa[i]], \tag{1.1}$$

where $S[x, y]$ is a score that reflects the similarity of x and y and $S[x, \_]$ is the gap score. The optimal alignment is the alignment that maximizes the similarity between s_1 and s_2:

$$\alpha^*(s_1, s_2) = \underset{\kappa}{\operatorname{argmax}}\, \mathcal{S}\big(\alpha^\kappa(s_1, s_2)\big).$$

There may be more than one optimal alignment.

In general, amino acid alignments are scored with a substitution matrix, S, that assigns a similarity score to each pair of amino acid residues. Broadly speaking, a higher similarity score will be given to a site where the same amino acid is observed in both sequences ($S[x, x]$) than to a site where the amino acids in the two sequences are different ($S[x, y], x \neq y$). However, some amino acid matches will be assigned higher similarity scores than others. When different amino acids are aligned ($x \neq y$), $S[x, y]$ is typically higher when x and y have similar biochemical properties. Examples of amino acid substitution matrices used to score alignments include the PAM and BLOSUM matrices. Substitution matrices are used less often with nucleic acid sequences because the alphabet is smaller (four nucleotides versus 20 amino acids) and the biophysical properties of nucleotides are less varied. In Chapter 3, we will discuss how amino acid substitution matrices are derived.

In this chapter, we will ignore these biological nuances and consider a simple similarity scoring function that treats all symbols in Σ equally. In this simple system, the symbols at given site in an alignment may be the same (a match) or they may differ (a mismatch):

$$S[x, y] = \begin{cases} M & \text{if } x = y, \\ m & \text{if } x \neq y, \end{cases} \tag{1.2}$$

where M is a match score and m is the mismatch score. For this simple scoring function, we require that $M > m$, because matches are preferred over mismatches. The third case of interest arises when a residue is aligned with the gap character ($'\_'$). This case is scored with a gap penalty[2],

$$S[x, \_] = g$$

. The value of g is selected so that a substitution is preferred over two gaps (i.e., $m > 2g$). A scoring function with $m < 2g$ would exclude the possibility of an alignment with

[2]Strictly speaking, the gap score is not an integral part of the substitution matrix. We use this notation here for convenience, as in the sum in Equation 1.1.

substitutions, because the score of any substitution could be improved by replacing it with two gaps.

A final requirement is that the scoring function be symmetric; i.e., $S[x, y] = S[y, x]$ and $S[x, \_] = S[\_, x]$. Given a column with symbols x and y in a pairwise alignment, we have no way of knowing whether the ancestral symbol was an x that was later replaced by a y, or vice versa. (It is also possible that the ancestor was neither x nor y and some combination of substitutions gave rise to x in one sequence and to y in the other.) For this reason, the same score is assigned when x in s_1 is aligned with y in s_2 as when y in s_1 is aligned with x in s_2. Similarly, when a symbol, x, in sequence s_1 is aligned with a gap in sequence s_2 (or vice versa), there is no way to know whether x was inserted in s_1 or deleted from s_2. For this reason, sites with gaps are also called "indels" to indicate that either an insertion or a deletion may have occurred. Our very simple scoring function (Equation 1.2) is, by definition, symmetric because all mismatches have the same score. In Chapter 3, we will see that more complex similarity functions have this property.

Note that the score of an alignment is defined to be the sum of the scores for the individual positions in the alignment (Equations 1.1), 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 and other mathematical analyses.

1.1.2 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_1 and s_2 is $O(2^{n_1+n_2})$. (Convince yourself this is the case.)

Dynamic programming can be used to find the optimal alignment efficiently. This strategy takes advantage of the fact that every prefix of an optimal pairwise alignment is the optimal alignment of a prefix of s_1 and a prefix of s_2. This means that the optimal alignment of pairs of progressively longer prefixes of s_1 and s_2 can be obtained by extending the optimal alignment of shorter prefixes of s_1 and s_2. It is not necessary to examine a suboptimal alignment of prefixes in order to find the optimal alignment of the full length strings.

The dynamic programs for all three sequence alignment problems compute an n_1 x n_2 *alignment matrix* $\mathcal{A}$, where $\mathcal{A}[i, j]$ is the score of the optimal alignment of the prefixes $s_1[1..i]$ and $s_2[1..j]$, that is, the prefixes of s_1 and s_2 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 $\mathcal{A}$.
- A recurrence relation that specifies how to calculate the value of $\mathcal{A}[i,j], i>0, j>0$, from the values of neighboring cells.
- Determination of the score of the optimal alignment from the entries in matrix $\mathcal{A}$.
- A procedure to trace back through the matrix to obtain the optimal alignment. This is achieved by storing the information required to construct the optimal alignment in an n_1 x n_2 *traceback matrix*, $\mathcal{T}$.

The calculation of the optimal global alignment of s_1 and s_2 requires two passes through the $\mathcal{A}$ and $\mathcal{T}$ matrices. In the first pass, the dynamic program proceeds from the upper left to the lower right corner of $\mathcal{A}$, populating the cells in $\mathcal{A}$. Whenever a value is assigned to a cell in $\mathcal{A}$, the associated pointer is entered into the corresponding cell in $\mathcal{T}$. In the second pass, the optimal alignment is constructed from the pointers in $\mathcal{T}$. The details of each of these steps are what differentiate global, semi-global, and local alignment.

A formal statement of the dynamic program for global sequence alignment is given below. In the first pass, the dynamic program progresses from $\mathcal{A}[0,0]$ to $\mathcal{A}[n_1,n_2]$, populating the cells in $\mathcal{A}$ with the scores of progressively longer optimal alignments of prefixes. At each iteration, the value of $\mathcal{A}[i,j]$ is calculated from the values of $\mathcal{A}[i-1,j]$, $\mathcal{A}[i-1,j-1]$, and $\mathcal{A}[i,j-1]$.

The initialization step calculates the values in the first row and column of $\mathcal{A}$. The score in $\mathcal{A}[0,j]$ reflects the alignment obtained by inserting gaps at the beginning of s_1 and aligning them to the first j symbols in s_2 . Similarly $\mathcal{A}[i,0]$ is the score of the alignment of i gaps inserted prior to the first symbol in s_2 with the first i symbols in s_1.

The internal entries in $\mathcal{A}$ are calculated by the recurrence relations. The value in $\mathcal{A}[i,j]$ is the score of the optimal alignment of the first i symbols in s_1 (i.e., $s_1[1..i]$) with the first j symbols in s_2 (i.e., $s_2[1..j]$). The optimal alignment of $s_1[1..i]$ and $s_2[1..j]$ is obtained by inserting one additional column at the end of a shorter optimal alignment. There are three optimal alignments of prefixes of s_1 and s_2 that, when extended in this way, will yield a candidate optimal alignment of $s_1[1..i]$ and $s_2[1..j]$.

1. The column $\begin{smallmatrix}-\\ s_2[j]\end{smallmatrix}$ can be added to the end of the optimal alignment of $s_1[1..i]$ and $s_2[1..j-1]$. The alignment score of this prefix is stored in $\mathcal{A}[i,j-1]$, the cell to the immediate left of $\mathcal{A}[i,j-1]$.
2. The column $\begin{smallmatrix}s_1[i]\\ s_2[j]\end{smallmatrix}$ can be added to the end of the optimal alignment of $s_1[1..i-1]$ and $s_2[1..j-1]$, which corresponds to $\mathcal{A}[i-1,j-1]$, the cell diagonally up and to the left of $\mathcal{A}[i,j]$.
3. The column $\begin{smallmatrix}s_1[i]\\ -\end{smallmatrix}$ can be added to the end of the optimal alignment of $s_1[1..i-1]$ and $s[1..j]$, which corresponds to the cell immediately above $\mathcal{A}[i,j]$.

Of these candidates, the alignment that optimizes the scoring function is the optimal alignment of $s_1[1..i]$ and $s_2[1..j]$. The score of this alignment is entered in $\mathcal{A}[i,j]$. The indices of the entry (or entries) in $\mathcal{A}$ that optimize the right hand side of the recurrence relation (Equation 1.3) are stored in the traceback matrix, $\mathcal{T}$. $\mathcal{T}[i,j]$ contains the index of an adjacent cell to the left $(i,j-1)$, upper left $(i-1,j-1)$, and/or above $(i-1,j)$ the current cell. In class, we use arrows ($\leftarrow$,$\nwarrow$, and $\uparrow$) to designate these indices. Note that more than one of the recurrence cases may optimize the value of $\mathcal{A}[i,j]$. In this case, more than one pointer will be added to $\mathcal{T}[i,j]$.

The first pass completes when all entries in the matrix $\mathcal{A}$ have been assigned values. At

DYNAMIC PROGRAM FOR GLOBAL ALIGNMENT: SIMILARITY SCORING

Input:

Sequences s_1 and s_2 of lengths n_1 and n_2, respectively.

Initialization:

$$\begin{aligned} \mathcal{A}[0,j] &= \mathcal{A}[0,j-1]+g \qquad \text{(top row)} \\ \mathcal{A}[i,0] &= \mathcal{A}[i-1,0]+g \qquad \text{(left column)} \end{aligned}$$

Recurrence relation:

$$\mathcal{A}[i,j] = \max \begin{cases} \mathcal{A}[i,j-1]+g \\ \mathcal{A}[i-1,j-1]+S[s_1[i],\, s_2[j]] \\ \mathcal{A}[i-1,j]+g \end{cases} \tag{1.3}$$

$$\mathcal{T}[i,j] = \underset{i,j}{\text{argmax}} \begin{cases} \mathcal{A}[i,j-1]+g & \leftarrow \\ \mathcal{A}[i-1,j-1]+S[s_1[i],\, s_2[j]] & \nwarrow \\ \mathcal{A}[i-1,j]+g & \uparrow \end{cases} \tag{1.4}$$

Optimal alignment score:

The score of the optimal alignment is $\mathcal{A}[n_1,n_2]$.

Trace back:

Follow the pointers from $\mathcal{T}[n_1,n_2]$ to $\mathcal{T}[0,0]$ to obtain the optimal alignment.

this point, $\mathcal{A}[n_1, n_2]$ contains the score of the full length optimal alignment, but the actual alignment has not been explicitly determined. In the second pass, the alignment, α, is constructed in reverse order by following the pointers through $\mathcal{T}$ from the lower right corner to the upper left corner. This traceback procedure begins with an alignment consisting of only the last column of the alignment, which is determined from the indices in $\mathcal{T}[n_1, n_2]$, like this:

if $\mathcal{T}[n_1, n_2] = \leftarrow$
  $\alpha = \begin{smallmatrix} - \\ s_2[n_2] \end{smallmatrix}$
  $i = n_1,\ j = n_2 - 1$
if $\mathcal{T}[i, j] = \nwarrow$
  $\alpha = \begin{smallmatrix} s_1[n_1] \\ s_2[n_2] \end{smallmatrix}$
  $i = n_1 - 1,\ ,\ j = n_2 - 1$
if $\mathcal{T}[n_1, n_2] = \uparrow$
  $\alpha = \begin{smallmatrix} s_1[n_1] \\ - \end{smallmatrix}$
  $i = n_1 - 1,\ j = n_2$

The second pass reconstructs the optimal alignment(s) by tracing back though $\mathcal{T}$ from the lower right ($(i = n_1, j = n_2)$ corner to the upper left corner ($i = 0, j = 0$). At each iteration, α is extended by inserting an additional column at the beginning of α, according to the following rules:

if $\mathcal{T}[i, j] = \leftarrow$
  insert $\begin{smallmatrix} - \\ s_2[j] \end{smallmatrix}$ at the beginning of α
  $j = j - 1$
if $\mathcal{T}[i, j] = \nwarrow$
  insert $\begin{smallmatrix} s_1[i] \\ s_2[j] \end{smallmatrix}$ at the beginning of α
  $i = i - 1;\ j = j - 1$
if $\mathcal{T}[i, j] = \uparrow$
  insert $\begin{smallmatrix} s_1[i] \\ - \end{smallmatrix}$ at the beginning of α
  $i = i - 1$

If an entry in $\mathcal{T}$ with more than one pointer is encountered during the traceback, then there is more than one optimal alignment of s_1 and s_2. Multiple passes through $\mathcal{T}$ are required to

generate all optimal alignments, with additional bookkeeping to ensure that each optimal alignment is constructed once and only once.

This two pass alignment algorithm outputs an optimal alignment score, $\mathcal{A}[n_1, n_2]$, and one or more optimal global alignments. In the first pass, the dynamic program computes the scores of all pairs of prefixes in $O(n_1 \cdot n_2)$ time. The trace back through the alignment matrix to obtain the optimal alignment requires $O(n_1 + n_2)$ time for each optimal alignment. Note that with similarity scoring, the entries in $\mathcal{A}$ may be positive or negative; the optimal alignment score may also be positive or negative.

1.1.3 Global sequence alignment with distance scoring

With similarity scoring, we seek the alignment that maximizes the similarity between s_1 and s_2. With distance-based scoring, the optimal alignment is one that minimizes the distance between s_1 and s_2. One advantage of this approach is that distance functions are metrics, in the formal sense, and have nice mathematical properties. The alignment distance also has a concrete interpretation: the minimum alignment distance corresponds to the smallest number of operations required to transform one sequence into another. In computational biology, a major disadvantage of distance scoring is that it can be used for global and semi-global alignment, but not for local alignment. In addition, distance functions penalize mismatches and gaps, but do not reward matches.

We define the distance score of an alignment $\alpha^\kappa(s_1, s_2) = \{s_1^\kappa, s_2^\kappa\}$ to be

$$\begin{aligned} D\big(\alpha^\kappa(s_1, s_2)\big) &= D(s_1^\kappa, s_2^\kappa) \\ &= \sum_{i=1}^{l^\kappa} d(s_1^\kappa[i], s_2^\kappa[i]), \end{aligned} \tag{1.5}$$

where $d(x, y)$ is the distance between a pair of symbols x and y in Σ' and l^κ is the length of the alignment. The optimal alignment, denoted α^*, is the alignment that minimizes the distance between s_1 and s_2:

$$\alpha^*(s_1, s_2) = \operatorname*{argmin}_\kappa D\big(\alpha^\kappa(s_1, s_2)\big).$$

The function specifying the distance between pairs of symbols must satisfy the following constraints, for all x, y, and z in Σ':

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

The first three constraints guarantee that $D(s_1^\kappa, s_2^\kappa)$ is a metric. In particular, D satisfies the triangle inequality. This means that the penalty for replacing x with y is never greater than the penalty for first replacing x with z and then replacing z with y. When $z =$ '_', this says that deleting x and then inserting y is never an improvement on a direct substitution of x with y. One consequence of the triangle inequality is that the cost of a substitution can never be greater than twice the cost of an indel. The symmetric property, $d(x, y) = d(y, x)$, implies that there is no directionality in the scoring system; we have no information about the order in which events occured. Like similarity scoring, distance scoring assumes positional independence; Equation 1.5 is the sum of the distance scores for the individual positions in the alignment. If $d(x, y) = 1$ and $d(x, \_) = 1$, $\forall x, y$, then

DYNAMIC PROGRAM FOR GLOBAL ALIGNMENT: DISTANCE SCORING

Input:

Sequences s_1 and s_2 of lengths n_1 and n_2, respectively.

Initialization:

$$\begin{aligned} \mathcal{A}[0, j] &= \mathcal{A}[0, j-1] + d(\_, s_2[j]) \qquad \text{(top row)} \\ \mathcal{A}[i, 0] &= \mathcal{A}[i-1, 0] + d(s_1[i], \_) \qquad \text{(left column)} \end{aligned}$$

Recurrence relation:

$$\mathcal{A}[i, j] = \min \begin{cases} \mathcal{A}[i, j-1] + d(\_, s_2[j]) \\ \mathcal{A}[i-1, j-1] + d(s_1[i],\, [j]) \\ \mathcal{A}[i-1, j] + d(s_1[i], \_) \end{cases} \tag{1.6}$$

$$\mathcal{T}[i, j] = \underset{i,j}{\operatorname{argmin}} \begin{cases} \mathcal{A}[i, j-1] + d(\_, s_2[j]) & \leftarrow \\ \mathcal{A}[i-1, j-1] + d(s_1[i],\, [j]) & \nwarrow \\ \mathcal{A}[i-1, j] + d(s_1[i], \_) & \uparrow \end{cases} \tag{1.7}$$

Optimal alignment score:

The score of the optimal alignment is $\mathcal{A}[n_1, n_2]$.

Trace back:

Follow the pointers from $\mathcal{T}[n_1, n_2]$ to $\mathcal{T}[0, 0]$ to obtain the optimal alignment.

$D\big(\alpha^*(s_1, s_2)\big)$ corresponds to the minimum number of operations required to transform s_1 into s_2, where the operations are substitution, insertion, and deletion. This is called the *edit distance*. If $d(x, y) \neq 1$ or $d(x, \_) \neq 1$ or both, then $D(\alpha^*(s_1, s_2))$ is called the *weighted edit distance*.

Global alignment with a distance metric uses the same the two pass procedure described above for global alignment with similarity scoring. However, the details of the initialization and recurrence differ. And in contrast to similarity scoring, with distances, all entries in $\mathcal{A}$ are non-negative, since $d(x, y) \geq 0, \forall\, x, y$.

1.2 Local pairwise alignment

Global alignment is used in cases where we expect that s_1 and s_2 are related from end to end. Semi-global allows for some gaps at the beginning and/or end of one sequence, but the underlying assumption is the same: s_1 and s_2 share a relationship within the entire aligned region. In contrast, local alignment is used in cases where s_1 and s_2 share one or more local regions that are related, but are not related from end to end.

The alignment of any substring $s_1[h..i]$ of s_1 and any substring $s_2[j..k]$ of s_2 is a local alignment of s_1 and s_2. The optimal alignment of $s_1[h..i]$ and $s_2[j..k]$ is the highest scoring *global* alignment of those substrings, where $1 \leq h \leq i \leq n_1$, and $1 \leq j \leq k \leq n_2$ and S is the similarity scoring function defined in Equation 1.1. Note that there may be more than one. The optimal *local* alignment of s_1 and s_2 is the highest scoring optimal alignment of all possible substrings of s_1 and s_2, that is,

$$\alpha^*(s_1, s_2) = \underset{h,i,j,k}{\operatorname{argmax}}\, S(\alpha^*(s_1[h..i], s_2[j..k])).$$

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 (see p. 13) must be modified to allow the alignment to start and stop anywhere in s_1 and s_2. Unlike the dynamic program for global alignment, the local alignment recurrence (Equation 1.8) has a fourth term that sets the score $\mathcal{A}[i, j]$ 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_1 and in s_2.

The value in $\mathcal{A}[i, j]$ is the score of the optimal local alignment of a substring in $s_1[1..i]$ and a substring in $s_2[1..j]$. The score of the optimal local alignment over all possible substrings of s_1 and s_2 is the maximum value in $\mathcal{A}$, taken over all i and all j. The alignment itself is obtained by tracing back through $\mathcal{T}$, starting from the cell corresponding to the maximum score. The alignment is constructed in reverse order by following the pointers through $\mathcal{T}$ until end symbol "$\Diamond$" is encountered; this corresponds to reaching the first

DYNAMIC PROGRAM FOR LOCAL ALIGNMENT: SIMILARITY SCORING

Input:

Sequences s_1 and s_2 of lengths n_1 and n_2, respectively.

Initialization:

$$\begin{aligned} \mathcal{A}[0,j] &= 0,\ \forall_j \\ \mathcal{A}[i,0] &= 0,\ \forall_i \end{aligned}$$

Recurrence relation:

$$\mathcal{A}[i,j] = max \begin{cases} \mathcal{A}[i,j-1] + g \\ \mathcal{A}[i-1,j-1] + S[s_1[i],\ s_2[j]] \\ \mathcal{A}[i-1,j] + g \\ 0 \end{cases} \tag{1.8}$$

$$\mathcal{T}[i,j] = \underset{i,j}{\text{argmax}} \begin{cases} \mathcal{A}[i,j-1] + g & \leftarrow \\ \mathcal{A}[i-1,j-1] + S[s_1[i],\ s_2[j]] & \nwarrow \\ \mathcal{A}[i-1,j] + g & \uparrow \\ 0 & \Diamond \end{cases} \tag{1.9}$$

Optimal alignment score:

The score of the optimal alignment is

$$\max_{i,j} \mathcal{A}[i,j].$$

Trace back:

Follow the pointers from $\mathcal{T}[i^*,j^*]$, where $(i^*,j^*) = \text{argmax}_{i,j}\, \mathcal{A}[i,j]$, and end the trace back at the first cell with value zero encountered.

zero-valued cell in $\mathcal{A}$ on the trace back path. This is in contrast to global alignment where the trace back starts in the lower right hand corner and proceeds to position $i = 1, j = 1$.

At each step,

If $\mathcal{T}[i,j] = \leftarrow$
  align $s_2[j]$ with a gap '_'
  $j = j-1$
If $\mathcal{T}[i,j] = \nwarrow$
  align $s_1[i]$ with $s_2[j]$
  $i = i-1$; $j = j-1$
If $\mathcal{T}[i,j] = \uparrow$
  align $s_1[i]$ with a gap '_'
  $i = i-1$
If $\mathcal{T}[i,j] = \Diamond$
  align $s_1[i]$ with $s_2[j]$ and end
  $i = 0$; $j = 0$

The point at which $\mathcal{A}[i,j]$ drops below zero and restarts 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:

- The scoring function must be a similarity function. Edit distance penalizes mismatches and gaps, but does not reward matches and is therefore not appropriate for finding local regions of similarity.
- There must be at least one pair of residues, x and y, for which the similarity score $S[x,y]$ is positive. Without this requirement, all entries in the alignment matrix, $\mathcal{A}$, would be set to zero.
- The expected alignment score of a pair of randomly generated sequences (i.e., sequences sampled from a background distribution) must be negative.
- The standard requirements for the similarity function must be upheld ($m > 2g$).

The rationale for a negative expected alignment score is as follows: The goal of local alignment is to find similar regions in a pair of sequences, $s_1, s_2 \in \Sigma^*$. The optimal local alignment obtained from the dynamic programming algorithm will depend on the function used to score matches and mismatches. We seek a scoring function that will yield local alignments that correspond to biologically meaningful features. However, even

unrelated sequences, when aligned, will match at a few positions. In order to have a strong presumption that a high-scoring local alignment is biologically meaningful, the match and mismatch scores should be chosen in such a way that chance matches contribute little to the alignment score.

If a pair of symbols, sampled at random, were appended to the end of an existing alignment, how much would the alignment score increase? The increase, on average, is given by the expected alignment score per position. Let x be a symbol in Σ and let p_x be the background frequency of x in the genomes from which sequences s_1 and s_2 were sampled. Then, using our simple scoring function, the expected alignment score per position is

$$\begin{aligned}\overline{\mathcal{S}} =& \sum_{x\in\Sigma}\sum_{y\in\Sigma} p_x p_y \; S[x,y] \\ =& \sum_{x\in\Sigma} p_x^2 \; M + \sum_{x\in\Sigma}\sum_{\substack{y\in\Sigma,\\ y\neq x}} p_x p_y \; m,\end{aligned}$$

where M and m are the match and mismatch scores, respectively.

For example, suppose that s_1 and s_2 are DNA sequences with uniform nucleotide frequencies; i.e., $p_A = p_G = p_C = p_T = 0.25$. Since the nucleotide frequencies are uniform, the nucleotide pairs also have uniform frequencies: $p_x p_y = (0.25)^2, \forall x, y$. There are 16 possible pairs of nucleotides; four pairs consist of the same nucleotide (AA, GG, CC, and TT) and 12 pairs are made up of different nucleotides. In this case, the expected alignment score per position is

$$\begin{aligned}\overline{\mathcal{S}} =& 4\;(0.25)^2\; M + 12\;(0.25)^2\; m \\ =& 0.25\; M + 0.75\; m.\end{aligned}$$

Why is it important that the expected alignment score be negative? If the expected score were positive, then extending a local alignment with unrelated pairs of symbols would increase its score. Such an alignment would have a higher score because it is longer, not because it contains stronger evidence of a shared biological relationship. In the nucleotide example given above, M and m should be selected so that

$$0.25\; M + 0.75\; m < 0.$$

The rules above 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 achieve these goals by requiring a positive score for matches ($M > 0$) and a negative score for mismatches ($m < 0$) and gaps ($g < 0$).

1.3 Semi-global alignment

Global alignment seeks the best, full length alignment of a pair of sequences; 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 at the end of the alignment. Another example is aligning cDNAs with genomic DNA to identify gene structure. Because the cDNA corresponds to a small region in the genome, the cDNA fragment will be flanked by gaps at both ends when aligned with the genomic DNA.

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. The optimal semi-global alignment can be found using dynamic programming, with modifications to adapt the dynamic program for global pairwise alignment to the semi-global alignment problem. These modifications are described in terms of alignment with similarity scoring. Similar modifications can be made to obtain a semi-global alignment algorithm that uses distances.

For penalty-free prefixes, the gap penalties in the first column or row are replaced with zeros. To achieve penalty-free gaps at the beginning of s_1, the entries in the first row are set to zero. Allowing penalty-free gaps at the beginning of s_2 requires zeros in the first column. Penalty-free suffixes are achieved by re-defining the optimal alignment score to be the maximum in the last row or column, rather than the value in the cell at the lower right hand corner of the alignment matrix. Considering all cells in the last row allows for penalty-free gaps at the end of s_1 For penalty-free gaps at the end s_2, the algorithm may consider all cells in the last column. As with global alignment, the traceback begins from the cell associated with the optimal score. This means that, unlike global alignment, the traceback can start from a cell in the last row or column, rather than being required to start from the lower right corner.

There may be more than one optimal semi-global alignment. First, two or more cells in the last row (respectively, column) may contain the maximum value. In addition, as with global alignment, for each maximum-valued cell in the last row (respectively, column) there may be more than one path that traces back through in the alignment matrix.

DYNAMIC PROGRAM FOR SEMI-GLOBAL ALIGNMENT: SIMILARITY SCORING

Input:

Sequences s_1 and s_2 of lengths n_1 and n_2, respectively.

Initialization:

To allow penalty-free gaps at the beginning of s_1 (as in Case 1), set the first row to zero and initialize the first column as in global alignment.

$$\begin{aligned} \mathcal{A}[0,j] &= 0,\ \forall_j && \text{(top row)} \\ \mathcal{A}[i,0] &= \mathcal{A}[i-1,0]+g && \text{(left column)} \end{aligned}$$

To allow penalty-free gaps at the beginning of s_2 (as in Case 2), set the first column to zero and initialize the first row as in global alignment.

$$\begin{aligned} \mathcal{A}[0,j] &= \mathcal{A}[0,j-1]+g && \text{(top row)} \\ \mathcal{A}[i,0] &= 0,\ \forall_i && \text{(left column)} \end{aligned}$$

Recurrence relation:

Same as global.

Optimal alignment score and trace back:

To avoid trailing gap penalties at the end of s_1 (as in Case 3), we define the optimal score to be the optimal score in the bottom row

$$\max_j \mathcal{A}[n_1,j].$$

Trace back from $\mathcal{T}[n_1,j^*]$, where $j^* = \text{argmax}_j\, \mathcal{A}[n_1,j]$. In other words, trace back from the cell(s) in the last row with optimal score.

To avoid trailing gap penalties at the end of s_2 (as in Case 4), we define the optimal score to be the optimal score in the last column

$$\max_i \mathcal{A}[i,n_2].$$

Trace back from $\mathcal{T}[i^*,n_2]$, where $i^* = \text{argmax}_i\, \mathcal{A}[i,n_2]$. In other words, trace back from the cell(s) in the last column with optimal score.

	0		S		P		E		L		L		B		I		N		D		I		N		G
0	0		0		0		0		0		0		0		0		0		0		0		0		0
		\	|	\	|	\	|	\	|	\	|	\						\	|	\	|	\	|	\	|
B	-1		-1		-1		-1		-1		-1		2	--	1	--	0	--	-1		-1		-1		-1
		\	|	\	|	\	|	\	|	\	|		|	\		\		\		\	|	\	|	\	|
O	-2		-2		-2		-2		-2		-2		1		1	--	0	--	-1	--	-2		-2		-2
		\	|	\	|	\	|	\	|	\	|		|	\	|	\		\		\		\	|	\	|
U	-3		-3		-3		-3		-3		-3		0		0		0	--	-1	--	-2	--	-3		-3
		\	|	\	|	\	|	\	|	\	|		|	\	|	\					\			
N	-4		-4		-4		-4		-4		-4		-1		-1		2	--	1	--	0		0	--	-1
		\	|	\	|	\	|	\	|	\	|		|	\	|		|	\							
D	-5		-5		-5		-5		-5		-5		-2		-2		1		4	--	3	--	2	--	1

Figure 1.1: Semi-global alignment matrix with similarity scoring ($M = 2, m = -1, g = -1$) and penalty-free gaps at the beginning and end of s_1. With this configuration, there are two optimal semi-global alignments. The traceback starts at the cell corresponding to the maximum value in the last row, highlighted in pink. Arrows along the traceback are indicated in red. Zeros in the first row allow the traceback to terminate any position in the top row.

As an example, Figure 1.1 shows the alignment matrix for the semi-global alignment of $s_1 =$ `BOUND` and $s_2 =$ `SPELLBINDING`, with penalty-free gaps at the beginning and end of s_1. Zeros in the top row allow the algorithm to start late in `SPELLBINDING`. This results in five gaps that are not penalized before the first character in s_1. The alignment terminates at the highest-scoring cell in the last row, resulting in three penalty-free gaps after the last character in s_1. In this example, there is a unique maximum in the bottom row.

With this scoring function, there are two optimal semi-global alignments because there are two paths from the maximum value in the last row to the a zero-valued cell in the first row.

```
s_1:             B O U N D
s_2:   S P E L L B I _ N D I N G
```

and

```
s_1:             B O U N D
s_2:   S P E L L B _ I N D I N G
```

Multiple optimal semi-global alignments could also arise if there are two or more maximum-valued cells in the last row, but that is not the case here. To emphasize that indels at the beginning and end of s_1 are penalty-free, we do not show the gap symbols before and after

BOUND. This distinguishes leading and lagging indels from the indel in SPELLBINDING that *is* penalized, because it is in the middle of the alignment.

In the example in Fig. 1.1, the semi-global alignment algorithm was configured to allow gaps to be penalty-free at the beginning and end of s_1. In general, semi-global alignment can be set up to allow penalty-free gaps at the beginning of either sequence. There are eight possible cases to consider:

1. Gaps are penalty-free at the beginning of s_1 ($\mathcal{A}[0, j] = 0, \forall_j$); *e.g.*,

```
s_1:       D O
s_2:   R E D O
```

2. Gaps are penalty-free at the beginning of s_2 ($\mathcal{A}[i, 0] = 0, \forall_i$); *e.g.*,

```
s_1:   R E D O
s_2:       D O
```

3. Gaps are penalty-free at the end of s_1 (opt $= \max_j \mathcal{A}[n_1, j]$); *e.g.*,

```
s_1:   D O
s_2:   D O N E
```

4. Gaps are penalty-free at the end of s_2 (opt $= \max_i \mathcal{A}[i, n_2]$); *e.g.*,; *e.g.*,

```
s_1:   D O N E
s_2:   D O
```

5. Gaps are penalty-free at the beginning and end of s_1; ($\mathcal{A}[0, j] = 0, \forall_j$ and opt $= \max_j \mathcal{A}[n_1, j]$.); *e.g.*,

```
s_1:       D O
s_2:   R E D O N E
```

6. Gaps are penalty-free at the beginning and end of s_2; ($\mathcal{A}[i, 0] = 0, \forall_i$ and opt $= \max_i \mathcal{A}[i, n_2]$.); *e.g.*,

```
s_1:   R E D O N E
s_2:       D O
```

7. Gaps are penalty-free at the beginning of s_1 and at the end of s_2; ($\mathcal{A}[0, j] = 0, \forall_j$ and opt $= \max_i \mathcal{A}[i, n_2]$.); *e.g.*,

```
s_1:       D O N E
s_2:   R E D O
```


8. Gaps are penalty-free at the beginning of s_2 and at the end of s_1; ($\mathcal{A}[i,0] = 0, \forall_i$ and opt $= \max_j \mathcal{A}[n_1, j]$.); *e.g.*,

```
s_1:   R E D O
s_2:       D O N E
```

In semi-global alignment, we do not allow penalty-free gaps at the beginning of s_1 *and* the beginning of s_2 in the same alignment. Nor do we allow penalty-free gaps at the end of s_1 and the end of s_2. Why not?

