2.7 Gap Penalty

Up until now the central elements used to measure the score of an alignment have been matches, mismatches and spaces. Now we introduce another important element, *gaps*. A *gap* is a consecutive run of spaces in an alignment. Gaps help create alignments that better conform to underlying biological models and more closely fit patterns that one expects to find in meaningful alignments. The idea is to treat a gap as a whole, rather than give each of its spaces the same weight. There are many ways a weight/value can be given to a gap. In this section we present several gap penalty models, and an algorithm for finding a best alignment using one of them (affine gap penalty model). The number of gaps in an alignment will be denoted by #gaps.

Definition A *gap* is any maximal, consecutive run of spaces in a single sequence of a given alignment.

Definition The *length* of a gap is the number of indel operations in it.

Example Consider the alignment:
S = `a t t c - - g a - t g g a c c`
T = `a - - c g t g a t t - - - c c`
This alignment has four gaps containing a total of eight spaces. The alignment would be described as having seven matches, no mismatch, four gaps and eight spaces.

2.7.1 Motivation

The concept of a gap in an alignment is important in many biological application, since the insertion or deletion of an entire subsequence often occurs as a single mutational event. Moreover, many of these single mutational events can create gaps of varying sizes. We will need to score a gap as a whole when we try to align two sequences of DNA so as to avoid asigning high cost to these mutations.

At the protein level, two protein sequences might be relatively similar over several intervals but differ in intervals where one contains a protein subunit that the other does not. Again, the introduction of gaps will help us treat these cases as ”good matches”, although there are long consecutive runs of indel operations in them.

One concrete illustration of the use of gaps in the alignment model comes from the problem of cDNA matching [1, chapter 11]. To better understand the problem, we give a short biological background:

Recall that an RNA molecule is transcribed from the DNA of a gene. The RNA transcript (pre-mRNA) is a complement of the gene's DNA, where each A in the gene is replaced by U in the RNA, each T is replaced by A, each C by G, and each G by C. Moreover, the RNA transcript spans the entire gene: introns and exons.

After the pre-mRNA is created, a splicing process takes place: each intron-exon boundary is located, the RNA regions corresponding to the introns are spliced out, and the RNA regions corresponding to exons are concatenated. The resulting RNA molecule is called the *messenger* RNA (*mRNA*). It includes only regions that correspond to exons. The mRNA leaves the cell nucleus and is used to create the protein it encodes.

Each cell (usually) contains a copy of all the chromosomes and hence, of all the genes of the entire individual. Yet, in each specialized cell (a liver cell for example) only a small fraction of the genes are expressed, that is, only a small fraction of the proteins encoded in the genome are actually produced in that specialized cell.

A standard method to determine which proteins are expressed in the specialized cell and to find the location of the encoding genes, uses the mRNA: After "capturing" the mRNA, it is used to create a DNA sequence complementary to it. This sequence is called *cDNA* (complementary DNA). The cDNA contains the same sequence of bases as the exons on the original DNA that encode for the specific protein, with the introns missing. To determine where the gene associates with that cDNA (hence protein) resides, we now need to match the cDNA with the original DNA.

We know, that in the alignment we search for, there are many long gaps. These gaps are due to introns on the DNA that are missing on the cDNA. Using a good gap penalty model will prevent us from giving a very low score for these alignments (since they have many consecutive indel's in them) and allow us to find the true alignment.

2.7.2 Constant Gap Penalty Model

The simplest choice is the *constant* gap penalty, where each individual space is free (has no weight), and each gap is given a weight of W_g independent of its length.

- Let σ denote the weights of match and mismatch only ($\forall x\ \sigma(x,-) = \sigma(-,x) = 0$)
- Let S' and T' represent S and T after inserting spaces.
- Thus we have to find an alignment that maximizes:

$$\Sigma\sigma(S'_i, T'_i) + W_g \times \#gaps$$

A generalization of this model adds weight not only to an existence of a gap (W_g), but also another weight proportional to its length (W_s). This model is described below.

2.7.3 Affine Gap Penalty Model

In the *Affine gap penalty model*, a gap is given two weights. One, W_g is the weight to "open the gap", and the other, W_s is the weight to "extend the gap" with one more space.
The total penalty for a gap of length q is:

$$W_{Total} = W_g + qW_s$$

The model is called "affine" after its affine formula above.

Note that the constant gap weight model is simply the affine model with $W_s = 0$, Thus the algorithm described bellow can be used for the *constant gap penalty model* as well.

Using the same symbols as above, we have to find an alignment that maximizes:

$$\Sigma\sigma(S'_i, T'_i) + W_g \times \#gaps + W_s \times \#spaces$$

Affine gap penalty Algorithm

To align sequences S, T, consider the prefixes $S_{1...i}$ of S and $T_{1...j}$ of T. Any alignment of these two prefixes is one of the following three types:

1. S ————i
   T ————j
   alignment of $S_{1...i}$ and $T_{1...j}$ where characters $S(i)$ and $T(j)$ are aligned opposite each other. This includes both the case that $S_i = T_j$ and that $S_i \neq T_j$.

2. S ————i_ _ _ _ _ _ _
   T ————————j

   alignment of $S_{1...i}$ and $T_{1...j}$ where character S_i is aligned to a character strictly to the left of character T_j. Therefore, the alignment ends with a gap in S.

3. S ————————i
   T ————j_ _ _ _ _ _ _ _

   alignment of $S_{1...i}$ and $T_{1...j}$ where character S_i is aligned to a character strictly to the right of character T_j. Therefore, the alignment ends with a gap in T.

Notation Let us use the following notation:

- $G(i,j)$ the maximum value of any alignment of type 1
- $E(i,j)$ the maximum value of any alignment of type 2
- $F(i,j)$ the maximum value of any alignment of type 3
- $V(i,j)$ the maximum value of an alignment

Using these notations, we can recursively define the alignment table, as was done in previous algorithms. In the base conditions, we need to look at indel operations and assign the correct value: not only the weight of the spaces (qW_s), but also the weight of "opening the gap" (W_g).

We will define 3 recurrence relations, one for each of $G(i,j)$, $E(i,j)$ and $F(i,j)$. Each will be calculated from previously computed values.
Take $E(i,j)$ for example. We are looking at alignments in which S ends to the left of T:
S ————i_ _ _ _ _ _ _
T ——————————j
There are two possible cases for the previous alignment:

1. It looked the same, i.e., S ended to the left of T.
   In this case, we only need to add another "extension weight" to the value, forming the new weight $E(i,j-1)+W_s$
2. S and T ended at the same place (type 1 alignment).
   In this case, we need to add both the gap "opening weight" and the gap "extension weight", forming the new weight $V(i,j-1)+W_g+W_s$.

Taking the maximum of the two yields the value for $E(i,j)$.

Calculating $F(i,j)$ and $G(i,j)$ is done using similar arguments. $V(i,j)$ is Calculated by simply taking the maximum of the three. As in *Global alignment*, we search for the value $V(n,m)$, and trace the alignment back using pointers created while filling the table.

Recursive definition:

- Base conditions: $V(0,0) = 0$
  $V(i,0) = E(i,0) = W_g + iW_s$
  $V(0,j) = F(0,j) = W_g + jW_s$

- Recurrence relation: $V(i,j) = \max\{E(i,j), F(i,j), G(i,j)\}$ where
  $G(i,j) = V(i-1,j-1) + \sigma(S_i, T_j)$
  $E(i,j) = \max\{E(i,j-1) + W_s, V(i,j-1) + W_g + W_s\}$
  $F(i,j) = \max\{F(i-1,j) + W_s, V(i-1,j) + W_g + W_s\}$

Complexity

- **Time complexity.** As before $O(nm)$, as we only compute four matrices instead of one.

- **Space complexity.** There's a need to save four matrices (for E, F, G, and V respectively) during the computation. Hence, $O(nm)$ space is needed for the trivial implementation.

2.7.4 Convex Gap Penalty Model

- Each additional space in a gap contributes less to the gap weight than the previous space.

- This model is said to better describe biological behavior.

- Example: $W_g \log(q)$, where q is the length of the gap.

- The problem is solvable in $O(nm\log(m))$ time [1].

2.7.5 Arbitrary Gap Penalty Model

- Any gap weight function is acceptable (this is the most general case).

- Weight of a gap is an arbitrary function of its length $w(q)$.

- The problem is solvable in $O(nm(m+n))$ time.