6.9 Alignment with Gap Penalties

Mutations are usually caused by errors in DNA replication. Nature frequently deletes or inserts entire substrings as a unit, as opposed to deleting or inserting individual nucleotides. A *gap* in an alignment is defined as a contiguous sequence of spaces in one of the rows. Since insertions and deletions of substrings are common evolutionary events, penalizing a gap of length x as $-\sigma x$ is cruel and unusual punishment. Many practical alignment algorithms use a softer approach to gap penalties and penalize a gap of x spaces by a function that grows slower than the sum of penalties for x indels.

To this end, we define *affine gap penalties* to be a linearly weighted score for large gaps. We can set the score for a gap of length x to be $-(\rho + \sigma x)$, where $\rho > 0$ is the penalty for the introduction of the gap and $\sigma > 0$ is the penalty for each symbol in the gap (ρ is typically large while σ is typically small). Though this may seem to be complicating our alignment approach, it turns out that the edit graph representation of the problem is robust enough to accommodate it.

Affine gap penalties can be accommodated by adding "long" vertical and horizontal edges in the edit graph (e.g., an edge from (i, j) to $(i + x, j)$ of length $-(\rho + \sigma x)$ and an edge from (i, j) to $(i, j + x)$ of the same length) from each vertex to every other vertex that is either east or south of it. We can then apply the same algorithm as before to compute the longest path in this graph. Since the number of edges in the edit graph for affine gap penalties increases, at first glance it looks as though the running time for the alignment algorithm also increases from $O(n^2)$ to $O(n^3)$, where n is the longer of the two string lengths.[11] However, the following three recurrences keep the running time down:

$$\overset{\downarrow}{s}_{i,j} = \max \begin{cases} \overset{\downarrow}{s}_{i-1,j} - \sigma \\ s_{i-1,j} - (\rho + \sigma) \end{cases}$$

$$\overrightarrow{s}_{i,j} = \max \begin{cases} \overrightarrow{s}_{i,j-1} - \sigma \\ s_{i,j-1} - (\rho + \sigma) \end{cases}$$

11. The complexity of the corresponding Longest Path in a DAG problem is defined by the number of edges in the graph. Adding long horizontal and vertical edges imposed by affine gap penalties increases the number of edges by a factor of n.

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) \\ \overset{\downarrow}{s}_{i,j} \\ \overrightarrow{s}_{i,j} \end{cases}$$

The variable $\overset{\downarrow}{s}_{i,j}$ computes the score for alignment between the i-prefix of $\mathbf{v}$ and the j-prefix of $\mathbf{w}$ ending with a deletion (i.e., a gap in $\mathbf{w}$), while the variable $\overrightarrow{s}_{i,j}$ computes the score for alignment ending with an insertion (i.e., a gap in $\mathbf{v}$). The first term in the recurrences for $\overset{\downarrow}{s}_{i,j}$ and $\overrightarrow{s}_{i,j}$ corresponds to extending the gap, while the second term corresponds to initiating the gap. Essentially, $\overset{\downarrow}{s}_{i,j}$ and $\overrightarrow{s}_{i,j}$ are the scores of optimal paths that arrive at vertex (i, j) via vertical and horizontal edges correspondingly.

Figure 6.18 further explains how alignment with affine gap penalties can be reduced to the Manhattan Tourist problem in the appropriate city grid. In this case the city is built on three levels: the bottom level built solely with vertical $\downarrow$ edges with weight $-\sigma$; the middle level built with diagonal edges of weight $\delta(v_i, w_j)$; and the upper level, which is built from horizontal edges $\rightarrow$ with weight $-\sigma$. The lower level corresponds to gaps in sequence $\mathbf{w}$, the middle level corresponds to matches and mismatches, and the upper level corresponds to gaps in sequence $\mathbf{v}$. Also, in this graph there are two edges from each vertex $(i, j)_{middle}$ at the middle level that connect this vertex with vertex $(i + 1, j)_{lower}$ at the lower level and with vertex $(i, j + 1)_{upper}$ at the upper level. These edges model a start of the gap and have weight $-(\rho + \sigma)$. Finally, one has to introduce zero-weight edges connecting vertices $(i, j)_{lower}$ and $(i, j)_{upper}$ with vertex $(i, j)_{middle}$ at the middle level (these edges model the end of the gap). In effect, we have created a rather complicated graph, but the same algorithm works with it.

We have now introduced a number of pairwise sequence comparison problems and shown that they can all be solved by what is essentially the same dynamic programming algorithm applied to a suitably built Manhattan-style city. We will now consider other applications of dynamic programming in bioinformatics.

6.10 Multiple Alignment

The goal of protein sequence comparison is to discover structural or functional similarities among proteins. Biologically similar proteins may not exhibit a strong sequence similarity, but we would still like to recognize resem-

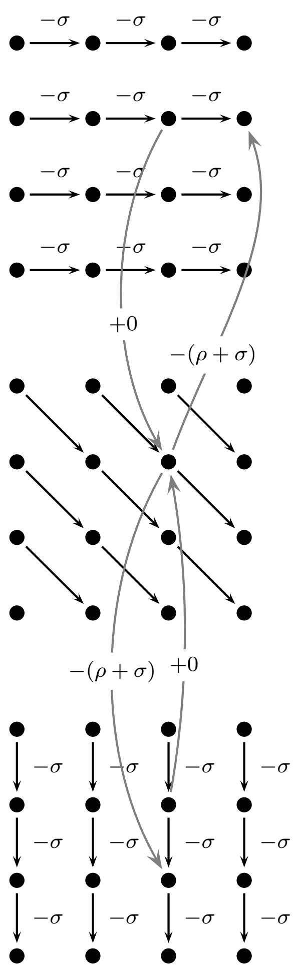

Figure 6.18 A three-level edit graph for alignment with affine gap penalties. Every vertex (i, j) in the middle level has one outgoing edge to the upper level, one outgoing edge to the lower level, and one incoming edge each from the upper and lower levels.