for members of a repeat family in genomic DNA, since normally these are whole copies of the repeat, but sometimes only fragments are seen.

When performing a sequence similarity search we should ideally always consider what types of match we are looking for, and use the most appropriate algorithm for that case. In practice, there are often only good implementations available of a few of the standard cases, and it is often more convenient to use those, and postprocess the resulting matches afterwards.

2.4 Dynamic programming with more complex models

So far we have only considered the simplest gap model, in which the gap score $\gamma(g)$ is a simple multiple of the length. This type of scoring scheme is not ideal for biological sequences: it penalises additional gap steps as much as the first, whereas, when gaps do occur, they are often longer than one residue. If we are given a general function for $\gamma(g)$ then we can still use all the dynamic programming versions described in Section 2.3, with adjustments to the recurrence relations as typified by the following:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j), & \\ F(k,j) + \gamma(i-k), & k = 0,\ldots,i-1, \\ F(i,k) + \gamma(j-k), & k = 0,\ldots,j-1. \end{cases} \tag{2.15}$$

which gives a replacement for the basic global dynamic relation. However, this procedure now requires $O(n^3)$ operations to align two sequences of length n, rather than $O(n^2)$ for the linear gap cost version, because in each cell (i,j) we have to look at $i+j+1$ potential precursors, not just three as previously. This is a prohibitively costly increase in computational time in many cases. Under some conditions on the properties of $\gamma()$ the search in k can be bounded, returning the expected computational time to $O(n^2)$, although the constant of proportionality is higher in these cases [Miller & Myers 1988].

Alignment with affine gap scores

The standard alternative to using (2.15) is to assume an affine gap cost structure as in (2.5): $\gamma(g) = -d - (g-1)e$. For this form of gap cost there is once again an $O(n^2)$ implementation of dynamic programming. However, we now have to keep track of multiple values for each pair of residue coefficients (i,j) in place of the single value $F(i,j)$. We will initially explain the process in terms of three variables corresponding to the three separate situations shown in Figure 2.4, which we show again here for convenience.

I G A x_i	A I G A x_i	G A x_i – –
L G V y_j	G V y_j – –	S L G V y_j

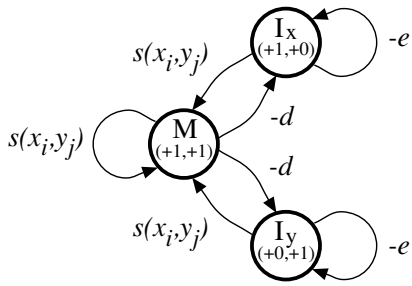

Figure 2.9 *A diagram of the relationships between the three states used for affine gap alignment.*

Let $M(i,j)$ be the best score up to (i,j) given that x_i is aligned to y_j (left case), $I_x(i,j)$ be the best score given that x_i is aligned to a gap (in an insertion with respect to y, central case), and finally $I_y(i,j)$ be the best score given that y_j is in an insertion with respect to x (right case).

The recurrence relations corresponding to (2.15) now become

$$\begin{aligned} M(i,j) &= \max \begin{cases} M(i-1,j-1)+s(x_i,y_j), \\ I_x(i-1,j-1)+s(x_i,y_j), \\ I_y(i-1,j-1)+s(x_i,y_j); \end{cases} \\ I_x(i,j) &= \max \begin{cases} M(i-1,j)-d, \\ I_x(i-1,j)-e; \end{cases} \\ I_y(i,j) &= \max \begin{cases} M(i,j-1)-d, \\ I_y(i,j-1)-e. \end{cases} \end{aligned} \tag{2.16}$$

In these equations, we assume that a deletion will not be followed directly by an insertion. This will be true for the optimal path if $-d-e$ is less than the lowest mismatch score. As previously, we can find the alignment itself using a traceback procedure.

The system defined by equations (2.16) can be described very elegantly by the diagram in Figure 2.9. This shows a state for each of the three matrix values, with transition arrows between states. The transitions each carry a score increment, and the states each specify a $\Delta(i,j)$ pair, which is used to determine the change in indices i and j when that state is entered. The recurrence relation for updating each matrix value can be read directly from the diagram (compare Figure 2.9 with equations (2.16)). The new value for a state variable at (i,j) is the maximum of the scores corresponding to the transitions coming into the state. Each transition score is given by the value of the source state at the offsets specified by the $\Delta(i,j)$ pair of the target state, plus the specified score increment. This type of description corresponds to a *finite state automaton* (FSA) in computer science. An alignment corresponds to a path through the states, with symbols from the underlying pair of sequences being transferred to the alignment according to the $\Delta(i,j)$ values in

the states. An example of a short alignment and corresponding state path through the affine gap model is shown in Figure 2.10.

It is in fact frequent practice to implement an affine gap cost algorithm using only two states, M and I, where I represents the possibility of being in a gapped region. Technically, this is only guaranteed to provide the correct result if the lowest mismatch score is greater than or equal to $-2e$. However, even if there are mismatch scores below $-2e$, the chances of a different alignment are very small. Furthermore, if one does occur it would not matter much, because the alignment differences would be in a very poorly matching gapped region. The recurrence relations for this version are

$$M(i,j) = \max \begin{cases} M(i-1,j-1)+s(x_i,y_j), \\ I(i-1,j-1)+s(x_i,y_j); \end{cases}$$

$$I(i,j) = \max \begin{cases} M(i,j-1)-d, \\ I(i,j-1)-e, \\ M(i-1,j)-d, \\ I(i-1,j)-e. \end{cases}$$

These equations do not correspond to an FSA diagram as described above, because the I state may be used for $\Delta(1,0)$ or $\Delta(0,1)$ steps. There is, however, an alternative FSA formulation in which the $\Delta(i,j)$ values are associated with the transitions, rather than the states. This type of automaton can account for the two-state affine gap algorithm, using extra transitions for the deletion and insertion alternatives. In fact, the standard one-state algorithm for linear gap costs can be expressed as a single-state transition emitting FSA with three transitions corresponding to different $\Delta(i,j)$ values ($\Delta(1,1)$, $\Delta(1,0)$ and $\Delta(0,1)$). For those interested in pursuing the subject, the simpler state-based automata are called *Moore machines* in the computer science literature, and the transition-emitting systems are called *Mealy machines* (see Chapter 9).

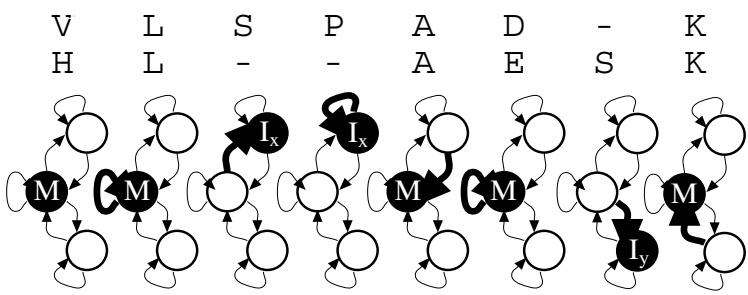

Figure 2.10 *An example of the state assignments for an alignment using the affine gap model.*