6. Multiple sequence alignment methods

In Chapter 5, we assumed that a reasonable multiple sequence alignment was already known and provided the starting point for constructing a profile HMM. We now look at what a reasonable multiple alignment is, and at ways to construct one automatically from unaligned sequences.

Biologists produce high quality multiple sequence alignments by hand using expert knowledge of protein sequence evolution. Importance factors include: specific sorts of columns in alignments, such as highly conserved residues or buried hydrophobic residues; the influence of secondary and tertiary structure; and expected patterns of insertions and deletions. Furthermore, the phylogenetic relationships between sequences dictate constraints on the changes that occur in columns and in the patterns of gaps.

Manual multiple alignment is tedious. Automatic multiple sequence alignment methods are a topic of extensive research in computational biology. In general, an automatic method must have a way to assign a score so that better multiple alignments get better scores. One of our goals in probabilistic modeling is to incorporate as many of an expert's evaluation criteria as possible into our scoring procedure.

6.1 What a multiple alignment means

In a multiple sequence alignment, homologous residues among a set of sequences are aligned together in columns. Homologous is meant in both the structural and evolutionary sense. Ideally, a column of aligned residues occupy similar three-dimensional structural positions and all diverge from a common ancestral residue. However, except for trivial cases of highly identical sequences, it is not possible to unambiguously identify structurally or evolutionarily homologous positions and create a single correct multiple alignment. Since protein structures also evolve, we do not expect two protein structures with different sequences to be entirely superposable. An evolutionary correct alignment can be even more difficult to infer than a structural alignment. While structural alignment has an independent point of reference (superposition of crystal or NMR structures), the evolutionary history of the residues of a sequence family is not independently known from any source; it must itself be inferred from sequence alignment.

Thus, our ability to define a single 'correct' alignment will vary with relatedness of the sequences being aligned. An alignment of very similar sequences will generally be unambiguous, but these alignments are not of great interest to us. For cases of interest, e.g. for a family of proteins sharing perhaps only 30% average pairwise sequence identity, there is no objective way to define an unambiguously correct alignment. We should focus attention on the subset of columns corresponding to key residues and core structural elements that can be aligned with more confidence.

6.2 Scoring a multiple alignment

Our scoring system should take into account at least two important features of multiple alignment: (1) the fact that some positions are more conserved than others, e.g. position-specific scoring; and (2) the fact that the sequences are not independent, but instead are related by a phylogenetic tree. An idealized way to score a multiple alignment would be to specify a complete probabilistic model of molecular sequence evolution. Given the correct phylogenetic tree for the sequences, the probability of a multiple alignment is the

product of the probabilities of all the evolutionary events necessary to produce that alignment via ancestral intermediate sequences times the prior probability of the root ancestral sequence.

Unfortunately, we do not have enough data to parameterize such a complex evolutionary model. Simplifying assumptions must be made. We use approximations that partly or entirely ignore the phylogenetic tree while doing some sort of position-specific scoring of aligning structurally compatible residues.

Many alignment methods assume that the individual columns of an alignment are statistically independent. Such a scoring function can be written as

$$S(m) = G + \sum_i S(m_i) \qquad (6.1)$$

where m_i is column *i* of the multiple alignment *m*, $S(m_i)$ is the score for column *i*, and *G* is a function for scoring the gaps that occur in the alignment. We write *G* as an unspecified function because methods of scoring gaps in multiple alignments differ greatly. The simplest method is to treat a gap symbol as an extra residue type, which then just gives $S(m) = \sum_i S(m_i)$. For simplicity, we will focus in the next several paragraphs on definitions of $S(m_i)$ for scoring a column of aligned residues with no gaps.

Minimum entropy

We now define some notations. As above, *m* is a multiple alignment. Let m_i^j be the symbol in column *i* for sequence *j*. Let c_{ia} be the observed counts for residue *a* in column *i*; $c_{ia} = \sum_j \delta(m_i^j = a)$, where

$$\delta(m_i^j = a) = \begin{cases} 1, \text{ if } m_i^j = a, \\ 0, \text{ otherwise.} \end{cases}$$

Let m_i be the column $m_i^1,...,m_i^N$ of aligned symbols in column *i*, and let c_i be the count vector $c_{i1},...,c_{iK}$ of observed symbols in column *i* for an alphabet of *K* different residues.

If the phylogenetic tree for the sequences has many intermediate ancestors, then the statistical dependence between sequences is complex. The scoring problem is greatly simplified if we assume that sequences have all been generated independently. If we assume that residues within the column are independent, as well as being independent between columns, then the probability of a column m_i is

$$P(m_i) = \prod_a p_{ia}^{c_{ia}}, \qquad (6.2)$$

where p_{ia} is the probability of residue *a* in column *i*. We can define a column score as the negative logarithm of this probability:

$$S(m_i) = -\sum_a c_{ia} \log p_{ia} . \qquad (6.3)$$

This is an entropy measure directly related to the equation for Shannon entropy in information theory. It is a convenient measure of the variability observed in an aligned column of residues. The more variable the column is, the higher the entropy. A

completely conserved column would score 0. We could define a good alignment to be one which minimizes the total entropy of the alignment, e.g. $\sum_i S(m_i)$.

As we have seen before, the parameters p_{ia} can be estimated from counts c_{ia}, for instance,

$$p_{ia} = \frac{c_{ia}}{\sum_{a'} c_{ia'}} . \qquad (6.4)$$

In practice we would normally regularize this probability estimate with pseudocounts. This is near to the HMM formulation of the problem. Profile HMMs go further and also model insertions and deletions. The assumption that the sequences are independent can be reasonable if representative sequences of a sequence family are carefully chosen. It is often the case, though, that the sample of sequences is biased and certain evolutionary sub-families are under- or over-represented. A variety of tree-based weighting schemes have been proposed to deal with this problem.

Sum of pairs: SP scores

The standard method of scoring multiple alignments is not the HMM formulation, but is similar in that it does not use a phylogenetic tree and it assumes statistical independence for the columns. Columns are scored by a 'sum of pairs' (SP) function using a substitution scoring matrix. The SP score for a column is defined as:

$$S(m_i) = \sum_{k<l} s(m_i^k, m_i^l) , \qquad (6.5)$$

where scores $s(a,b)$ come from a substitution scoring matrix such as a PAM or BLOSUM matrix. For simple linear gap costs, gaps are handled by defining $s(a,-)$ and $s(-,a)$ to be the gap cost. Otherwise gaps are scored separately.

Summing all the pairwise substitution scores in the column might seem to be a natural thing to do. However, substitution scores are usually derived as log-odds scores for pairwise comparisons. The correct extension to multiple alignments would be, for instance, $\log(p_{abc} / q_a q_b q_c)$ for a three-way alignment, rather than the SP score $\log(p_{ab} / q_a q_b) + \log(p_{bc} / q_b q_c) + \log(p_{ac} / q_a q_c)$.

6.3 Multidimensional dynamic programming

It is possible to generalize pairwise dynamic programming alignment to the alignment of *N* sequences. However, this turns out to be impractical for more than a few sequences. We assume the columns of an alignment are statistically independent, and we also assume that gaps are scored with linear gap cost. Thus we can calculate the overall score $S(m)$ for an alignment as a sum of the scores $S(m_i)$ for each column i:

$$S(m) = \sum_i S(m_i) . \qquad (6.6)$$

Multidimensional dynamic programming with affine gap costs and multiple states is possible, using methods like those in Chapter 2, but the formalism becomes tedious in many dimensions.

Define $\alpha_{i_1 i_2 \ldots i_N}$ as the maximum score of an alignment up to the subsequences ending with $x_{i_1}^1, x_{i_2}^2, \ldots, x_{i_N}^N$. The dynamic programming algorithm is

$$\alpha_{i_1 i_2 \ldots i_N} = \max \begin{cases} \alpha_{i_1-1,i_2-1,\ldots i_N-1} + S(x^1_{i_1}, x^2_{i_2}, \ldots, x^N_{i_N}), \\ \alpha_{i_1,i_2-1,\ldots i_N-1} + S(-, x^2_{i_2}, \ldots, x^N_{i_N}), \\ \alpha_{i_1-1,i_2,i_3-1\ldots i_N-1} + S(x^1_{i_1}, -, \ldots, x^N_{i_N}), \\ \vdots \\ \alpha_{i_1-1,i_2-1,\ldots i_N} + S(x^1_{i_1}, x^2_{i_2}, \ldots, -), \\ \alpha_{i_1,i_2,\ldots i_N-1} + S(-,-,\ldots, x^N_{i_N}), \\ \vdots \\ \alpha_{i_1,i_2-1,\ldots i_{N-1}-1,i_N} + S(-, x^2_{i_2}, \ldots, -), \\ \vdots \end{cases}$$

where all combinations of gaps appear except the one where all residues are replaced by gaps. There are $2^N - 1$ such combinations. Initialization, termination, and traceback steps for the algorithm follow analogously from the pairwise dynamic programming algorithm.

The algorithm requires the computation of the whole dynamic programming matrix with $L_1 L_2 \ldots L_N$ entries. To calculate each entry we need to maximize over all $2^N - 1$ combinations of gaps in a column. Assuming the sequences are of roughly the same length $\overline{L}$, the memory complexity of the multidimensional dynamic programming algorithm is of the order $\overline{L}^N$ and the time complexity is of order $2^N \overline{L}^N$.

MSA

A clever algorithm for reducing the volume of the multidimensional dynamic programming matrix that needs to be examined was described by some authors. The algorithm was implemented in the multiple alignment program MSA. MSA can optimally align up to five to seven protein sequences of reasonable length (200-300 residues).

The SP scoring system is used in MSA. Let a^{kl} denote the pairwise alignment between sequences k and l. Then the score of the complete alignment is given by

$$S(a) = \sum_{kl} S(a^{kl}).$$

The idea of MSA algorithm is that we only need to consider pairwise alignments implied in the multiple alignment that score better than a lower bound. The detailed discussion is referred to the book and its reference.

6.4 Progressive alignment methods

Probably the most commonly used approach to multiple sequence alignment is progressive alignment. This works by constructing a succession of pairwise alignment. Initially, two sequences are chosen and aligned by standard pairwise alignment; this alignment is fixed. Then, a third sequence is chosen and aligned to the first alignment, and this process is iterated until all sequences have been aligned.

Progressive alignment is heuristic: it does not separate the process of scoring an alignment from the optimization algorithm. It does not directly optimize any global scoring function of alignment correctness. The advantage of progressive alignment is that it is fast and efficient, and in many cases the resulting alignments are reasonable.

The most important heuristic of progressive alignments is to align the most similar pairs of sequences first. These are the most reliable alignments. Most algorithms build a 'guide tree'. This is a binary tree whose leaves represent sequences and whose interior nodes represent alignments. The root node represents a complete multiple alignment. The nodes furthest from the root represent the most similar pairs.

Feng-Doolittle progressive multiple alignment

The Feng-Dolittle algorithm was one of the first progressive alignment algorithms. In overview, it is as follows:

(i) Calculate a diagonal matrix of $N(N-1)/2$ distances between all pairs of N sequences by standard pairwise alignment, converting raw alignment scores to approximate pairwise 'distances'.

(ii) Construct a guide tree from the distance matrix using the clustering algorithm.

(iii) Starting from the first node added to the tree, align the child nodes (which may be two sequences, a sequence and an alignment, or two alignments). Repeat for all other nodes in the order that they were added to the tree (i.e. from most similar pairs to least similar pairs) until all sequences have been aligned.

The method for converting alignment scores to distances does not need to be especially accurate, as the goal is only to create an approximate guide tree. Feng & Doolittle calculate the distance *D* as

$$D = -\log S_{\text{eff}} = -\log \frac{S_{\text{obs}} - S_{\text{rand}}}{S_{\text{max}} - S_{\text{rand}}}, \qquad (6.11)$$

where S_{obs} is the observed pairwise alignment score; S_{max} is the maximum score, the average of the score of aligning either sequence to itself; and S_{rand} is the expected score for aligning two random sequences of the same length and residue composition. The last one, S_{rand}, may either be calculated by random shuffling of the two sequences, or by an approximate calculation given in Feng & Doolittle. The effective score S_{eff} can thus be viewed as a normalized percentage similarity; it is expected to roughly decay exponentially towards zero with increasing evolutionary distance, hence the $-\log$ to make the measure more approximately linear with evolutionary distance.

Sequence-sequence alignments are done with the usual pairwise dynamic programming algorithm. A sequence is added to an existing group by aligning it pairwise to each sequence in the group in turn. The highest scoring pairwise alignment determines how the sequence will be aligned to the group. For aligning a group to a group, all sequence pairs between the two groups are tried; the best pairwise sequence alignment determines the alignment of the two groups. Thus, the scoring system is essentially the standard pairwise score with an affine gap penalty. After an alignment is completed, gap symbols are replaced with a neural X character. The rule allows pairwise sequence alignments to be used to guide the alignment of sequences to groups or groups to groups.

A problem with the Feng-Doolittle approach is that all alignments are determined by pairwise sequence alignments. Once an aligned group has been built up, it is advantageous to use position-specific information from the group's multiple alignment to align a new sequence to it. When we use SP score, and linear gap scoring, profile alignment is relative simple. Let us set $s(-,a) = s(a,-) = -g$ and $s(-,-) = 0$. Assume we have two multiple alignments (or profiles), one containing sequence 1 to *n*, and the other containing sequence $n+1$ to *N*. An alignment of these two profiles means that gaps are inserted in whole columns, so the alignment within one of the profiles is not changed. The score of the global alignment is then

$$\sum_i S(m_i) = \sum_i \sum_{k<l\le N} s(m_i^k, m_i^l)$$
$$= \sum_i \sum_{k<l\le n} s(m_i^k, m_i^l) + \sum_i \sum_{n<k<l\le N} s(m_i^k, m_i^l) + \sum_i \sum_{k\le n, n<l\le N} s(m_i^k, m_i^l).$$

All we did was to split up the sum into two sums only concerning the two profiles and one sum containing all the cross terms. The first two sums are unaffected by the global alignment, because adding columns of gap characters to a profile adds zero to the score ($s(-,-) = 0$). Therefore the optimal alignment of the two profiles can be obtained by only optimizing the last sum with the cross terms. This can be done exactly like a standard pairwise alignment, where columns are scored against columns by adding the pair scores. When one of the profiles consist of a single sequence only, it corresponds to aligning a single sequence to a profile.

One widely used implementation of profile-based progressive multiple alignment is the CLUSTALW program. CLUSTALW works in much the same way as the Feng-Doolittle method except for its carefully tuned use of profile alignment methods. In overview, the CLUSTALW algorithm is as follows:

(i) Construct a distance matrix of all $N(N-1)/2$ pairs by pairwise dynamic programming alignment followed by approximate conversion of similarity scores to evolutionary distances.
(ii) Construct a guide tree by a neighbor-joining clustering algorithm.
(iii) Progressively align at nodes in order of decreasing similarity, using sequence-sequence, sequence-profile, and profile-profile alignment.

Iterative refinement methods

One problem with progressive alignment algorithms is that the subalignments are 'frozen'. That is, once a group of sequences has been aligned, their alignment to each other cannot be changed at a later stage as more data arrive. Iterative refinement algorithms attempt to circumvent this problem.

In iterative refinement, an initial alignment is generated. Then one sequence is taken out and realigned to a profile of the remaining aligned sequences. If a meaningful score is being optimized, this either increases the overall score or results in the same score. Another sequence is chosen and realigned, and so on, until the alignment does not change.

Barton-Sternberg multiple alignment algorithm

(i) Find the two sequences with the highest pairwise similarity and align them using standard pairwise dynamic programming alignment.

(ii) Find the sequence that is most similar to a profile of the alignment of the first two, and align it to the first two by profile-sequence alignment. Repeat until all sequences have been included in the multiple alignment.

(iii) Remove sequence x^1 and realign it to a profile of the other aligned sequences $x^2,...,x^N$ by profile-sequence alignment. Repeat for sequences $x^2,...,x^N$.

(iv) Repeat the previous realignment step a fixed number of times, or until the alignment score converges.

6.5 Multiple alignment by profile HMM training

In Chapter 5 it was shown that sequence profiles could be recast in probabilistic form as profile HMMs. Thus, profile HMMs could simply be used in place of standard profiles in progressive or iterative alignment methods. The advantages of using profile HMM include that the SP scoring scheme can be replaced by the more explicit profile HMM assumption that the sequences are generated independently from a single 'root' probability distribution. Profile HMMs can also be trained from initially unaligned sequences using the Baum-Welch expectation maximization algorithm from Chapter 3.

Multiple alignment with a known profile HMM

Let us consider the simpler problem of obtaining a multiple alignment from a known model. This problem often arises in sequence analysis, for instance when we have a multiple alignment and a model of a small representative set of sequences in a family, and we wish to use that model to align a large number of other members together.

We have seen how to align a sequence to a profile HMM; the most probable path through the model is found by the Viterbi algorithm. Constructing a multiple alignment just requires calculating a Viterbi alignment for each individual sequence. Residues aligned to the same profile HMM match state are aligned in columns.

Figure 6.4 shows a small profile HMM and the multiple alignment it was derived from. The shaded residues were arbitrarily defined to be insertions for the purposes of this example, and the other ten columns correspond to ten profile HMM match states. The same seven sequences were realigned to the model, giving the optimal Viterbi paths shown in Figure 6.5. These paths result in the multiple alignment shown in Figure 6.6, left, where lower-case residues were assigned to an insert state and upper-case residues were assigned to a match state. The original alignment and the new alignment are the same alignment. A profile HMM does not attempt to align the low-case residues assigned to insert states. The choice of how to put the insert residues in the alignment is arbitrary; some profile HMM implementations simply left-justify insert regions, as shown in Figure 6.6. The insert state residues usually represent parts of the sequences which are atypical, unconserved, and not meaningfully alignable. This is a biologically realistic view of multiple alignment. For instance, we expect loops of homologous protein structures often to be structurally different and unalignable.

The alignment on the right in Figure 6.6 shows a new sequence aligned to the same model. This sequence has more inserted residues than any of the other seven sequences in the shaded area assigned to insert state 6, so the alignment of the other seven sequences must be adjusted to allow space for these two new residues. In an implementation, we typically look at all the Viterbi paths and find the maximum number

of inserted residues for each insert state before building the multiple alignment, so we know up front how much room we need to leave to accommodate insertions.

Overview of profile HMM training from unaligned sequences

It is a harder problem of estimating both a model and a multiple alignment from initially unaligned sequences. The method is summarized as follows:

Initialization: Choose the length of the profile HMM and initialize parameters
Training: Estimate the model using the Baum-Welch algorithm or the Viterbi alternative. It is usually necessary to use a heuristic method for avoiding local optima.
Multiple alignment: Align all sequences to the final model using the Viterbi algorithm and build a multiple alignment as described in the previous section.

Initial model

A profile HMM is a repeating linear structure of three states (match, delete, and insert). The only decision that must be made in choosing an initial architecture for Baum-Welch estimation is the length of the model M. Here M is the number of match states in the profile HMM rather than the total number of states. A commonly used rule is to set M to be the average length of the training sequences (or to set it based on prior knowledge).

Since Baum-Welch estimation finds local optima, not global, it is important to choose initial models carefully. The model should be encouraged to use 'sensible' transitions; for instance, transitions into match states should be large compared to other transition probabilities. At the same time, we want to start Baum-Welch from multiple different points to see if all converge to approximately the same optimum, so we want some randomness in the choice of initial model parameters.

One approach is to sample the model's initial parameters from a prior distribution over parameters. Alternatively, we can initialize the model with frequencies derived from the prior. A further possibility is to estimate the initial model by model construction from an existing guess at the multiple alignment of some of the sequences.

Baum-Welch expectation maximization

The basic parameter estimation is done by a straightforward application of the Baum-Welch algorithm from Chapter 3. We consider the multiple sequences as the independent training sequences, and use forward algorithm and backward algorithm to calculate the forward value and backward value respectively. The expected emission counts and transition counts are estimated from these values, and model parameters are re-estimated. The detailed algorithm is left to students as an exercise.

Avoid local maxima

The Baum-Welch algorithm is guaranteed to find a local maximum on the probability 'surface' but there is no guarantee that this local optimum is anywhere near the global optimum nor a biologically reasonable solution. Much the same is true for any practical score optimizing multiple alignment method (multidimensional dynamic programming finds global optima but is not practical). Part of the reason is that these models are usually quite long, and thus there are many opportunities to get stuck in a wrong solution. For instance, two variations of the same conserved motif may end up being modeled as

two different motifs or a conserved region is squeezed in between two other regions and ends up as being modeled as an insert. One way to search the parameter space is simply to start again many times from different (random) initial models and keep the best scoring final one.

A more involved approach is to use some form of stochastic search algorithm that 'bumps' Baum-Welch off from local maxima. The most common stochastic algorithm is simulated annealing.

Theoretical basis of simulated annealing

Some compounds only crystallize if they are slowly annealed from high temperature to low temperature. If the temperature is lowered too fast the structure ends up in a local free energy minimum and is disordered. In an optimization problem we have some function to minimize, which we can call the 'energy' $E(x)$, where x represents all the variables in which it has to be minimized. (Maximizing a function is identical to minimizing the negative value of the function). Inspired by the physics example, one can introduce an artificial 'temperature' T, and by the laws of statistical physics the probability of a configuration (or 'state') x is given by the Gibbs distribution

$$P(x) = \frac{1}{Z}\exp\left(-\frac{1}{T}E(x)\right). \qquad (6.12)$$

The normalizing term $Z = \int \exp\left(-\frac{1}{T}E(x)\right)dx$ is called the partition function in statistical physics. Since x is usually multidimensional, this is a complicated integral and often it is impossible to calculate Z.

In the limit of $T \to 0$, all configurations except the ones with the lowest energy have probability 0 (the system is 'frozen'). In the limit of $T \to \infty$, all configurations are equiprobable (the system is 'molten'). By analogy with crystallization, the minimum (or minima) can be found by sampling this probability distribution at high temperature first, and then at gradually decreasing temperatures. This is called simulated annealing. In other applications, not considered here, simulated annealing is usually done by so-called Monte Carlo methods.

For an HMM, a natural energy function is the negative logarithm of the likelihood, $-\log P(\text{data} \mid \theta)$, so the probability (6.12) is

$$\frac{\exp\left(-\frac{1}{T}[-\log P(\text{data} \mid \theta)]\right)}{Z} = \frac{P(\text{data} \mid \theta)^{1/T}}{Z} = \frac{P(\text{data} \mid \theta)^{1/T}}{\int P(\text{data} \mid \theta')^{1/T}\, d\theta'} \qquad (6.13)$$

To pick up a model from this distribution appears distinctly non-trivial. The two methods we mention below are approximations.

Noise injection during Baum-Welch HMM re-estimation

The important point of simulated annealing is that it is possible to escape local minima because of the stochastic choice of configuration (as opposed to algorithms that seek to always lower the energy). A similar effect can be obtained by adding noise to the counts estimated in the forward-backward procedure, and to let the size of this noise decrease

slowly, just as the temperature decreases in simulated annealing. Some researchers generate the noise by a random walk in the initial model.

Simulated annealing Viterbi estimation of HMMs

Another approach is to obtain a model by a simulated annealing variant of the Viterbi approximation to Baum-Welch estimation. Recall that in Viterbi estimation the most probable path for each sequence is used to obtain the counts from which the new model is estimated, rather than summing over all paths to obtain expectations for the counts. If there are *N* sequences, there is an exact translation from the *N* paths $\pi^1,...,\pi^N$ to the parameters of the model. Therefore, we can treat the paths as the fundamental parameters in which to maximize the likelihood, so the simulated annealing can be done in these (discrete) variables instead of the (continuous) model parameters, θ.

The key difference between Viterbi estimation and the simulated annealing variant is that while Viterbi selects the highest probability path π for each sequence *x*, simulated annealing samples each path π according to the likelihood of the path given the current models as modified by a temperature *T*:

$$\text{Prob}(\pi) = \frac{P(\pi, x \mid \theta)^{1/T}}{\sum_{\pi'} P(\pi', x \mid \theta)^{1/T}}$$

The denominator is *Z*, the partition function. However, it is just a sum over all paths and therefore can be obtained by a modified forward algorithm using exponentiated transition and emission parameters. Exponentiated parameters are pre-calculated for computational efficiency: $\hat{a}_{ij} = a_{ij}^{1/T}$, and $\hat{e}_j(x) = e_j(x)^{1/T}$ and used in place of the unmodified probability parameters in the forward calculation. The partition function *Z* is then the result of the forward algorithm, which would be $P(x)$ when the unexponentiated parameters are used.

A suboptimal path π is then selected from the forward dynamic programming by a stochastic traceback. The alignment consists of a series of states π_i which are recursively chosen with a probability determined from the forward variable.

Stochastic sampling traceback algorithm for HMMs

Initialization: π_{L+1} = End.

Recursion: for $L+1 \geq i \geq 1$,

$$\text{Prob}(\pi_{i-1} \mid \pi_i) = f_{i-1,\pi_{i-1}} \hat{a}_{\pi_{i-1}\pi_i} / \left(\sum_k f_{i-1,k} \hat{a}_{k,\pi_i} \right).$$

In other words, for each state reached in the traceback, a previous state is chosen based on its share of the path sum probability coming into the current state.

This suboptimal alignment algorithm is then used to implement a simulated annealing variant of Viterbi training. Instead of determining an optimal multiple alignment with respect to the current model at each step of each iteration, a suboptimal multiple alignment is sampled. The degree of the suboptimality is controlled by a temperature factor *T*, which is started high and slowly reduced.

Finding the best 'schedule' for how fast to lower the temperature is a whole science in itself. There is a theoretical result for simulated annealing saying that if the

temperature is lowered slowly enough, finding the optimum is guaranteed, but the time required for this is prohibitive. In practice a simple exponentially or linearly decreasing temperature schedule is often used, where each step amounts to either multiplying T by some number less than 1 or to reducing it by some small constant amount.

Comparison to Gibbs sampling

We will introduce the 'Gibbs sampler' algorithm described by Lawrence et al. (1993) in the following motif alignment. The statistical model is a short ungapped motif model which is essentially a profile HMM with no insert or delete states. The training data consist of a set of sequences which contain exactly one instance of some motif, such as a specific protein-binding site on DNA, where the position of the motif is initially unknown. The problem is to simultaneously find the motif positions and to estimate the parameters for a consensus statistical model of them. In fact by knowing one, we can find the other, just as an alignment implies an HMM and vice versa. It is a natural problem for expectation maximization (EM), where the missing data are the positions of the motif, that can simply be specified by their start points in the sequences; these correspond to the alignments which are the missing data we trying to infer in HMM training. Indeed, earlier algorithms applied EM to the problem, but these approaches proved prone to poor local optima.

In an HMM framework, both the above simulated annealing algorithm and the Gibbs sampler are stochastic sampling variants of the Viterbi approximation of EM. At each iteration of Gibbs sampling, a sequence is removed from the alignment; an HMM is built of the remaining aligned sequences; and then a new alignment of the sequence to the rest is sampled probabilistically using a stochastic sampling algorithm. This iteration is repeated until the model reaches a region of high probability.

The last issue of profile HMM alignment algorithm is to modify model architecture after training a model. We will skip this content, but instead review the Gibbs sampling algorithm.

When we have a probabilistic model of many variables, it may often be possible to sample from the distribution obtained by keeping all variables fixed except one, i.e. the conditional distribution. Gibbs sampling exploits this idea. It works by choosing points from the conditional distribution $P(x_i \mid x_1,...,x_{i-1},x_{i+1},...,x_N)$ for each i, cycling repeatedly through $i = 1,...,N.$ Under some conditions, Gibbs sampling will inevitably converge to the probability $P(x_1,...x_N).$ However, Gibbs sampling can get stuck in the case that there are two pieces of density which do not overlap along any of the coordinate directions. If there were even a small overlap, the transition between regions is infrequently. Therefore, Gibbs sampling is hard to converge.