2. PAjHMMA – Parameter Adjustable Java Hidden Markov Model Architecture

2.1. Introduction

In this chapter, we present a flexible software framework, PAjHMMA, for detecting signals in nucleotide sequences. The resulting model is based on the observation that regions of different biological function can have constrained sub-sequences or nucleotide distributions. By varying the parameters in the model, it is possible to model many different encoded biological signals. The technique used allows us to model both long-range, diffuse sequence information, as well as exact or stringent matches to well-characterised sequence motifs. The flexibility of the framework is provided by the separation of the model from the algorithms used to search for model hits. For this reason, it is possible to search sequences for different biological motifs quickly. As the model file is provided in a very simple syntax, a model can be changed or developed afresh with no change required to the decoding software. PAjHMMMA is available for download from http://www.sanger.ac.uk/Software/analysis/pajhmma.

2.2. An overview of hidden Markov models (HMMs)

2.2.1. Background

As discussed in chapter 1, biologically meaningful signals are encoded within biological sequences as discrete regions of given function, which can range from being a local exact sequence match (such as an in-frame STOP codon), to much more

diffuse, poorly-defined motifs, such as eukaryotic promoters (Down et al. 2002). Hidden Markov models have been used successfully to detect signals of varying strengths and combine them together, such as in assigning proteins to various families based on the position and order of protein domains (Bateman et al. 2004), and in gene finding (Burge et al. 1997; Zhang 2002).

2.2.2. Hidden Markov models

An HMM is a statistical model, which has been used in diverse fields in which information occurs in sequences of discrete emissions. Examples of such uses include speech recognition (Rabiner 1989), music recognition (Raphael 1999), gene finding (Burge et al. 1997), and in other biological sequence analysis (Durbin et al. 1998). The model has a finite number of states, each of which has a distinctive frequency distribution over the 'alphabet' of possible emissions. The states are connected to one another by a set of probabilities. A state can be analogous to some kind of functional or characterised feature, such as a season of the year, which has a distinctive emission frequency for particular types of weather.

We can think of an HMM as a machine generating a sequence left to right according to an underlying state path that is hidden from us. The machine has a finite number of interconnected states, and a fixed alphabet. Hence the machine is constrained to emit symbols existing only in that alphabet, and at the frequencies prescribed by the current state. At each stage, the model emits a nucleotide according to the nucleotide emission characteristics of the model's current state. It also moves into a new state (possibly the same one) according to a state-state transition probability.

The HMM in Figure 3 shows a model that can generate a sequence of **a** and **b** emissions. Each of the two states has a characteristic emission. The transition probabilities between the states dictate the order and lengths of the states, and also the overall topology of a pass through the model. An example of a model-directed topology is shown; a pass through the model must end in state B. An HMM is a natural method to model DNA sequence, as regions of different functions typically have different nucleotide frequencies, and can have functionally constrained positions relative to each other. The simple model in Figure 3 models two different states with different emission characteristics, typically encourages state A to emit fewer emissions than state B, and constrains the model such that a state sequence can swap between A and B as many times as it likes, but can only terminate when in the B state.

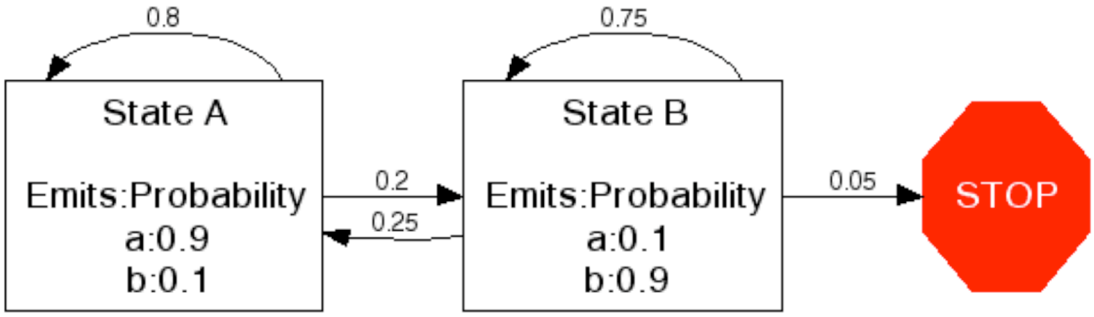

Figure 3. A diagram of an HMM that generates a sequence of a and b emissions. Note that states A and B have different emission frequencies for possible emissions a and b. State A has a tendency to emit a, and B a tendency to emit b. States A and B may transition into themselves or into each other with the given probabilities. State B can transition into an end state. Emission probabilities dictate the constitution of each state.

2.2.3. HMMs for prediction

Rather than using the model as a sequence generator, it is possible to score a given sequence according to all possible state paths through the model. We scan the sequence, and place each emission in one of the states. We score an observed **a** with

0.99 if we decide to put it in state A and 0.01 if it is to be put in state B. The opposite scoring scheme is used if we encounter a **b**. At any point, the model must either transition to a different state, or elect to stay in the same one. This transition has a characteristic cost, depending on which of the possible transitions occurs. We note the state into which each emission can be placed, trying to maximise the score at all times. The state path that scores the highest, represents a demarcation of the sequence into partitions of different emission frequencies. The highest scoring state path, for a sequence

aaaaaabbbbbbaaaaaabbbbbb

would be

AAAAAABBBBBBAAAAAABBBBBB

This state path would also be the highest scoring for a sequence

aabaaabbbabbaaabaabbabbb

but not for

aabbaabbaabbaabbaabbaabb

for which the highest scoring state path would be

AABBAABBAABBAABBAABBAABB

The difference in the two highest scoring state paths is that in the final example, it is deemed preferable to transition into a different state rather than accommodate a suboptimal emission, which is tolerated in the previous sequence.

Given a sequence, therefore, the inference of this state path represents a predicted annotation for the sequence. If this is DNA sequence, and the states score

nucleotides with emission probabilities characteristic of exons, splice sites, and introns etc., then the highest scoring state path represents a gene prediction.

Each sequence emission in any given state is given a transition and emission score which is equal to the probability of that transition and emission. The score of a sequence path through an HMM represents a joint probability distribution over state paths and sequences. Given a sequence, it is possible to find the most likely state path using the Viterbi algorithm (Viterbi 1967). Rather than evaluate every sequence element - nucleotides in the case of a DNA sequence - in every possible state, which would be a brute-force approach, we can use dynamic programming to search the state-space more efficiently. The Viterbi algorithm accomplishes this by a left–to–right sweep to establish partial scores of matching a sequence prefix, given it ends in some state. At any time during the forward sweep, only the best score so far ending in each state is stored. This is followed by a trace back from right to left, extracting the most likely state path. This means that sub-optimal solutions are not evaluated, and thus the total search time actually used is much smaller than that would be used by a brute force method.

Additional information available from using an HMM comes from the ability to calculate the probability that nucleotide n was generated by a state k, using the forward and backward algorithms, which will be covered later. Later in this chapter, I give the equations used to implement these algorithms.

2.2.4. HMMs for gene finding

We can partition DNA sequence into exons, introns, intergenic sequence etc. These have different nucleotide properties on account of their different biological

functions. As a result, HMMs have been used successfully as gene finders. In a given genome sequence, the state path of biologically functional regions is hidden in emissions made up of nucleotides, dinucleotides, or higher order emissions, such as coding triplets. Using sets of manually annotated genes, it is possible to build states representing each biological entity, such as an exon, by finding its characteristic emissions. These states are then connected to each other in a biologically meaningful manner. By this, we mean that the design of the model must obey the rules of biology as we understand them. Hence, a reasonable gene prediction must not contain in-frame stop codons, nor can a model pass in and out of an intron without flanking it with donor and acceptor splice sites.

HMMs are ideal constructs for modelling 3' UTRs and polyadenylation signals, because the region to be modelled can be partitioned into functional areas of distinctive nucleotide properties, as we shall observe in chapter 3.

Polyadenylation signal prediction can therefore be viewed as a logical extension to gene prediction, as virtually all protein coding genes have a 3' UTR and polyadenylation signal.

2.2.5. Length issues

In a simple HMM, when one state accounts for several bases, it does so by having some transition back to itself, and some into the next state. The more sequence is emitted from this state, the more times the transition back to itself must have been chosen. If the probability of transition to itself is p, then the transition of leaving is $(1-p)$. Disregarding emission probabilities, the probability of remaining in a state for m nucleotides

$$P(m\ residues) = (1 - p)p^{m-1}.$$

This leads to the length of sequences emitted by the state being distributed according to the geometric distribution; they decay exponentially. Although certain biological sequence lengths, such as 3' UTRs, can be approximated using this distribution, HMMs are weaker at modelling features with distinctive non-geometric length distributions. There are various strategies allowing a combination of geometric-length states to model a non-geometric phenomenon (Durbin et al. 1998), but a PAjHMMA model allows the user to specify an explicit length distribution for a given state when required. We implement this using a *generalised* HMM.

2.2.5.1. Generalised HMM

In a generalised HMM a state can emit a region of sequences in one step, rather than one nucleotide at a time. This means that we can specify the length distributions for sub-sequences emitted by given states, so we can model states with non-geometric length distribution shapes more accurately. For example, this technique can be used to impose a minimum length on introns, which is appropriate, as these have a biologically constrained minimum length.

The algorithm is extended so that when it transitions into a state with an explicit length distribution, the whole sequence region is emitted and scored, according to the provided length distribution. This means that the number of emissions coming from a particular state is influenced by an observed length distribution rather than a transition probability.

2.2.6. Model training

To build an accurate HMM, it is necessary to determine emission and transition probabilities that closely reflect the biological feature being modelled. As we shall discover in chapter 3, polyadenylation signals can be found experimentally for a small number of genes. For some species, this number is big enough to find emission probabilities by counting nucleotide frequencies. Transition probabilities can also be found, by counting occurrences of an annotated transition event. Manual training is also possible, by calculating a transition probability to approximate an observed length distribution.

Some state sequences correspond to a pass through a weight matrix. There is one state per weight matrix column. This means that each of these states emits exactly one nucleotide, so the transition probability from one state to another is always set to 1.

2.3. Software design

To annotate a sequence against an HMM, the user provides a sequence in FASTA format and a generalised HMM with topology and parameters in a file described below. We describe the format of the model and the pre-processing of the sequence, before describing the dynamic programming algorithm.

2.3.1. Objects

The code is written in the Java programming language. This was partly due to the availability of the BioJava project (http://www.biojava.org/), which provides easy access to software libraries for computational biology. Of interest to this project were classes used for handling FASTA files and DNA sequence utilities.

Having access to objects allows an intuitive setting and access of parameters for the model. We declare a GeneralisedHMM object, which gives us access to a number of States, connected by transition probabilities. Each State is also an object, having methods to return its characteristic nucleotide frequencies, and length distribution, if it has one.

After a few initialisation steps, dynamic programming is carried out by first principles. Sequences are parsed using BioJava utilities and converted into streams of integers. Transition probabilities and state emission frequencies are stored as elements in two dimensional arrays. Modelling the DNA sequence as a sequence of integers means that for each emission, emission and transition scores can easily be looked up by using array indices. Calculated scores are stored in and looked up from a dynamic programming matrix, which is also a 2D array.

2.3.2. Model

The model to be used for sequence decoding is provided as a simple tab-delimited text file. It consists of an HMM declaration, followed by a list of states. Each state consists of a state declaration, followed by the transition, emission, and (optionally) length parameters. Once the file is parsed, a GeneralisedHMM object and a collection of State objects is constructed. This allows the formation of lightweight 2D double arrays containing emission and transition frequencies. As we will be converting the DNA sequence into numbers (ints), this means that all lookups in the

dynamic programming matrix fill stage will be array lookups, rather than hash lookups, which are slower.

2.3.2.1. The HMM declaration

```
HMM C.elegans-3'end 14
```

This declaration informs the constructor of the GeneralisedHMM object how many states there are. Although this information is redundant, it is useful to instruct the program how much memory to allocate. Java memory allocation is automatic, but pre-specifying the number of states allows the use of arrays. These are used in preference to ArrayLists, as the latter's gain in flexibility comes at a price of poorer performance.

2.3.2.2. The State declaration

```
State UTR 1    0   2   0   0
Transition UTR  0.99
Transition A1   0.01
Emissions  0.270   0.198   0.127   0.405
EndState
.
.
.
State SP 0 0   1   0   30
Transition C1  1.0
Emissions  0.271   0.138   0.137   0.454
Length /Users/ashwin/models/length_distribution.txt
EndState
```

The state declaration contains the name of the state, and five numbers. The first two are flags with 0/1 values for being the initial and terminal states. The third

value gives the number of states to which this state has legitimate transitions. The fourth figure gives the order of the state, though mixed order models are not currently implemented.

The ability to model non-zero order (dinucleotides, trinucleotides, etc) emissions is important, as certain features are either not coded in mononucleotides (Gardiner-Garden et al. 1987), or better modelled using a higher order alphabet (Salzberg et al. 1999). Building higher order models requires more data than zero order models, on account of the need to avoid overfitting.

The last number in the state declaration is zero for geometric states, but when a length distribution is explicitly specified, then this value is equal to the length of the length distribution. As with the HMM declaration, this up-front declaration allows more efficient parsing of the model file into a Java object.

2.3.2.3. The state specification

For each state, there is a list of legal transitions and their probabilities. All other transitions are set to zero. There then follows a list of emission frequencies for each nucleotide, given in alphabetical order. If the *StateOrder* is the order of the emissions from a state, the number of emissions expected is $4^{StateOrder+1}$, so with a first order model, 16 dinucleotide frequencies should be given.

In the example above, the SP state has a length distribution specified explicitly. The length declaration is the path to a file containing tab-delimited text in the form of a length and frequency or count. The distribution counts are normalised to add up to one.

2.3.2.4. Model attributes

Once the model file has been parsed, two 2D matrices are created. One is the transition matrix, and the other is the emission matrix. The transition matrix has dimensions to contain transition probabilities from each state to every other state in the model, including itself. Transitions disallowed by the model topology are set to zero. All values are stored as log-probabilities for arithmetic reasons (discussed further below).

The emission matrix contains the emission probabilities for each nucleotide (dinucleotide etc.) within each state. These are also stored as log probabilities.

2.3.3. Sequence pre-processing

The DNA sequence to be annotated is converted to an array of integers depending on the order of the model. For a zero order model, this array contains values 0-3, corresponding to A, C, G, and T. There are 16 values if the model is first order. This pre-processing allows us to avoid the use of hash lookups in the dynamic programming loops, in favour of array lookups, which are faster. The conversion of the sequence into a numerical form means that if we refer to a state by a number, given a model and a sequence, we can call a particular emission probability from the emission matrix by giving the state number and the nucleotide number as a lookup from the 2-dimensional array.

2.3.4. Dynamic programming algorithms

The three algorithms commonly used in annotating a sequence with states from an HMM are well documented (Durbin et al. 1998). As the HMM used here is generalised – states are allowed to have an explicitly specified length – modifications are needed to the Viterbi, forward, and backward algorithms when finding maximal or sum evaluations in a state with an explicit length.

Given a sequence of length L having emission x at position i., we require a dynamic programming matrix to store the evaluation of sequence emissions in each state. For each position in the sequence, this matrix, $v_k(i)$, stores the probability of the highest scoring path ending with the i-th nucleotide in state k.

2.3.4.1. Viterbi algorithm for geometric states

The standard Viterbi algorithm used for states with geometric length distributions is reproduced with slight alteration below from (Durbin et al. 1998).

$$\text{Initialisation } (i = 0): \quad v_0(0) = 1, v_k(0) = 0 \ \textit{for } k > 0.$$

Then for all nucleotides in a given state l, where the previous nucleotide was in state k, a score v_l is calculated. There are two components to this score. The first is the emission score $e_l(x_i)$ of nucleotide x at position i in state l. This is constant regardless of the value of the previous state k. The second component is the maximum value found by evaluating, for all values of k, the product of the previous maximal score at the previous nucleotide ($v_k(i-1)$) and the transition probability a_{kl} from state k to l. Only storing the maximal of all previous values means that at each extension by one nucleotide, the extension is being carried out only on the optimal prefix, rather than on all prefixes. It is for this reason that the Viterbi algorithm finds the optimal

path, termed π*, much faster than a brute-force evaluation. The two components are multiplied together to give the score v_l at nucleotide position *i*. To keep track of which state transitions were occurred at which positions in the optimum path π*, the value at each *l*, of the optimal previous state, $\arg\max_k(v_k(i-1)a_{kl})$, is stored in an array of pointers.

$$\text{Recursion } (i = 1...L): \quad v_l(i) = e_l(x_i)\max_k(v_k(i-1)a_{kl});$$
$$pointer_i(l) = \arg\max_k(v_k(i-1)a_{kl}).$$

$$\text{Termination}: \quad P(x,\pi^*) = \max_k(v_k(L)a_{k0});$$
$$\pi_L^* = \arg\max_k(v_k(L)a_{k0}).$$

Then by tracking backwards through the sequence, we know at each nucleotide, which state the preceding nucleotide was in for an optimal scoring path, so following this pointer through the whole sequence will give the state annotation that scores highest, given each state's characteristic nucleotide emission and the transition probabilities between the states.

$$\text{Traceback } (i = L...1): \pi_{i-1}^* = pointer_i(\pi_i^*).$$

For a zero order model, an emission equates to one nucleotide, but for higher order models, the sequence must be tokenised into higher order emissions, such as hexamers, and the emission scores trained on appropriate measurements.

2.3.4.2. Posterior decoding

The Viterbi algorithm detailed above gives the most probable state path through the sequence. However, for modelling a biological system, it is often more appropriate to find sub-optimal matches to a model. An example of this is in splice site analysis, in which it has been shown that an exact match to a consensus causes reduction of splicing activity on account of the U1 snRNA binding too strongly to its binding site (Lund et al. 2002). The most probable path is also not particularly informative if there are many high probability paths with very similar probabilities. In these circumstances, it may be better to capture every occurrence of a sequence being in a particular state above a threshold probability, so it is informative to know the probability of being in a particular state at a given nucleotide. This is the posterior probability, and requires us to calculate the forward and backward probabilities $f_k(i)$ and $b_k(i)$, which are explained below.

The posterior probability of a particular nucleotide at position *i* in a particular state *k* is calculated as

$$P(\pi_i = k \mid x) = \frac{f_k(i)b_k(i)}{P(x)},$$

which is the probability of the observed sequence *x* over all paths up to nucleotide *i* in state *k* (the forward probability) multiplied by the probability of the observed sequence in all paths following nucleotide *i* being in state *k* (backward probability), divided by the probability of the sequence. To work out these values, we use the forward and backward algorithms.

2.3.4.3. The forward algorithm

The forward algorithm calculates the probability of all possible paths through the sequence instead of the most likely. It is similar to the Viterbi algorithm, but for a particular position in a particular state, it evaluates the sum of all the possible paths leading up to the one in question, rather than finding the maximum. The initialisation is the same, but the recursion and termination stages of the forward algorithm are therefore summations, not evaluations of the maximum. The probabilities, up to and including point *i,* of all paths putting nucleotide *i* in state *k* are stored in a forward matrix ($f_k(i)$).

$$\text{Recursion } (i = 1...L): \quad f_l(i) = e_l(x_i)\sum_k f_k(i-1)a_{kl};$$

$$\text{Termination}: \quad P(x) = \sum_k f_k(L)a_{k0};$$

2.3.4.4. The backward algorithm

The backward algorithm $b_k(i)$ calculates the probability of the sequence following *i*, given that nucleotide *i* was put in state *k*. This is like the forward algorithm, but has a backward recursion though the sequence, and for a given nucleotide, score sums are evaluated over which state the next nucleotide can be put.

$$\text{Initialisation } (i = L): \quad b_k(L) = a_{k0} \text{ for all } k.$$

$$\text{Recursion } (i = L-1,...,1): \quad b_k(i) = \sum_l a_{kl}e_l(x_{i+1})b_l(i+1);$$

$$\text{Termination}: \quad P(x) = \sum_l a_{0l} e_l(x_1) b_l(1).$$

2.3.5. Modifications required for decoding explicit length states

The length duration of an HMM state is usually implied by the transition probability for leaving that state. For a state with out-transition probability P, the probability of remaining in that state after N transitions is $(1\text{-}P)^N$. The length of a sequence that is modelled this way is thus geometrically distributed with mean 1/(1-P). Geometric length distributions are reasonable approximations for certain biological sequences, such as eukaryotic intergenic regions (Burge et al. 1997), but not for others, such as the region between the *C. elegans* polyadenylation signal and the cleavage site, as we shall show in chapter 3.

States with an explicit length distribution add a complication to the three dynamic programming algorithms discussed above. One way to deal with them is to have different out-transition probabilities dependent on how many emissions have been made from that state. However, no length information is stored by the Viterbi algorithm, and thus if a preceding state has a non-geometric length distribution, it is necessary to evaluate all possible lengths from that state separately.

For a given state transition being evaluated in PAjHMMA, the dynamic programming loop has a switch that asks whether the next emission/set of emissions should be scored in an explicit duration state or a geometric one. This information is stored in each state of the model.

2.3.5.1. Viterbi algorithm for explicit length states

According to the Viterbi algorithm, for each nucleotide, it is necessary to find the state that scores it and its prefixed state path maximally. Now in addition, each time an evaluation is made in a non-geometric state, it is necessary to evaluate (and maximise over) not just a single emission in that state, but a compound score calculated over all possible sequences of emissions with that starting point. There are two components to the extra information required in explicit length states. For a state length distribution *d* with *D* elements, the first is an emission score for the whole sequence being evaluated within the state, and a length score based on the frequency of that length in the distribution (Figure 4).

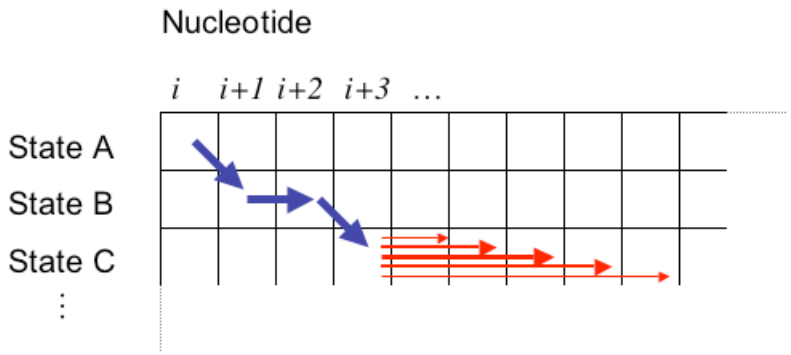

Figure 4. A diagram of a dynamic programming matrix showing one of many possible paths. States A and B are normal states having geometric length distributions. The maximal scoring path at any stage is maximised over which state the next nucleotide should be put in. State C has an explicit length distribution, shown with red arrows. Certain lengths in this distribution are favoured. Scores are found for all possible lengths of sequence in this state. The maximal scoring path is maximised over the combination of (a) the emission score of all these lengths and (b) a scaling factor according to frequency of each length in the length distribution.

All possible lengths within the length distribution are evaluated. At any particular sampling length *m,* the score of the maximal prefix to this sequence is found by looking back *m* nucleotides. The length score is *d(m).* If *l* is a state with a specified length distribution,

$$v_l(i) = e_l(x_i)\max_{\substack{k;\\ m\in\{1,\ldots,D\}}} v_k(i-m)a_{kl}d(m)\prod_{q=1}^{m} e_l(x_{i-q}).$$

Each time the algorithm attempts a transition into an explicit length state, the pointer containing the optimal source state is kept as before, but it also stores the length of sequence causing the maximum score. This value is required in the traceback procedure.

2.3.5.2. Posterior decoding with explicit length states

As with geometric length states, posterior decoding requires the forward and backward probabilities. The forward algorithm for explicit length state *l* is once again similar to the Viterbi; it is simply the sum of all the terms from which the maximum was stored previously.

$$\text{Forward}: \quad f_l(i) = e_l(x_i)\sum_{\substack{k;\\ m=1}}^{m=D} f_k(i-m)a_{kl}d(m)\prod_{q=1}^{m} e_l(x_{i-q})$$

The backward algorithm differs from the non-explicit length version simply by having extra terms for the emissions from the *m* residues being scored in state *l*, which

is $e_l(x_{i+m})$, the sum of backward scores up to that point $b_l(i+m)$, and the length score $d(m)$.

$$\text{Backward}: \quad b_k(i) = \sum_{\substack{l; \\ m=1}}^{m=D} b_l(i+m)a_{kl}d(m)\prod_{q=1}^{m} e_l(x_{i+q});$$

2.3.6. Preventing numerical underflow

All the algorithms given above feature the multiplication of probabilities. In particular, in the Viterbi and forward algorithms, the maximum score at nucleotide *i* is a product of all the previous maximum scores. The minimum value of a Java `double` is 2^{-1074}. A straightforward implementation of even a simple HMM would therefore underflow (depending on parameters) within a few thousand nucleotides at the most. A joint sequence probability with a mean probability per nucleotide of 0.5 would underflow after 1074 residues. To prevent such problems, the standard solution is to work in log space. This turns all of the multiplications into sums. In the standard Viterbi algorithm, as the score value is just the product of all the score components, using logs is simple enough.

$$\ln v_l(i) = \ln e_l(x_i) + \max_k(\ln v_k(i-1) + \ln a_{kl}).$$

In the forward and backward algorithm, an added complication is that calculated path scores have to be summed, so these log values need to be re-exponentiated before they are summed, and the logarithm found again. Hence for the forward algorithm.

$$f_l(i) = e_l(x_i)\sum_k f_k(i-1)a_{kl}$$

$$\ln f_l(i) = \ln e_l(x_i) + \ln \sum_k f_k(i-1)a_{kl}$$

$$= \ln e_l(x_i) + \ln(\exp(n_1) + \exp(n_2) + ... + \exp(n_k)),$$

$$where\ n_k = \ln f_k(i-1) + \ln a_{kl}.$$

However, these exponentiations are likely to underflow, preventing an accurate summation, so we instead rearrange, using the following observation:

$$\ln \sum_{x=a}^{b} \exp(x) = \ln\left(\exp(q)\sum_{x=a}^{b}\frac{\exp(x)}{\exp(q)}\right)$$

$$= q + \ln \sum_{x=a}^{b} \exp(x-q)$$

If we choose the value of the scaling factor q to be the smallest of all the exponents (n_1,n_2…n_k), then the smallest exponent becomes 0.

2.3.7. Methods of usage

PAjHMMA has two principal output forms; one is a traceback through the pointers matrix, which annotates each nucleotide to a state according to the most probable path through the model, with the state boundaries and sequence being printed. Alternatively, the dynamic programming algorithm makes the posterior decoding matrix available to the user. It is thus possible to list the probability of the sequence being in any given state along its length. For a given sequence, all paths

having probability greater than some threshold in some state can be output. The use of this can be seen in Chapter 3.

2.4. Conclusion

In this chapter, I have presented a flexible framework for building an HMM for nucleotide sequences. The resulting HMM is based on a series of interconnected states of different possible types. It can be used to annotate a sequence according to its most probable state path through the model, or the posterior probability of each base matching a particular state. This software is used throughout this thesis to predict polyadenylation signals. PAjHMMA allows the user to specify an HMM in a model file containing the number of states, their characteristic emission frequencies, and their transition probabilities. A model file is provided; this can contain any reasonable number of states, each having characteristic nucleotide emission frequencies. One particular motivation for the design of this software was to support states with an explicit length distribution.

The standard decoding algorithms have been modified to allow states to have a user-defined length distribution. Although the software described here was originally designed for the prediction of *C. elegans* polyadenylation signals, it is possible, given a manually built model with nucleotide frequency and length parameters for each state, to annotate any sequence for any feature having a sequence of states with distinctive nucleotide frequencies.

I next proceed to use the software described to predict polyadenylation signals in *C. elegans*. Chapter 3 explains how to build an accurate model of the *C. elegans*

polyadenylation signal. Based on the success of this, I build models for other species, which eventually enables me to carry out analyses on orthologues (chapters 5 and 6).