Hidden Markov models

Enrique Blanco Garcia
eblanco@imim.es

Contents

1 Signals and models

Actions in the real world produce observable outputs that can be characterized as signals. Signals can be discrete (symbols in an alphabet) or continous (speech samples), stationary or non stationary (signal properties vary with time), pure or corrupted from other sources (noise) or by transmission problems. Real world signals can be characterized in terms of signals models for several reasons:

1. A theoretical description or model can be used to process the output, enhance the signal, remove the noise and undo transmission distortions.
2. To learn about the signal source (the real world process) without having the source available. Very interesting when the cost of obtaining the signals from the actual source is high.
3. To develope practical systems for prediction, identification or recognition of the same signals in sources different from the original ones.

In general, signal models to characterize the properties of a signal can be classified into deterministic or statistical models. Deterministic models are based on some fixed and specific properties of the signal that must be measured in real examples. Statistical models characterize the signal as a parametric random process whose parameters must be estimated in a well-defined manner.

Cells functioning is essentially based on the mechanisms of transmission of biochemical signals among different cellular components. Models have been proposed to simulate the different signal pathways (growth, genetics, defense, ...) that occur simultaneously into the cell. Bioinformatics has provided uncountable models to deal with genomic sequences in gene prediction, modelling of protein structure, configuration of regulatory elements and other related problems classified into the field of sequence analysis.

Once a collection of data containing a signal has been obtained, different methods can be used to build a representation or model to characterize that family of sequences. This model will be also useful to detect the signal in other collections of sequences. Deterministic models such as consensus, regular expressions, position weight matrices or the basic pairwise or a multiple alignment of sequences are nowadays very common and well-known tools.

Important statistical models to represent a signal from a set of common sequences are discriminant analysis, iterative methods (e.g. Expectation Maximization or Gibbs sampling), Markov chains and Hidden Markov models (HMM). From the AI, non symbolic approaches as neural networks and other classification methods as Decision Trees or Support Vector Machines have been used to solve the same problems as well.

2 Markov chains

A discrete Markov process is a system $S = \{S_1, S_2, \ldots, S_N\}$ evolving from one state to another during a set of time instants $t = \{1, 2, \ldots, T\}$ associated with state changes. A state transition is defined according to a set of probabilities associated with the current and the predecessor states. For the case of discrete first order Markov chains, the state transition function is based on the current state and the previous state (q_t and q_{t-1}):

$$a(i,j) = P(q_t = S_j | q_{t-1} = S_i), \quad 1 \leq i, j \leq N \tag{1}$$

with the coefficients having the properties

$$a(i,j) \geq 0 \tag{2}$$

$$\sum_{j=1}^{N} a(i,j) = 1 \tag{3}$$

The initial state probabilities are denoted as:

$$\pi(i) = P(q_1 = S_i), \quad 1 \leq i \leq N \tag{4}$$

Starting probabilities can be also modelled by adding an initial state in the Markov chain, the beginning of the sequence. Traditionally the end of a sequence of observations is not modelled because it is assumed to finish anywhere, but an extra terminal state could be added to model a distribution of lenghts of the sequence.

The output of the process is the set of states at each instant of time, each one corresponding to a physical event (*observable Markov model*). Given a sequence of observations $O = \{O_1, O_2, \ldots O_T\}$ corresponding to the instants $t_1, t_2, \ldots t_T$, it is possible to determine the probability of O given the model $\lambda = (N, \{a(i,j)\}, \{\pi(i)\})$ as:

$$P(O|\lambda) = P(O_1) \cdot P(O_2|O_1) \ldots P(O_T|O_{T-1}) = \pi(O_1) \cdot a(O_1, O_2) \cdot a(O_2, O_3) \ldots a(O_{T-1}, O_T) \quad (5)$$

3 Hidden Markov models (HMM)

Markov chains are useful for discrimination. Given a sequence and a set of examples containing a signal, a Markov model can be designed to recognize the part of the sequence containing the signal in the examples (real model) and a second Markov model can be developed to identify the rest of the sequence (false model). Then, both Markov models are used on unannotated sequences to find the signal within, measuring the value for every window of x elements in every sequence (window sliding) and computing a likelihood ratio between both values to decide whether a region of the sequence is containing or not the signal.

However, the boundaries between a signal and the rest of element in the observations sequence are not exactly defined. On the contrary, there could be some overlapping effect between both regions or models that can not be detected using the window sliding technique because the lenght of the correct window is unknown and is changing from one example to another. A more satisfactory approach is to build a single model for the whole sequence of observations that incorporates both the real and the false previous models.

Then, the concept of a Markov model is extended to include the case where the observation is a probabilistic function of the state. Thus, the final model is a doubly embedded random process with: (1) a hidden stochastic process (*not observable*) that can observed through (2) another set of stochastic processes that produce the sequence of observations.

3.1 The urn and ball model

There are N urns containing colored balls in different proportions. Balls are in M distinct colours. The procedure to obtain a sequence of T observations is:

1. Init the clock ($t = 1$)
2. Select according to some random process, one of the urns
3. Extract a ball from the urn at random, recording the colour as observation
4. Replace the ball into the urn
5. Update new instant transition ($t = t + 1$) and repeat the process until the sequence of observations is finished ($t = T$)

The whole process generates a finite observation sequence of colours (observable output) but the sequence of urns dictated by some state transition function will be totally unknown (hidden states).

3.2 Elements of a HMM

1. States, $S = \{S_1, S_2 \ldots S_N\}$ (current state is denoted as q_t): hidden but in many applications with some physical significance attached (e.g. urns in the above example, ins-del-match operations in the alignment of genomic sequences or exons and introns in a genefinding program).
2. Observations, $V = \{V_1, V_2 \ldots V_M\}$: the symbols emitted by the states correspond to the physical output of the system being modelled (e.g. balls in the above example, nucleotides in the alignment of genomic sequences, protein coding sequence in a genefinding program)
3. The state transition probability distribution $A = \{a(i,j)\}$:

$$a(i,j) = P(q_{t+1} = S_j | q_t = S_i), \quad 1 \leq i, j \leq N \quad (6)$$

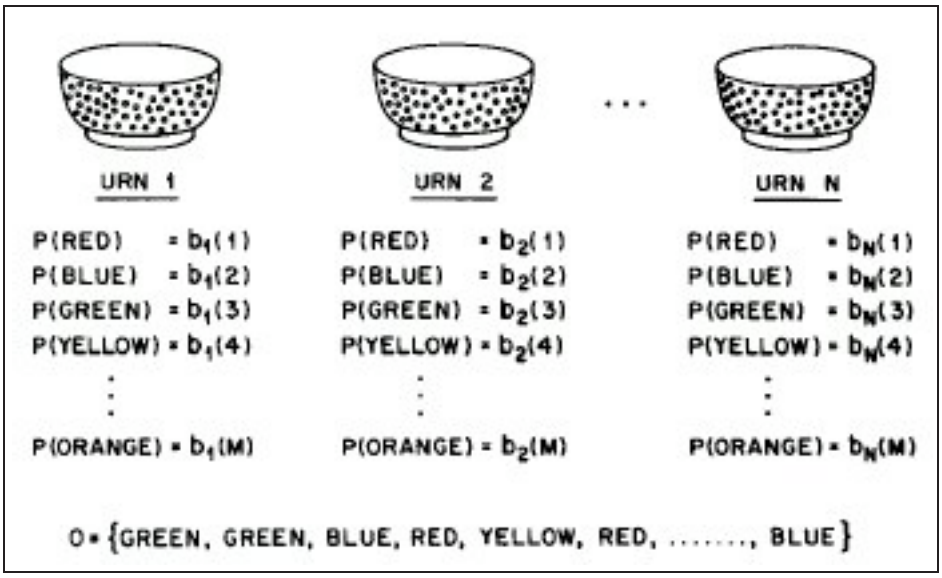

Figure 1: The urn and ball model: the observable output is the sequence of colours while the hidden model is the election procedure of states [1].

4. The emission probability distribution from a given state S_j, $B = \{b_j(k)\}$:

$$b_j(k) = P(V_k|q_t = S_j), \quad \begin{matrix} 1 \leq j \leq N \\ 1 \leq k \leq M \end{matrix} \tag{7}$$

5. The initial state distribution $\Pi = \{\pi(i)\}$:

$$\pi(i) = P(q_1 = S_i), \quad 1 \leq i \leq N \tag{8}$$

Given the appropiate values of N, M, A, B and Π, the HMM can be used as a generator to give a sequence of observations $O = \{O_1, O_2, \ldots O_T\}$ where each individual observation is one of the symbols in V:

1. Choose an initial state $q_1 = S_i$ $(t = 1)$
2. Obtain $O_t = V_k$ according to b_i
3. Transit to a new state $q_{t+1} = S_j$ according to $a(i, j)$ and the current state $(t = t + 1)$
4. Return to step 2 if $t \leq T$, terminate the procedure otherwise

4 The three basic problems

Once the main framework of the HMM has been defined there are several interesting problems that must be solved to use the model in real-world applications:

1. **The likelihood question**
   Given a sequence of observations $O = \{O_1, O_2, \ldots O_T\}$ and a HMM $\lambda = (N, M, A, B, \Pi)$, which is the probability $P(O|\lambda)$ that the observed sequence was produced by the model?
2. **The decoding question**
   Given a sequence of observations $O = \{O_1, O_2, \ldots O_T\}$ and a HMM $\lambda = (N, M, A, B, \Pi)$, which is the most probable state path $Q = \{q_1, q_2, \ldots q_T\}$ across the model that produced that observed sequence?
3. **The learning question (The training problem)**
   Given a sequence of observations $O = \{O_1, O_2, \ldots O_T\}$ and a HMM $\lambda = (N, M, A, B, \Pi)$, how do we adjust the model parameters to maximize the probability $P(O|\lambda)$?

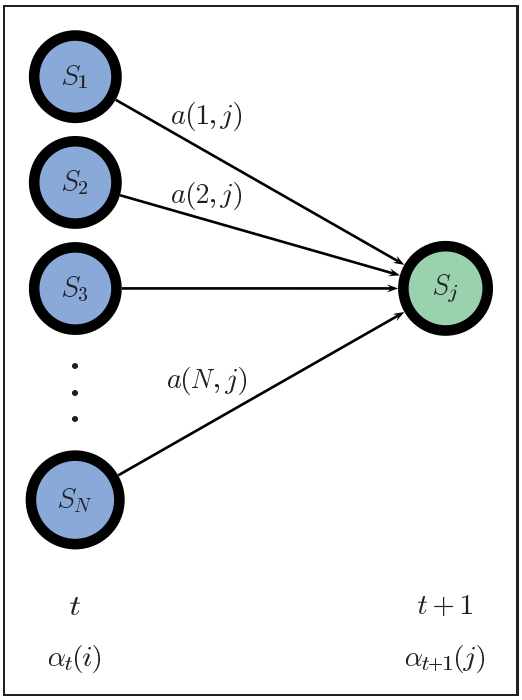

Figure 2: Forward procedure: the probabilities to generate every preffix of the sequence of observations are transferred from every previous instant t and state q_t to the next instant $t+1$ and state q_{t+1} as a propagation network [1].

5 The likelihood question

Given a model representing a signal or a family of signals, we would like to know whether a new sequence of observations $O = \{O_1, O_2, \dots O_T\}$ is a member of the previous family or not. In terms of the HMM, we are interested in the probability $P(O|\lambda)$ that such a sequence was produced by the current model.

Because many different state paths could produce the same sequence O, we must compute and add the probability for all possible paths $Q = \{q_1, q_2, \dots q_T\}$ generating the sequence of observations to obtain the final probability $P(O|\lambda)$[1]:

$$P(O|\lambda) = \sum_{\forall Q \to O} P(O, Q|\lambda) = \sum_{\forall Q \to O} \pi(q_1) \cdot b_{q_1}(O_1) \cdot a(q_1, q_2) \cdot b_{q_2}(O_2) \cdot a(q_2, q_3) \dots a(q_{T-1}, q_T) \cdot b_{q_T}(O_T) \quad (9)$$

Obviously, the number of paths increases exponentially with the length of the sequence so that in practice, the computation of the recurrence using this direct definition is not feasible. **Cost:** $\Theta(2TN^T)$ where $2T$ is the cost of computing the probability for a single path and N^T is the number of paths of length T.

5.1 The forward algorithm

Using dynamic programming techniques, a family of recursive algorithms known as *iterative propagation algorithms* provides feasible solutions to solve several problems related to the HMM. These methods are based on the transmission of results through the underlying directed graph structure associated to every HMM.

Let us define the problem of computing the probability to generate the given sequence of observations in terms of preffixes of the input: the forward variable $\alpha_t(i)$ is the probability to generate the subsequence $O_1 O_2 \dots O_t$ provided that the last symbol was generated from the state S_i.

[1] In fact, it is not necessary to discriminate between paths producing or not the sequence of observations. In some state in the path that can not generate O, there will be an emission probability zero for the given observation in that instant of time.

$$\alpha_t(i) = P(O_1 \dots O_t, q_t = S_i|\lambda) \tag{10}$$

and then, the probability that the sequence of observations O was produced by the model λ becomes the probability of finishing in all of the possible states:

$$P(O|\lambda) = \sum_{i=1\dots N} P(O_1 \dots O_T, q_T = S_i|\lambda) = \sum_{i=1\dots N} \alpha_T(i) \tag{11}$$

The following equation describes the process of (forward) transmission of the probabilities previously computed from the previous states to the current one taking into account the produced sequence of observations until that moment and the current observation:

$$\alpha_{t+1}(j) = \Big[\sum_{i=1\dots N} \alpha_t(i) \cdot a(i,j) \Big] \cdot b_j(O_{t+1}), \quad \begin{smallmatrix} 1 \le j \le N \\ 1 \le t \le T \end{smallmatrix} \tag{12}$$

The initial values for α are defined as:

$$\alpha_1(i) = \pi(i) \cdot b_i(O_1), \quad 1 \le i \le N \tag{13}$$

Algorithm 1 The Forward algorithm. **Cost:** $\theta(N^2T)$

INPUT: $\lambda = (N, M, A, B, \Pi), O = \{O_1, O_2, \dots, O_T\}$
OUTPUT: $p = P(O|\lambda)$

```
    (* Initialize the forward variable *)
 1: for i = 1...N do
 2:    α_1(i) = π(i) · b_i(O_1)
 3: endfor

    (* Forward procedure *)
 4: for t = 2...T do
 5:    for j = 1...N do (* Compute the paths finishing on every state S_j *)
 6:       sum = 0
 7:       for i = 1...N do (* Retrieve the forward values coming from previous states S_i *)
 8:          sum = sum + α_t(i) · a(i, j)
 9:       endfor
10:       α_{t+1}(j) = sum · b_j(O_{t+1})
11:    endfor
12: endfor

    (* Final probability *)
13: p = 0
14: for i = 1...N do
15:    p = p + α_t(i)
16: endfor
```

6 The decoding question

With the Forward algorithm, it is computed the probability $P(O|\lambda)$ that a sequence of observations was produced by a given model. Thus, this probability can be rewritten in terms of a *score* or *p-value* to accept or not the sequence as a new member of the family that was used to build the HMM. However, it is also interesting to discover which is the most probable state path $Q = \{q_1, q_2, \dots q_T\}$ across the model that produced that observed sequence. In fact, this path of states (with some predefined meaning) divides the sequence into different parts that can be classified according to the presence or absence of the different signals hidden in the model (e.g. exons and introns in a HMM-genefinder).

6.1 The Viterbi algorithm

The most probable path (optimal, best) Q^* can be recursively found without generating all of the paths along the model, dividing the sequences of observations and states in preffixes of paths as in the case of the Forward algorithm. Nevertheless, at every layer of nodes we are now only interested in the path *maximizing* the probability to produce the sequence of observations. Thus, selecting one path implies discarding the other $N-1$ paths arriving at the current node because they have a lower probability until that stage[2].

As in the forward algorithm, the variable $\delta_t(i)$ is defined as the best score (highest probability) along a single path ending in a given a state S_i, that can be obtained at time t, producing the first t observations:

$$\delta_t(i) = max_{q_1 \ldots q_{t-1}} \big[P(q_1 \ldots q_t = i, O_1 \ldots O_t|\lambda)\big] \tag{14}$$

and by induction,

$$\delta_{t+1}(j) = \Big[max_i \ \delta_t(i) \cdot a(i,j)\Big] \cdot b_j(O_{t+1}), \quad \begin{smallmatrix} 1 \le i,j \le N \\ 1 \le t \le T \end{smallmatrix} \tag{15}$$

The initial values are:

$$\delta_1(i) = \pi(i) \cdot b_i(O_1), \quad 1 \le i \le N \tag{16}$$

For every state j and instant t, it is necessary to keep track of the argument which maximized δ_t:

$$\psi_t(j) = argmax_{1 \le i \le N} \ \big[\delta_{t-1} \cdot a(i,j)\big], \quad \begin{smallmatrix} 1 \le j \le N \\ 2 \le t \le T \end{smallmatrix} \tag{17}$$

with initial values:

$$\psi_1(i) = 0, \quad 1 \le i \le N \tag{18}$$

Then, the highest probability P^* and the associated path Q^* can be retrieved as:

$$P^* = max_{1 \le i \le N} \ \big[\delta_T(i)\big] \tag{19}$$

$$q_T^* = argmax_{1 \le i \le N} \ \big[\delta_T(i)\big] \tag{20}$$

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2 \ldots 1 \tag{21}$$

7 The learning question

Given a HMM and a set of training sequences $X^1 \ldots X^n$, there is not any method to estimate the model parameters which maximize the probability of the observation (*NP-complete problem*). However, there are several methods to adjust a HMM to locally maximize the value $P(O|\lambda)$ for every training sequence such as the Baum-Welch algorithm (a well-known variant of expectation-maximization procedure) or different forms of gradient descent.

The Baum-Welch algorithm is based on the same concept as the Forward and Viterbi procedures, that is, computing recursively the best preffixes of the general paths in such a way that it is not necessary to maintain the whole tree of combinations at every stage. In addition to this basic idea, as probabilities can be added following left-right or right-left connections, the best suffix of the complete sequence of observations and states can be computed using a variant of the Forward algorithm, the Backward algorithm. Thus, given a state of the HMM and one observation, the information computed from the best partial paths finishing (Forward) and starting (Backward) on such state can be combined to maximize the global value $P(O|\lambda)$.

[2]Due to the process of construction, the property of additivity must be observed and therefore the order relationship between two different paths arriving at the same node will be conserved even when new elements are included in the solution (both will share the rest of the solution).

Algorithm 2 The Viterbi algorithm. **Cost:** $\theta(N^2T)$

INPUT: $\lambda = (N, M, A, B, \Pi), O = \{O_1, O_2 \dots O_T\}$
OUTPUT: The best $Q^* = \{q_1^*, q_2^*, \dots q_T^*\}$ such that $P^* = P(Q^*, O|\lambda)$

(* Initialize the δ and ψ variables *)
for $i = 1 \dots N$ **do**
  $\delta_1(i) = \pi(i) \cdot b_i(O_1)$
  $\psi_1(i) = 0$
endfor

(* Viterbi procedure *)
for $t = 2 \dots T$ **do** (* Compute the best complete path finishing on every state S_j *)
  for $j = 1 \dots N$ **do**
    max $= 0$
    $\psi_t(j) = 0$
    for $i = 1 \dots N$ **do** (* Select the best path coming from every previous state S_i *)
      if $(\delta_t(i) \cdot a(i,j) \geq \max)$ **then**
        max $= \delta_t(i) \cdot a(i,j)$
        $\psi_t(j) = i$
      endif
    endfor
    $\delta_{t+1}(j) = \max \cdot b_j(O_{t+1})$
  endfor
endfor

(* Final processing: obtaining P^* and Q^* *)
$P^* = 0$
for $i = 1 \dots N$ **do**
  if $(\delta_T(i) \geq P^*)$ **then**
    $P^* = \delta_T(i)$
    $q_T^* = i$
  endif
endfor

(* Retrieving recursively the sequence of states *)
$Q^* = \text{RetrievePath}(Q^*, q_T^*, \psi, T)$

7.1 The Backward algorithm

Let us define the problem of computing the probability to generate the given sequence of observations in terms of suffixes of the input: the backward variable $\beta_t(i)$ is the probability to generate the subsequence $O_{t+1}O_{t+2} \dots O_t$ provided that the t-st symbol was generated from the state S_i.

$$\beta_t(i) = P(O_{t+1} \dots O_T, q_t = S_i|\lambda) \tag{22}$$

Now, paths are constructed as suffixes of the whole sequence of states and observations, from the end (O_T) to the beginning (O_1). Therefore, given a state S_i, to compute $\beta_t(i)$ is necessary to *backpropagate* (from right to left) the values of the probabilities for the paths starting on every state emitting the next symbol (O_{t+1}):

$$\beta_t(i) = \sum_{j=1 \dots N} a(i,j) \cdot b_j(O_{t+1}) \cdot \beta_{t+1(j)}, \quad \begin{matrix} t = T-1, T-2 \dots 1 \\ 1 \leq i \leq N \end{matrix} \tag{23}$$

Initialization of β variable is arbitrarily defined as:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{24}$$

And the final probability must be computed from the probabilities of starting in all of the possible states:

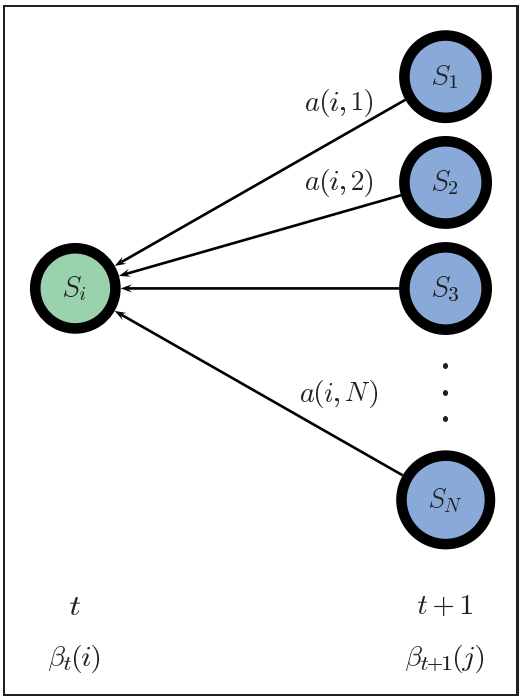

Figure 3: Backward procedure: the probabilities to generate every suffix of the sequence of observations are transferred from the following states q_{t+1} to the current state q_t as a propagation network [1].

$$P(O|\lambda) = \sum_{i=1\ldots N} \beta_1(i) \tag{25}$$

7.2 The Baum-Welch algorithm (Forward-Backward)

Given a set of training sequences $X^1 \ldots X^n$, the learning problem consists on adjusting the parameters to maximize the value $P(X^i|\lambda)$ for each sequence. Considering the independence between the training sequences of observations, the global likelihood of the model can be written as:

$$P(X^1 \ldots X^n|\lambda) = \Pi_1^n P(X^i|\lambda) \tag{26}$$

The likelihood of the model is exactly the function to maximize (*fitness function*). In terms of logarithms, the product becomes a sum and the log likelihood of the model is:

$$logP(X^1 \ldots X^n|\lambda) = \sum_1^n logP(X^i|\lambda) \tag{27}$$

Estimation of the model parameters A,B and Π when the state sequence is known

Also known as supervised learning in AI, the estimation of the initial parameters when the training sequences have been previously labelled simply consists on recording the states and transitions used in the examples to update the model (e.g experimentally annotated genes or proteins or precomputed multiple alignments):

1. Count the number of transitions from state i to state j in the training set ($\bar{a}(i,j)$).
2. Count the number of emissions of symbol V_k from state i in the training set ($\bar{b}_i(V_k)$).
3. Estimate the significance of every value using a quotient between the value and the total sum of values (*maximum likelihood estimator*). Thus, new values for A, B and Π can be computed as:

Algorithm 3 The Backward algorithm. **Cost:** $\theta(N^2T)$

INPUT: $\lambda = (N, M, A, B, \Pi), O = \{O_1, O_2 \dots O_T\}$
OUTPUT: $p = P(O|\lambda)$

```
    (* Initialize the backward variable *)
 1: for i = 1...N do
 2:    β_T(i) = 1
 3: endfor

    (* Backward procedure *)
 4: for t = T − 1...1 do (* Compute the paths starting on every state S_i *)
 5:    for i = 1...N do
 6:       sum = 0
 7:       for j = 1...N do (* Send back the probability values from following states S_j *)
 8:          sum = sum + [a(i, j) · b_j(O_{t+1} · β_{t+1(j)}]
 9:       endfor
10:       β_t(i) = sum
11:    endfor
12: endfor

    (* Final probability *)
13: p = 0
14: for i = 1...N do
15:    p = p + β_1(i)
16: endfor
```

$$a(i,j) = \frac{\bar{a}(i,j)}{\sum_{l=1}^{N} \bar{a}(i,l)}, \quad 1 \leq i,j \leq N \tag{28}$$

$$b_i(V_k) = \frac{\bar{b}_i(V_k)}{\sum_{l=1}^{M} \bar{b}_i(V_l)}, \quad \begin{matrix} 1 \leq i \leq N \\ 1 \leq k \leq M \end{matrix} \tag{29}$$

$$\pi(i) = \frac{\bar{\Pi}(i)}{\sum_{l=1}^{N} \bar{\Pi}(l)} \tag{30}$$

To avoid undefined values in the quotients, predetermined *pseudocounts* are usually added. These pseudocounts should reflect the prior knowledge about the probability values (large values for well-based knowledge and smaller for weaker prior information).

Estimation of the model parameters A, B and Π when the state sequence is unknown

Also known as unsupervised learning, the estimation of the values for the HMM is usually calculated using some type of iterative algorithm. These algorithms will re-estimate the parameters of the model given the training set of sequences until a maximum number of iterations is reached or when gains or losses in the global likelihood of the model are below some fixed threshold.

The main problem of this approach is the selection of the initial values which may lead the estimation procedure to a local maximum. Once initial values have been introduced, the technique of maximum likelihood can be employed to re-estimate the model parameters. The number of expected transitions and emissions is obtained from the forward and backward variables applied to the training sequences.

For a given sequence of the training set, the expected number of transitions from the state S_i to the state S_j in the training data are computed through the combination of the forward variable (paths finishing in S_i) and the backward (paths starting in S_j) for a given couple of observations (O_t, O_{t+1}) and a model λ:

$$\bar{a}(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{31}$$

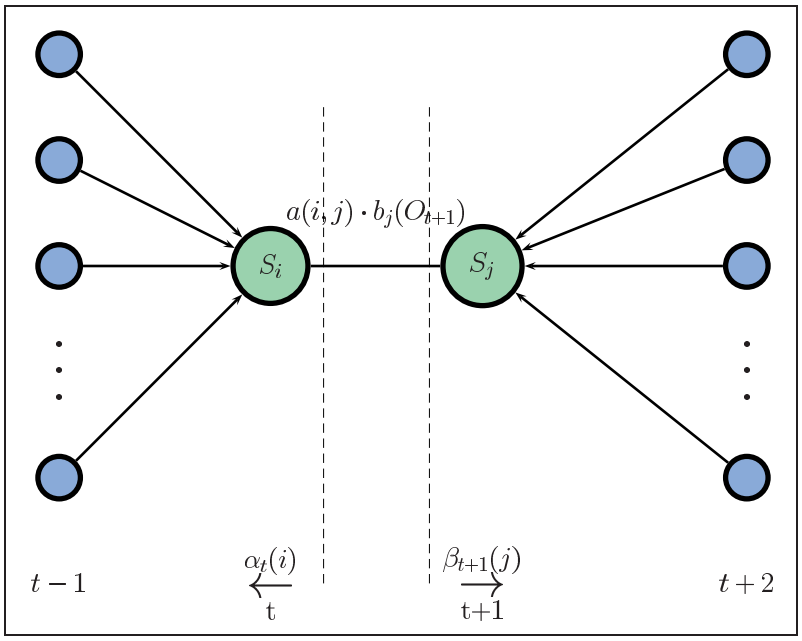

Figure 4: Forward-Backward procedure: the probability to be in a given state S_i in the instant t can be deduced from the probability of finishing on such state emitting the first t symbols and the probability of starting from that state, emitting the rest of the sequence of observations [1]

Using the forward and backward variables, the value of $\bar{a}$ can be computed:

$$\bar{a}(i,j) = \frac{\alpha_t(i) \cdot a(i,j) \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) \cdot a(i,j) \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)} \tag{32}$$

The numerator term (*observed probability*) is normalized using the sum of probabilities over all pairs of states in the model (*sum of probabilities*). For the expected emission probabilities, the expected number of times being in state S_j and observing V_k is:

$$\bar{b}_i(V_k) = P(q_t = S_i, O_t = V_k | O, \lambda) \tag{33}$$

$$\bar{b}_i(V_k) = \frac{\alpha_t(i) \cdot \beta_t(i), if(O_t = V_k)}{\sum_{i=1}^{N} \alpha_t(i) \cdot \beta_t(i)} \tag{34}$$

Again the numerator is normalized with the probability of being on the state, emitting any symbol.
Finally, for the initial states:

$$\bar{\Pi}(i) = P(q_1 = S_i | O, \lambda) = \frac{\alpha_1(i) \cdot \beta_1(i)}{\sum_{i=1}^{N} \alpha_1(i) \cdot \beta_1(i)} \tag{35}$$

Again, new values for A, B and Π are estimated from the expected values $\bar{A}$, $\bar{B}$ and $\bar{\Pi}$ using the method of maximum likelihood estimation (quotient between observed probability and total sum of probabilities). This procedure must be repeated for all of the sequences in the training set on each iteration of the Baum-Welch algorithm.

Algorithm 4 The Baum-Welch algorithm. **Cost:** θ(MAXITERATIONS n N^2T)

INPUT: $\lambda = (N, M, A, B, \Pi), X = X^1, X^2 \dots X^n$
OUTPUT: $p = \sum_1^n P(X^i|\lambda)$ is a local maximum

(* Initialize the model *)
1: λ = InitialModel()

(* EM procedure *)
2: numIterations = 0
3: **repeat** (* Compute $\bar{\lambda}$ using every sequence in the training set *)
4: $\bar{\lambda}$ = NewModel()
5: **for** $x = X^1 \dots X^n$ **do**
6: α = computeForwardVars(λ, O, x)
7: β = computeBackwardVars(λ, O, x)

(* E step: update the values of $\bar{A}, \bar{B}$ and $\bar{\Pi}$ *)
8: $\bar{\lambda}$ = updateValues(λ, α, β)
9: **endfor**

(* M step: compute maximum likelihood estimators *)
10: λ = estimateModel($\bar{\lambda}$)
11: likelihood = computeLikelihood(λ,O,X)
12: numIterations = numIterations + 1
13: **until** numIterations = MAXITERATIONS or Improvement(likelihood) = FALSE

8 Application 1: sequence alignment (profileHMMs)

The observed variation or conservation on some features in a family of biological sequences can be statistically described using diferent methods:

- Substitution matrices (e.g PAM and BLOSUM collections) contain the probability that one amino acid mutates to another. Both are used for pairwise alignments of proteins.
- Position weight matrices are calculated counting the frequency of the four nucleotides at all of the positions in a set of signals extracted from know examples (e.g. splice sites or transcription factor binding sites). Then, this matrix can be used to detect the same signal on other sequences.
- Codon usage bias or overrepresentation of oligonucleotides is widely used to predict distinct important regions in genomic-scale analysis (e.g. genes, CpG islands or promoters).

8.1 Profile hidden Markov model

A profile hidden Markov model (pHMM) is a probabilistic profile derived from a family of sequences (usually proteins). This profile can be used to identify new members of the same family in large databases of sequences. pHMMs are HMM with a structure specifically designed to allow position dependent gap penalties (*position sensitive gap scores*) to complement the substitution schema provided by HMMs (*position sensitive substitution scores*)[3]. In fact, pHMMs are an alternative to classical multiple sequence alignment methods although both approaches can be complementary because pHMM can be constructed from a previous multiple alignment or from scratch.

As pHMM are modelling multiple alignment, the states in a pHMM can be classified as:

1. **Match states:** they model the columns of the alignment. The emission probabilities are obtained directly from the frequency of each amino acid in a given position. In fact, every block of Match states without gaps is actually a position weight matrix.

[3]Substitution of amino acids in HMMs is implemented with the emission function so that no general substitution matrix is required.

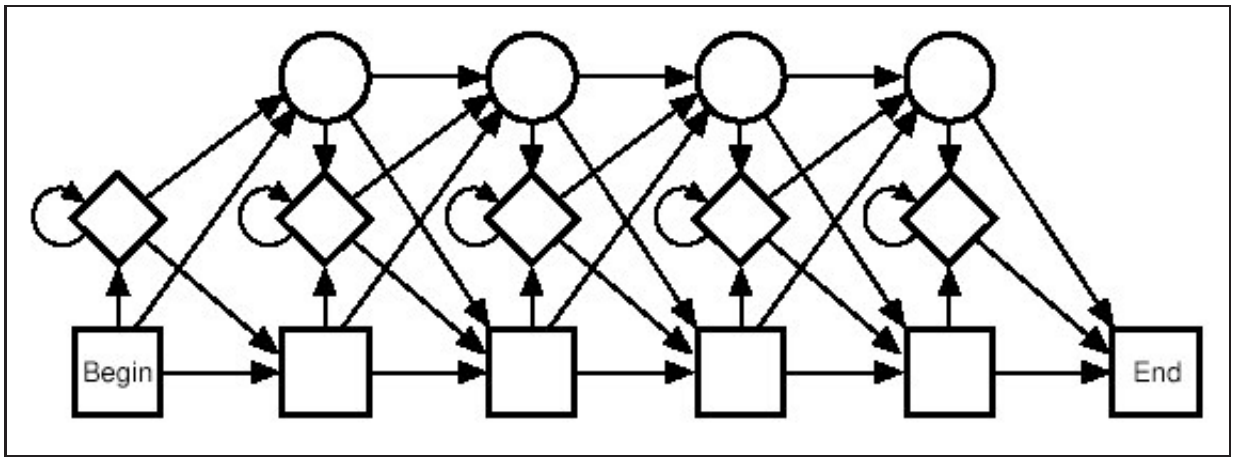

Figure 5: pHMM common architecture (left-right): squares are match states, diamonds are insertion states and circles are deletion states. Emission and transition probabilities are used exactly as in the case of simple HMM to generate a multiple alignment of sequences. [4]

2. **Insertion states:** they model highly variable regions in the alignment. The composition of those regions is employed to define the transition and emission probabilities of a given insertion state.
3. **Deletion state:** they do not emit any symbol (silent states). It is a mechanism to model gaps in blocks of match states. Deletion states are necessary to skip consecutive match states without implementing all of the connections between them.

8.2 pHMM Construction from a multiple sequence alignment

A pHMM defines a probability distribution over the whole space of sequences. The objective of the training process will be to control the shape of that distribution by associating the peaks of the function around members of the family. Two important decisions must be taken into account:

1. **The length of the model:** a decision about which multiple alignment columns must be assigned to match states and which must be modelled with insertion states. The heuristic rule is to consider columns with more than half gap characters as highly variable regions. MAP (maximum a posteriori) is a dynamic programming algorithm to find column assignments that maximize the posteriori probabilities of the model at the same time as fitting the HMM parameters.
2. **The parameters of the model $\lambda = (A, B, \Pi)$:** Initially, both emission, transition and initial probabilities can be estimated from the multiple alignment using the technique of maximum likelihood but using counts restricted to the current block (insert or match state). Again, pseudocounts can be introduced to avoid that unvisited transitions and emissions can produce errors in the quotients (e.g. Laplace's rule: add one in every frequency value).

8.3 Recovering the multiple alignment from the pHMM

Once the values of the multiple alignment have been loaded into the pHMM, the multiple alignment of those sequences can be retrieved using the Viterbi algorithm, that is, obtaining the best (most probable) path of states across the pHMM that could have produced that sequence. Thus, aligning the sequence to the model, a sequence of states (enumerated M,I and Ds) will be obtained. Amino acids in the match regions will correspond to the shadow parts in the previous multiple alignment. Amino acids in the insert regions will be arbitrarily left-justified (no aligned) as they are part of the variable regions without useful information to establish the relationship among the sequences[4]. Repeating this process for all the sequences and aligning the enumerated match states, the global multiple alignment of the sequences is retrieved from the pHMM.

[4]These regions ideally represent unconserved or atypical parts of the sequences that are however aligned by classical progressive methods.

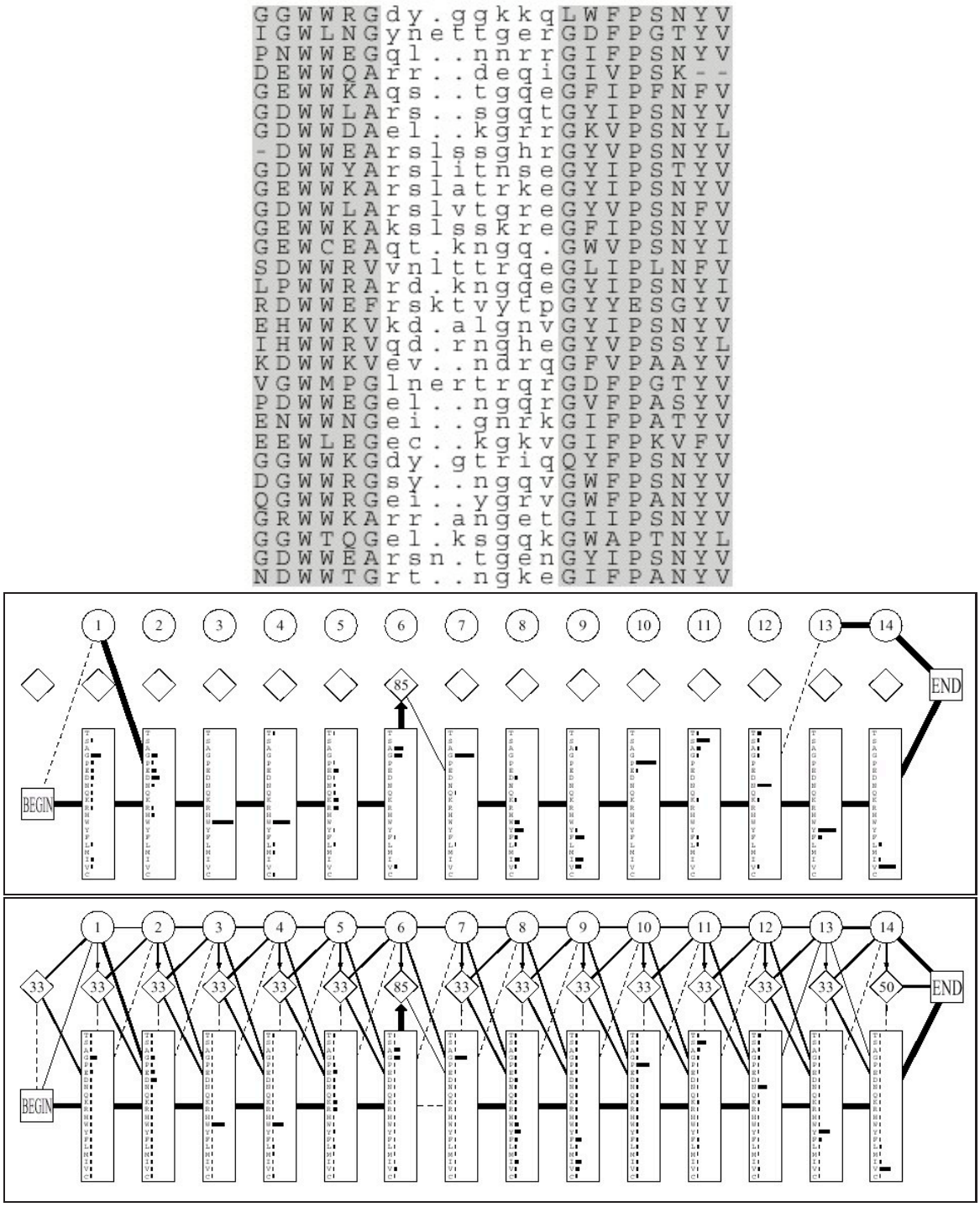

Figure 6: **Upper half:** Alignment of 30 short amino acid sequences extracted from the alignment of the SH3 domain. Because of their degree of conservation, shaded areas have been decided to be the match states and the unshadded area will be modelled with one state of insertion. **Middle half:** A pHMM made from the previous alignment. There are 14 main states corresponding to 6+8 columns in the shadded area, one insertion corresponding to the variable region in the alignment and two deletions corresponding to gaps in position1 (sequence 8) and in positions 13/14 (sequence 4).Bold transitions with high probability are shown as strong lines while those with small probabilities are shown as dashed lines. Transitions with probability zero are not shown. Number in the diamonds are the probability transtition to remain on that state. Emission in match states are shown as an histogram. **Lower half:** The same pHMM but using pseudocounts calculed with Laplace's rule. [4]

8.4 pHMM Construction from a set of unaligned sequences

From a set of unaligned sequences, it is possible to construct a pHMM producing the multiple alignment during the process by using the Viterbi learning or the classical Baum-Welch algorithm. Since both training procedures find local optima, the selection of the initial model parameters is critical and it must be performed very carefully. Some suggestions are:

1. Select different starting points to observe (or not) convergence in the results. Start from random points or from precomputed alignments using greedy algorithms.
2. Use some form of stochastic search algorithm to scape from local maxima (simulated annealing, Monte Carlo methods, Tabu search).
3. Define sensible transition probabilities (e.g. probability from Match to Match states must be higher than probability from Match state to Insertion state).
4. Use model surgery to adaptively evolve the architecture during the training process (e.g. erase match states that have not been frequently visited).

8.5 Searching a database with pHMMs

The most important application of a pHMM representing a family of sequences is finding new sequences in a database showing a high similarity to the members of this family. Given a database and a pHMM, the sequences of the database can be aligned to the model using the algorithm of Viterbi, obtaining the most probable path and the associated probability. This probability can be compared to the score obtained by random sequences in order to establish the real probability of the sequence (p-value). Then, sequences in the database can be ordered and filtering using this value. Alternatively, the Forward algorithm can be also employed to obtain the probability that a new sequence was generated by the model.

As it is known, the cost of performing a multiple alignment of n sequences (k nucleotides) is $\theta(n^k)$. The cost of Viterbi algorithm is $\theta(kn^2)$ which is clearly lower as long as the pHMM has been trained off-line. The algorithm of Viterbi is essentially the same, but connections between states are restricted to 3 options taking into account the class of the current state:

- $M_j \leftarrow M_{j-1}, I_{j-1}, D_{j-1}$
- $I_j \leftarrow M_j, I_j, D_j$
- $D_j \leftarrow M_{j-1}, I_{j-1}, D_{j-1}$

Therefore, the δ, A and B functions must be rewritten in terms of match, insert and delete states:

$$\delta_t^M(j) = b_j^M(O_t) + max\{ \begin{matrix} \delta_{t-1}^M(j-1) \cdot a^{MM}(j-1,j) \\ \delta_{t-1}^I(j-1) \cdot a^{IM}(j-1,j) \\ \delta_{t-1}^D(j-1) \cdot a^{DM}(j-1,j) \end{matrix} \tag{36}$$

$$\delta_t^I(j) = b_j^I(O_t) + max\{ \begin{matrix} \delta_{t-1}^M(j) \cdot a^{MI}(j-1,j) \\ \delta_{t-1}^I(j) \cdot a^{II}(j-1,j) \\ \delta_{t-1}^D(j) \cdot a^{DI}(j-1,j) \end{matrix} \tag{37}$$

$$\delta_t^D(j) = max\{ \begin{matrix} \delta_{t-1}^M(j-1) \cdot a^{MD}(j-1,j) \\ \delta_{t-1}^I(j-1) \cdot a^{ID}(j-1,j) \\ \delta_{t-1}^D(j-1) \cdot a^{DD}(j-1,j) \end{matrix} \tag{38}$$

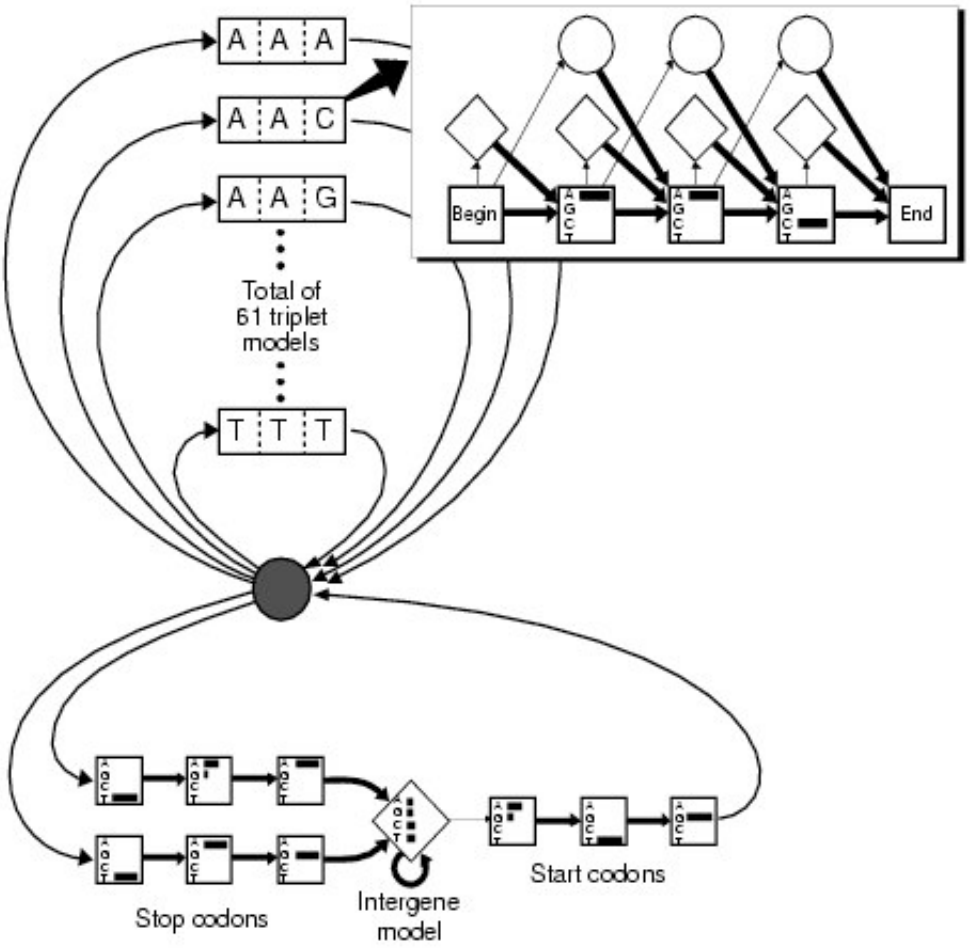

Figure 7: Basic architecture of the program EcoParse for prediction of genes in *E. Coli*. The central state is a silent node. The intergenic region is modelled using a insertion state while the coding regions are described with a profile HMM for each codon. The thickness of the arrows indicate the fraction of the sequences making this transition. [4]

9 Application 2: prediction of genes

There are a subset of problems in biological sequence analysis in which is essential to detect some interesting features (e.g. genes, regulatory elements, exons, ...). Thus, it is very common to define a set of grammatical rules describing those elements and their connections.

Eukaryotic gene structure (exons and introns) is a classical example of that family of problems that can be represented using a regular grammar. HMM states can be easily adapted to model exons, introns, promoter regions, intergenic regions, 5'UTRs or 3'UTRs. The probabilities for transitions between states are modelling the changes in the gene model (e.g. rules defining that an intron can be followed by an internal or terminal exon). The emission probabilities are based on the different composition of coding and non-coding DNA on each species.

Each random walk on the HMM has assigned a probability determined by the parameters of the model. Basically, predict genes on a DNA sequence (the sequence of observations) using a HMM is to obtain the most likely path of states through the model that produced that sequence (a parse of the sequence). Therefore, the input sequence will be translated (labelled) into series of states or gene components using the Viterbi algorithm providing a complete description of the precise locations of each feature. Viterbi equations are preserved although it must be taken into account that a δ function must be maintained for each type of state (exon, intron, ...).

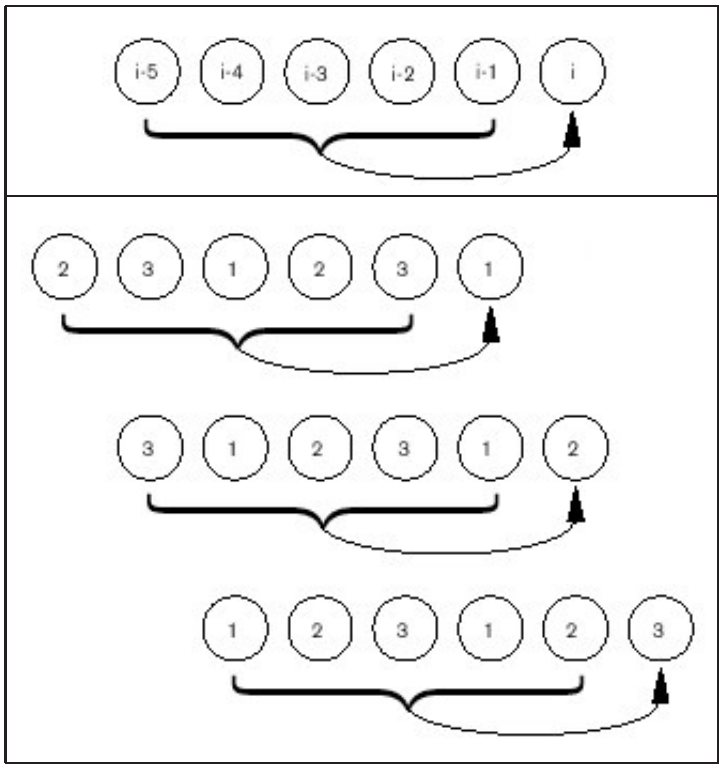

Figure 8: **Upper half:** example of 5-th order inhomogeneous model to analyze the composition in non-coding regions. **Lower half:** example of 5-th order homogeneous model to analyze the codon bias in protein coding regions. For a given state, one of them will be selected to emit the new symbol taking into account the position in the codon and the previous nucleotides. [6]

The simplest HMM consists of a collection of rings, connected to a central state. Each ring contains one or more profile HMMs to produce the 64 available codons. To deal with frameshift or sequencing errors, insertions and deletions are modelled into the profileHMMs. There is also an additional ring to generate intergenic DNA. The bias in codon usage is reflected in the probability of making a transition into the corresponding codon model.

9.1 HMM gene prediction in eukaryotes

To detect exon-defining signals in eukaryotic genes such as Splice sites or Translation signals, a pHMM can be trained for each type of signal, usually without insert and delete states.

To discriminate between coding and non-coding regions it is frequently used the observed codon bias in the exonic sequence. Thus, the emission probabilities in the coding states of the HMM can be conditioned on the 5 previous characters to model dependences between two adjacent amino acids in the protein. This is implemented using Markov models of order $n = 5$ which are configured to capture local dependencies between elements in oligonucleotides having $n + 1$ elements. Thus, the emission function in the state S_j for the symbol V_k can be rewritten in terms of the previous pentanucleotide:

$$b_j(V_k^t | V_{k_1}^{t-5} V_{k_2}^{t-4} V_{k_3}^{t-3} V_{k_4}^{t-2} V_{k_5}^{t-1}) \tag{39}$$

Higher-order Markov chains (e.g. position weight arrays) can be used to capture correlations between neighbouring nucleotides in splice site and translation signals as well.

In addition to capture correlations between the previous pentamer and the current nucleotide, it is important to take into account that these dependencies are differently expressed according to the position in the codon occupied for the emitted new symbol. Thus, Markov models can be classified as inhomogeneous or homogeneous:

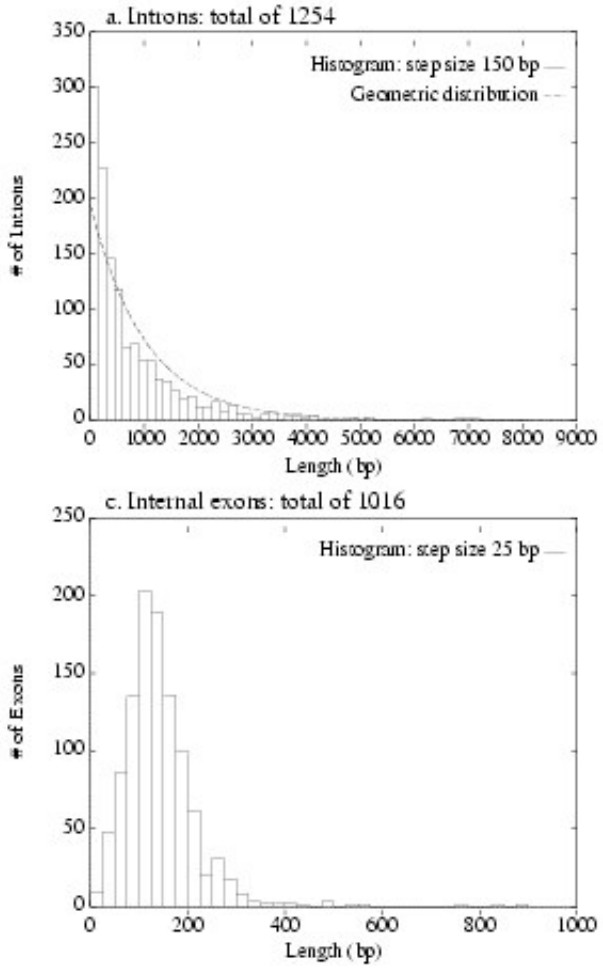

Figure 9: Length distributions of introns and internal exons in human genome [6].

- Inhomogeneous Markov models (coding regions): Markov models in which every state contains three diferent emission functions according to the position in the current codon of the region for that state (nucleotide). Inhomegeneous 3-periodic fifth-order Markov models are usually implemented to detect dependences within two consecutive codons.
- Homogeneous Markov models (non-coding regions): Markov models in which there is only one emission function.

9.2 Generalized HMMs

Experimental evidences indicates that the number of exons in a gene does not follow a geometric distribution (smaller exons are more likely than larger ones). On the contrary, it seems to be a preference in favour of medium-sized internal exons that could be more easily spliced. Therefore, it is necessary to describe the probability function of symbol emissions from exon states with an arbitrary distribution.

An important limitation of HMMs is that the length of consecutive strings generate from the same state are always geometrically distributed. To overcome this problem, the behaviour of the states can be redefined in the following manner. In a simple HMM, the system is during only one single unit time in a given state, generating zero or one symbols. In a Generalized HMM (GHMM) [5], the duration of the time spent in each

[5]GHMM receive also the name of Hidden semi-Markov models or explicit state duration models.

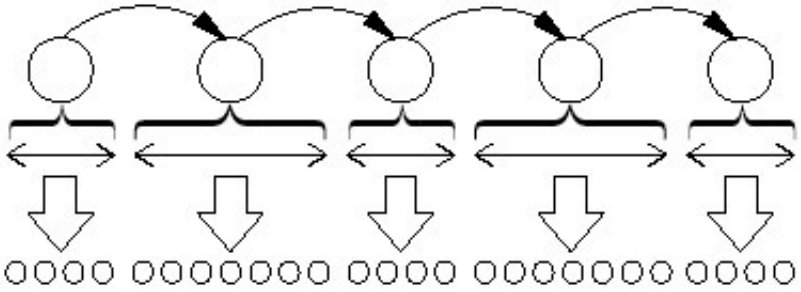

Figure 10: GHMMs are HMMs in which a state generates zero or more symbols. Thus, self-transitions are substituted for simple transitions from one state to another, spending a variable amount of time on it [6].

state is defined by a separate probability distibution which depends on the state type, producing zero or more symbols. Then, the length of predicted exons can be arbitrarily modelled configuring this additional function. The cost of Viterbi algorithm over GHMMs becomes prohibitive involving an additional recursion which must at each position, search back over all previous positions generated on the same state. This expensive cost can reduced in practice, fixing the maximum length of state durations or following another heuristical approaches.

The generation of an artificial genomic sequence of length L containing gene structures can be performed following the next steps:

1. Choose an initial state q_1 (function of initial probabilities).
2. Select a state duration l according to some especific distribution function defined for the type of the current state.
3. Produce a segment of l nucleotides (emission function for that state). If the system is in a coding state, the segment will have been generated taking into account the frame, codon position and exon-defining signals.
4. Move to another state q_j (function of state transition)
5. Repeat 2..4 until the sum of state durations is L

9.3 Advanced HMM gene prediction

Generalized Pair HMMs

Despite dynamic programming has been widely used in the areas of sequence alignment and gene finding, traditionally there was no connection between the solutions employed for solving both problems. HMMs can also used to perform pairwise sequence alignments (pair HMMs or PHMMs). Changing the emission function, states in PHMMs produce a pair of symbols corresponding to a match between both input sequences.

Generalized pair HMMs (GPHMMs) are based on GHMMs used to parse a genomic sequence classifiying every nucleotide into a different category: exon, intron, ... To analyze the information contained in the conservation of exon sequence in the same gene from two related species, GPHMM states must produce exon-pairs according to some predetermined joint distribution. Thus, the final product of the model is a global sequence alignment incorporating the annotation of exons in both organisms simultaneously. Allowing to produce symbols in one of the two sequences, it is possible to align cDNAs or proteins to genomic regions, improving the quality of the annotation.

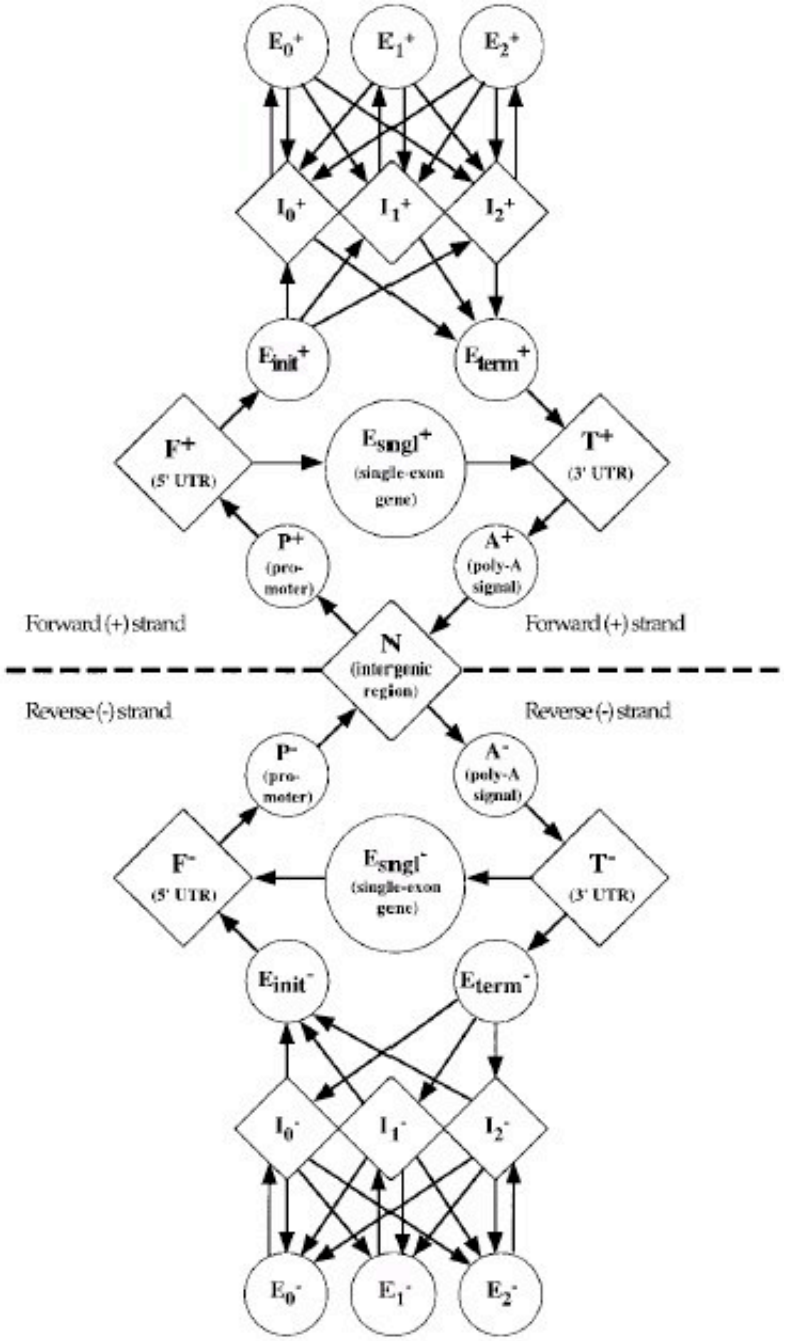

Figure 11: GENSCAN architecture: each circle represents a functional unit of a gene (exons and promoter elements) whereas each diamond corresponds to non-functional elements (introns, UTRs and intergenic regions). The system is divided into two parallel parts to detect genes in both strands of the DNA sequence [6].

HMM sampling

Optimization algorithms usually provide the best solution for a problem which may strongly depend on a particular set of parameters. However, it has frequently been observed with most deterministic strategies that biologically significant elements (real splice sites, exons or transcription factor binding sites) are classified as suboptimal solutions.

HMM sampling algorithm is a variant of the forward and backward algorithms based on randomly sampling state paths from the HMM, in order to find the optimal solution and addtionally other suboptimal solutions. This

alternative to the Viterbi algorithm has been employed to find genes affected by alternative-splicing including conserved forms between related organisms.

10 Bibliography

This work pretends to be a simple but comprehensive introduction to the exciting field of Hidden Markov models from an algorithmic and statistic point of view, including elemental procedures as Forward, Viterbi, Backward and Baum-Welch algorithms. Two important HMM applications in bioinformatics have been covered: alignment of sequences (profiles) and prediction of genes. Obviously, there are very important topics that have not be discussed here as the architecture of the HMM or more complex techniques for training such as using complex mixtures of data to set up the model (pseudocounts). Moreover, there was not room for important applications of HMMs inside and outside bioinformatics (e.g. speech recognition).

References

[1] *A tutorial on hidden Markov models and selected applications in speech recognition.* L.R. Rabiner. **Proceedings of the IEEE**, vol 77:257-285 (1989).

[2] *Biological sequence analysis: probabilistic models of proteins and nucleic acids.* R. Durbin, S. Eddy, A. Krogh and G. Mitchinson. **Cambridge University press** (1998).

[3] *Bioinformatics - The machine learning approach.* P. Baldi and S. Brunak. **MIT press** (2001)

[4] *Computational methods in molecular biology.* A. Krogh. **Elsevier** (1998)

[5] *Algorithms for molecular biology (course lectures)* R. Shamir.
At **http://www.,ath.tau.ac.il/∼rshamir/algmb/algmb00.html** (2000)

[6] *Identification of genes in human genomic DNA* C. Burge. **PhD thesis, Standford University** (1997)

[7] *Applications of generalized pair hidden Markov models to alignment and gene finding problems* L. Pachter, M. Alexandersson and S. Cawley. **Proceedings of the fifth annual international conference on Computational biology** (2001).

[8] *Applications of generalized pair hidden Markov models to alignment and gene finding problems* S. Cawley and L. Pachter. **Bioinformatics** vol. 19 Suppl. 2 ii36-ii41 (2003).