Journal of Bioinformatics and Computational Biology
Vol. 3, No. 2 (2005) 491–526

Imperial College Press
www.icpress.co.uk

HIDDEN MARKOV MODELS, GRAMMARS, AND BIOLOGY: A TUTORIAL

SHIBAJI MUKHERJEE
Association for Studies in Computational Biology
Kolkata 700 018, India
mshibaji@acm.org

SUSHMITA MITRA*
Machine Intelligence Unit, Indian Statistical Institute
Kolkata 700 108, India
sushmita@isical.ac.in

Received 23 April 2004
1st Revision 2 September 2004
2nd Revision 20 December 2004
3rd Revision 5 January 2004
Accepted 6 January 2005

Biological sequences and structures have been modelled using various machine learning techniques and abstract mathematical concepts. This article surveys methods using Hidden Markov Model and functional grammars for this purpose. We provide a formal introduction to Hidden Markov Model and grammars, stressing on a comprehensive mathematical description of the methods and their natural continuity. The basic algorithms and their application to analyzing biological sequences and modelling structures of bio-molecules like proteins and nucleic acids are discussed. A comparison of the different approaches is discussed, and possible areas of work and problems are highlighted. Related databases and softwares, available on the internet, are also mentioned.

Keywords: Computational biology; machine learning; Hidden Markov Model; stochastic grammars; biological structures.

1. Introduction

Hidden Markov Model (HMM) is a very important methodology for modelling protein structures and sequence analysis.[28] It mostly involves local interaction modelling. Functional grammars provide another important technique typically used for modelling non-local interactions, as in nucleic acids.[71] Higher order grammars, like Graph grammars, have also been applied to biological problems mostly to model cellular and filamentous structures.[25] Other mathematical structures like

*Corresponding author.

knots, geometric curves and categories have been used to model DNA, protein and cell structures.[25,33] Another important area of research in this domain has been the formulation of algorithms mostly based on Dynamic programming.[39,84] The complexity analysis of the problems have also been done, and an important class of problems have been shown to be NP hard. Most algorithms for structure determination are computationally very much intensive, and their porting to massively parallel systems and supercomputers is also an active area of study.[83]

The present article concentrates on providing a review and a tutorial involving two areas, *viz.*, HMM's and Functional grammars. We assume necessary molecular biology background and focus on the mathematical foundation of the formalisms. Attempt is made to provide a comprehensive survey of the field, with relevant reference to applications in the biological domain. The readership that we aim to target in this article consists of biologists and bioinformaticians who are looking for the necessary mathematical background and an introduction to the various modelling techniques so that they can bridge the gap between introductory articles[13] and highly technical expositions.[47] We will mainly focus on the mathematical aspects of the problem in this tutorial and will try to look at the biological problems from an analytical perspective. We have tried to provide easy explanation and relevant biological examples at each stage wherever possible, while including an exhaustive reference on applications of HMM and functional grammars to biological problems. However, due to space limitation, we do not discuss the methodologies or the models in further detail. Our principal focus in this paper, therefore, is on the algorithms and the structure of the mathematical representation of the problem domain.

This paper is divided into two major parts. While Secs. 2–6 deal with HMM, the Secs. 7–11 are concerned with grammar. Section 2 provides a summary of the necessary mathematical results from probability theory from the viewpoint of computational biology. Section 3 attempts to give a concise mathematical introduction to HMM, and discusses possible extensions to the model. Section 4 discusses in concise form the algorithms widely used for HMMs, *viz.*, Expectation Maximization, Viterbi and Forward Backward. Section 5 surveys applications in computational biology, using HMMs, involving sequences and structures. Section 6 provides an introduction to the tools based on the discussed algorithms, and their target biological databases. Section 7 attempts to give a concise mathematical introduction to grammars, and Stochastic Context Free Grammar SCFG in particular. Section 8 compares HMM to SCFG, and discusses how HMMs can be shown as a special case of regular stochastic grammars and functional grammars. Section 9 discusses in concise form the widely used and generalized algorithms for SCFG, like Inside Outside and Cocke Younger Kasami. Section 10 surveys applications in computational biology, using SCFG, particularly involving sequences and structure of RNA. Section 11 provides an introduction to tools based on the discussed algorithms and the RNA databases. Finally, Sec. 12 concludes the article.

2. Basics of Probability

In this section we provide important results and concepts from probability theory, and language grammar that will be used throughout this paper. We assume that the reader has the necessary background in molecular biology.[56]

Let us list the necessary nomenclature and mathematical results of probability calculus that are needed for a study of HMM. We will discuss mainly results from probability theory and language grammar from a computational biology perspective. The detailed analysis of probability theory can be found in the literature.[22,32] We will follow the notations by Koski,[47] as this monograph gives the details of the mathematical formalism of HMM from a computational biology perspective.

Random variables are the building blocks in probability theory, assuming values from an alphabet set or state space. In biology, proteins are represented in terms of the twenty amino acids, while deoxyribonucleic acid (DNA) [ribonucleic acid (RNA)] is decoded in terms of four nucleotides adenine (A), cytosine (C), thymine (T) [uracil (U)] and guanine (G). In this terminology, let S be an alphabet set ($S = A, C, T, G$), with X being a random variable taking values in S. Values taken up by X are generally denoted by x_i and the probability of the event is denoted as $f_X(x_i) = P(X = x_i)$. The whole sequence probability is often denoted as $f_X = (f_X(x_1), f_X(x_2), f_X(x_i), \ldots, f_X(x_L))$, where L is the length of the sequence. The boundary conditions on this probability function are $f_X(x_i) \geq 0$ and $\sum_{i=0}^{i=L} f_X(x_i) = 1$. The notation is subsequently simplified to $f(x_i)$ to eliminate clutter.

Probability of more than one event occurring simultaneously is determined by a joint probability distribution, denoted as $f_{X,Y}(x_i, y_j) = P(X = x_i, Y = y_j)$, and can be easily extended to n events. Conditional probability distribution gives the probability of an event with respect to another, and is defined as

$$f_{X|Y} = \frac{f_{X,Y}(x_i, y_j)}{f_Y(y_j)}. \tag{1}$$

If X and Y are independent random variables, then we have

$$f_{X,Y}(x_i, y_j) = f_X(x_i) \times f_Y(y_j). \tag{2}$$

Bayes' theorem is another fundamental result, which relates posterior and prior probabilities of events. It provides a probability of the cause on the basis of the observed effect, and is expressed as

$$f_{X|Y}(x_i, y_j) = \frac{f_{Y|X}(y_j, x_i) \times f_X(x_i)}{\sum_{i=1}^{i=L} f_{Y|X}(y_j, x_i) \times f_X(x_i)}. \tag{3}$$

Kullback distance is an important measure for comparison of probability distributions f and g, and is expressed as

$$D(f|g) = \sum_{i=1}^{i=L} f(x_i) \log \frac{f(x_i)}{g(x_i)}, \tag{4}$$

where $0 \times \log \frac{0}{g(x_i)} = 0$ and $f(x_i) \times \log \frac{f(x_i)}{0} = \infty$. It can be thought of as the relative distance between two distributions, but it is not a true metric since it is not symmetric and does not satisfy the triangle inequality. It does not obey the axioms of a metric, *viz.*, $d(x, x) \geq 0$, $d(x, y) = d(y, x)$ and $d(x, z) \leq d(x, y) + d(y, z)$.

Kullback distance is essentially a divergence measure of distributions, and is used to measure the log likelihood ratios at DNA splice sites. So if distribution f corresponds to a splice site, and g corresponds to background, then $D(f|g)$ is the average value in the log likelihood ratio when at a splice site. This can be used as a measure of the effectiveness of the log likelihood ratio. We will be using this measure in case of EM algorithms in Sec. 4.1.

3. Hidden Markov Model

An HMM is a powerful statistical technique for modelling signals and sources, and was developed as a part of stochastic calculus. It can also be viewed as a connected graph, with weighted edges representing state transition probabilities and the nodes representing states. Baum *et al.*[8–10] were mostly involved in developing the theory of HMM's. One of the earliest and widely reported applications of HMM was in speech recognition problems.[65] Application of HMM's has been found to be very suitable for a wide range of computational biology problems.[6] We discuss these in detail in Sec. 5.

A biological sequence is represented as transitions of states in an HMM. Here each state corresponds to a certain biological context like exon and intron, and emits a symbol such as a nucleotide or an amino acid. We observe the symbols without knowing which state emitted them. A state has two kinds of parameters, *viz.*, (i) a symbol emission probability which describes the probabilities of the possible outputs from the state, and (ii) a state transition probability which specifies the probability of moving to a new state from the current one. Starting at some initial state, the transitions generate an observed sequence of symbols while moving probabilistically from one state to another until some terminal state is reached, and also emit observable symbols from each state traversed. A sequence of states is represented as a first order Markov Chain. However, because the state sequences are being hidden here, with only the sequence of emitted symbols being observable to the outside world, we term it a *Hidden* Markov Model. The parameters of the HMM are generally learned from training sequences by means of the maximum likelihood or maximum a posterior method to find the model that best fit.

Markov Chain is a stochastic model based on nearest neighbor interactions. It assumes that the state at any instant of time is determined through a local correlation, such that any stochastic event in the chain is determined by the immediate past and does not depend on the whole chain path. The theory of Markov Chains was known in the mathematics community for quite a long time in the form of Random Walks.[23,44] We provide here a brief mathematical account of Markov Chains,[62]

since HMM's are a natural extension of Markov Chains when the observable states are hidden or unknown.

Let us consider a sequence of random variables $X_0, X_1, \ldots, X_n$ having values in an alphabet set $S = s_1, s_2, \ldots, s_n$. This sequence is called a Markov Chain if $\forall n \geq 1$ and $j_0, j_1, \ldots, j_n \in S$, we have

$$P(X_n = j_n \mid X_0 = j_0, \ldots, X_{n-1} = j_{n-1}) = P(X_n = j_n \mid X_{n-1} = j_{n-1}). \qquad (5)$$

This expression states that the probability of a random variable assuming a particular value, given that the other preceding variables have picked up a given set of values, is dependent solely on the last immediate event and not on the whole set. This is called the Markov property.

For example, let us consider a DNA sequence consisting of nucleotides A, C, T, G. Suppose it has already been decoded upto the 100th nucleotide and the 100th element is A. Then the probability that the 101th nucleotide will also be an A is dependent only on the value picked up by the 100th nucleotide, and it is not necessary to consider the values of the other 99 nucleotides. This is a stochastic nearest neighbor interaction model. It is obvious that this sort of model maps local interactions and local dependency, and trivially global dependency if that is a constant. Most linear situations can be modelled using Markov Chains. The only parameter of interest here is the transition probability of states, and if this is stationary then it is called a homogenous Markov Chain. The transition matrix is essentially stochastic.

The HMM literature is very extensive, with papers[13,30,37,48] providing extensive coverage from various viewpoints and Koski's[47] being a rigorous account of HMM's. A problem of fundamental interest is to characterize the relationship between signals and signal sources; *i.e.*, determining what we can learn about a signal from a given model of the source or vice versa. Signal models have been broadly characterized as either discrete or stochastic in nature. In case of discrete models the signal is generally provided as an exact algebraic representation, and the analysis is relatively straightforward. In case of stochastic models, it is assumed that the signal is a parametric random process, with its parameters being well defined. Generally Gaussian, Poisson, Markovian and Hidden Markov distributions of sources are used to model stochastic signals. We now describe simple situations where HMMs can be applied, followed by a description of the mathematical aspects of the model.

A classic description of the situation where HMM is applied can be explained by a coin toss experiment in a restricted situation.[65] Let a person be tossing a coin, observing the outcomes and publishing the information as a series of H and T outcomes to another observer. The observer has neither any knowledge of the experiment, nor is (s)he able to view the outcomes of the experiment (the states of the experiment being hidden), but essentially gets information about a sequence chain. Although the sequence is a Markov Chain, but to the second observer it assumes a different meaning as the process is doubly stochastic; one level of stochasticity being due to the coin toss outcome itself (H or T), and another level being due to

the hidden experimenter, who introduces another level of uncertainty (the states become stochastic). The simplest case can be that the experimenter is tossing one single coin, such that we have only one unknown parameter and it is a two state model. Again the experimenter may be tossing two coins and choosing any one of the coins through some random event, maybe another coin toss. Then the number of unknown parameter is four, with two states each corresponding to a different coin. Similarly, the number of coins in the experiment can be increased and accordingly the number of unknown parameters increases. Note that this stochasticity is introduced only because the experimenter is "Hidden". This situation describes a typical scenario where an HMM is used.

In practical situations the HMM easily translates to speech recognition, computational biology or gesture recognition problems. Figure 1 gives a schematic representation of a HMM. In the figure, S_i are the various states, with S_1 being the start state and S_5 the end state. The a_{ij}'s denote the elements of the transition matrix, which are actually the transition probabilities p_{ij} between states i and j.

We now formally define an HMM. Our base model is a sequence of symbols from an alphabet $O = o_1, o_2, \ldots, o_K$. An HMM is defined by the following three properties.

(I) Hidden Markov Chain: This is a Markov Chain $(X_n)_{n=0}^{\infty}$ having values in a finite state space $S = 1, 2, \ldots, J$. The conditional probabilities are defined as

$$p_{i|j} = P(X_n = j \mid X_{n-1} = i), \quad n \geq 1, \;\; i, j \in S \tag{6}$$

and are assumed to be time-homogenous (no nonlinear dependency on time). The transition matrix is a stochastic matrix defined by $T = (p_{i|j})_{i=1,j=1}^{J,J}$ with the boundary conditions $p_{i|j} \geq 0$, $\sum_{j=1}^{J} \sum_{i=1}^{J} p_{i|j} = 1$. The initial state X_0 is specified by the probability distribution $\pi_j(0) = P(X_0 = j)$, where $\pi(0) = (\pi_1(0), \ldots, \pi_J(0))$.

(II) Observable Random Process: Assume there is a random process $(Y_n)_{n=0}^{\infty}$, with a finite state space $Q = q_1, \ldots, q_K$, such that the two states S and Q may or may

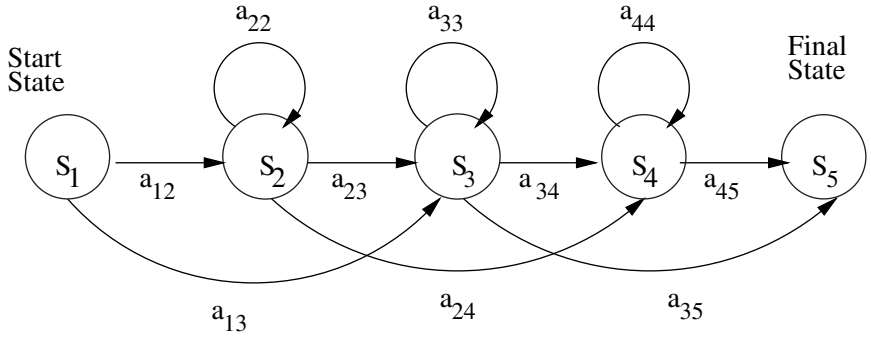

Fig. 1. Hidden Markov Model.

not have the same cardinality. The two random variables X and Y, for any fixed n, are related by a conditional probability distribution

$$e_j(k) = P(Y_n = q_k \mid X_n = j). \tag{7}$$

We define a matrix $E = e_j(k)_{j=1,k=1}^{J,K}$ as the emission probability matrix. This is again a stochastic matrix with $e_j(k) \geq 0$ and $\sum_{k=1}^{K} e_j(k) = 1$.

(III) Conditional Independence: This condition assumes that the emitted symbols are conditionally independent for the state sequence. Given a sequence of states $j_0, j_1, \ldots, j_n$, the probability of the sequence $o_0, o_1, \ldots, o_n$ is mathematically expressed as

$$P(Y_0 = o_0, \ldots, Y_n = o_n \mid X_0 = j_0, \ldots, X_n = j_n, E) = \prod_{l=0}^{n} e_{j_l}(l). \tag{8}$$

Using these assumptions, we can formulate a definition of the joint probability distribution of the symbols and the states $o_0, \ldots, o_n$ and $j_0, \ldots, j_n$ as

$$\begin{aligned} &P(Y_0 = o_0, \ldots, Y_n = o_n, X_0 = j_0, \ldots, X_n = j_n;\ T, E, \pi(0)) \\ &\quad = P(Y_0, \ldots, Y_n \mid X_0, \ldots, X_n, E) \times P(x_0, \ldots, X_n, T, \pi(0)) \\ &\quad = \pi_{j_0}(0) \times \prod_{l=0}^{n} e_{j_l} \times \prod_{l=1}^{n} p_{j_{l-1}|j_l}. \end{aligned} \tag{9}$$

Summing over all possible paths of the state sequence, we get

$$P(Y_0, \ldots, Y_n; T, E, \pi(0)) = \sum_{j_0=1}^{J} \cdots \sum_{j_n=1}^{J} \pi_{j_0}(0) e_{j_0}(0) \prod_{l=1}^{n} p_{j_{l-1}|j_l} e_{j_l}(l). \tag{10}$$

This implies that the finite dimensional distributions of the observable random process are fully specified by the choice of (i) the two stochastic matrices for transition probability and emission probability, and (ii) the initial distribution. So the model can be compactly represented as $\lambda = (T, E, \pi(0))$. Given an observation sequence $\mathbf{o} = o_0, \ldots, o_n$, which is doubly stochastic and Markovian in nature, a complete model can be specified if we know the state transition probability matrix T, state symbol distribution probability matrix E, and initial distribution $\pi(0)$ specifying the state at the start. So for an arbitrary sequence the probability of the sequence o having this structure, given a model λ, is

$$\begin{aligned} P(\mathbf{o}) &= P(Y_0 = o_0, \ldots, Y_n = o_n; \lambda) \\ &= \sum_{j_0=1}^{J} \cdots \sum_{j_n=1}^{J} P(Y_0 = o_0, \ldots, Y_n = o_n, X_0 = j_0, \ldots, X_n = j_n; \lambda), \end{aligned}$$

where

$$\begin{aligned} &P(Y_0 = o_0, \ldots, Y_n = o_n, X_0 = j_0, \ldots, X_n = j_n; \lambda) \\ &\quad = \pi_{j_0}(0) \times \prod_{l=0}^{n} e_{j_l} \times \prod_{l=1}^{n} p_{j_{l-1}|j_l}. \end{aligned} \tag{11}$$

Biological applications of these results are discussed in Sec. 5. Let us now classify the three generic types of problems an HMM can model.[47]

Problem 1: Given an observation sequence and a model, how to efficiently compute the probability of the observation sequence. This is also called the evaluation or scoring problem. Mathematically, it deals with computational complexity. The probability expression involves a summation over $J^{(n+1)}$ possible sequences, and the order of computation is $O(2(n+1)J^{(n+1)})$. This is reduced to a solvable problem by the *Forward Backward* algorithm, which we discuss in Sec. 4.2.

Problem 2: Given an observation sequence and a model, how to compute a state sequence, which best explains the observations. This problem is called the alignment or decoding problem. Mathematically, this reduces to finding an optimal sequence $j_0^*, \ldots, j_n^*$, which maximizes $P(\mathbf{X}, \mathbf{Y}, \lambda)$. This is solved by the *Viterbi* algorithm, which we discuss in Sec. 4.3.

Problem 3: How to adjust the parameters in a given model, so that the conditional probability of observation is maximum. This is called the training or estimation problem. It is not essential to have a straightforward solution for all cases. Generally the Expectation Maximization (EM) algorithm, a variation the *Baum Welch* algorithm, Maximum A Posteriori estimate, and Viterbi training are some of the methods applied.[65] We discuss these algorithms later, in Sec. 4.

In the following section we discuss the relationship between grammars and HMMs. Section 5 directly deals with the application of HMMs to various computational biology problems.

4. Algorithms for HMM

In this section we discuss the mathematical representation of each of the algorithms mentioned in Sec. 3, *viz.*, Expectation Maximization, Forward Backward and Viterbi. We construct a very simple HMM and simulate a detailed numerical calculation for the forward backward algorithm in Sec. 4.2. We provide relevant biological examples, wherever it is simple and straightforward to represent.

Let us consider a five-state HMM, with the begin state being represented as state 0 and the end state as state 5. The other states are indicated as states $1, 2, 3, 4$ respectively, and correspond to possible transition states. The model is similar to that shown in Fig. 1, with the addition of one more state to represent all possible scenarios. We simplify the notations as much as possible neglecting mathematical rigor to make things easier to understand, and depict it in Fig. 2. The figure can be thought of as representing a sequence motif. The state transition probabilities are represented as a_{ij} and the emission probabilities as $e_k(A)$, where a_{ij} represents transition probability from state i to state j and $e_k(A)$ is the probability of emitting character A in state k as explained in Sec. 3. We represent the transition matrix

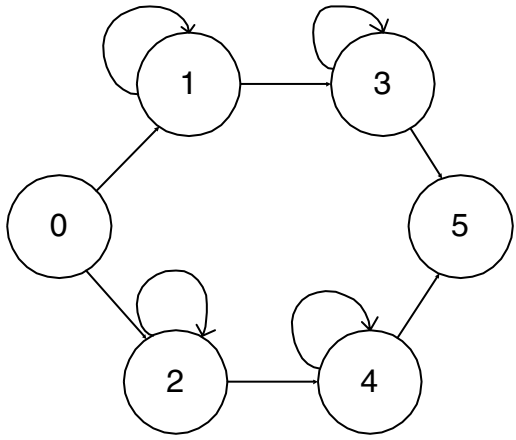

Fig. 2. Example illustrating Hidden Markov Model.

$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}$, having values $\begin{pmatrix} 0.0 & 0.5 & 0.5 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.2 & 0.0 & 0.8 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.6 & 0.0 & 0.2 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.4 & 0.0 & 0.6 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.1 & 0.9 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$ for the corresponding elements. We observe from the value matrix that no self transition is possible in either the end or start states, implying that neither can any state go back to the start state nor can it revert back from the end state. Note that only transitions between nearest states are allowed, and there is neither vertical or diagonal transition nor jumping of the nearest neighbor possible. Self transition in other states is, however, allowed. Although the matrix is not symmetric, but all non nearest neighbor terms are symmetric. Since the matrix contains a row and a column with all zero elements, it is a singular matrix and will always be so unless reverse transitions from end and start states is permissible. Let us represent sample states as follows:

$$\text{State } 1 = (A, C, G, T, 0.4, 0.1, 0.2, 0.3) \quad \text{State } 2 = (A, C, G, T, 0.2, 0.3, 0.3, 0.2)$$
$$\text{State } 3 = (A, C, G, T, 0.4, 0.1, 0.1, 0.4) \quad \text{State } 4 = (A, C, G, T, 0.1, 0.4, 0.4, 0.1).$$

The letters represent the possible emission symbols, while the numbers denote the corresponding probabilities.

4.1. *Expectation maximization*

EM, or Expectation Maximization algorithms,[26] form an important foundation of HMM's, where they are known in a modified form as Baum Welsch algorithm. EM algorithms have been used in statistical genetics for a long time.[53] They generally model situations with hidden variables, particularly in mixture models (parameter driven probabilistic models) and sequence analysis where part of the data is missing

or unobserved. Generally the algorithm consists of two steps: (i) the E (Expectation) step computing the distribution of the hidden variables on the basis of the observed data and the estimated model parameters; and (ii) the M (Maximization) step calculating the optimized values of the parameters. We discuss here the mathematical foundation of EM algorithms and some selected applications. Interested readers may refer to the literature[12,47,61] for a more detailed coverage.

We base the analysis on a mixture model. Let there be two data distributions $\mathbf{y} = (y_1, y_2, y_3, \ldots, y_n)$ and $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$, where y and x are the values of two independent pair of random variables Y, X. Here we consider $\mathbf{x}$ to be the hidden variable with probability $P(X_l = x_j) = \alpha_j$, and $\mathbf{y}$ the observable variable with conditional probability $P(Y_l = y \mid X_l = x_j) = p(y \mid \varphi_j)$ where φ_j are some parameter vectors of a parameter $\theta(\alpha_1, \alpha_2, \ldots, \alpha_L; \varphi_1, \varphi_2, \ldots, \varphi_L)$. The aim of EM algorithm is to estimate θ in the situations where $\mathbf{x}$ is hidden and $\mathbf{y}$ is observable. Assuming the variables to be pairwise independent, we have

$$p(\mathbf{x},\mathbf{y} \mid \theta) = \prod_{l=1}^{l=n} P(Y_l = y_l, X_l = x_{j_l} \mid \theta) = \prod_{l=1}^{l=n} p(y_l \mid \varphi_{j_l}) \times \alpha_{j_l}. \tag{12}$$

Rewriting the results in the framework of marginal distributions, we get for any element in X and Y

$$f(y \mid \theta) = \sum_{j=1}^{j=L} p(X_l = x_j, Y_l = y \mid \theta) = \sum_{j=1}^{j=L} \alpha_j \times p(y \mid \varphi_j). \tag{13}$$

This probability distribution is called a finite mixture, while the distribution over α is called a mixing distribution. The likelihood function for $\mathbf{y}$ with relation to θ is thus

$$p(y \mid \theta) = f(y_1 \mid \theta) \times f(y_2 \mid \theta) \times \cdots \times f(y_n \mid \theta). \tag{14}$$

The Maximum Likelihood estimate or ML is

$$\theta_{ML} = \arg\max_{\theta} p(\mathbf{y} \mid \theta), \tag{15}$$

where the probability or the estimate varies on the parameter θ which is to be modelled. The computation of this estimate is not an exactly solvable algebraic problem, and numeric methods have been constructed to solve it. Using posterior probability we have from Eqs. (12) and (14)

$$p(\mathbf{x} \mid \mathbf{y}, \theta) = \frac{p(\mathbf{x}, \mathbf{y} \mid \theta)}{p(\mathbf{y} \mid \theta)} = \prod_{l=1}^{l=n} \frac{p(y_l \mid \varphi_{j_l}) \times \alpha_{j_l}}{f(y_l \mid \theta)}. \tag{16}$$

Expressing in terms of logarithms

$$\log p(y \mid \theta) = \log p(\mathbf{x}, \mathbf{y} \mid \theta) - \log p(\mathbf{x} \mid \mathbf{y}, \theta). \tag{17}$$

Our target is to compute a lower bound on $\log p(y \mid \theta)$ by estimating on θ_{ML}. Assume that there is an approximation $\theta_a = \left(\alpha_l^a, \alpha_2^a, \ldots, \alpha_L^a; \varphi_1^a, \varphi_2^a, \ldots, \varphi_1^L\right)$, such

that we need to approximate $\theta_a \rightarrow \theta_{ML}$. Normalizing Eq. (17), using the factor $\sum_x p(\mathbf{x} \mid \mathbf{y}, \theta_a)$, we get

$$\log p(y \mid \theta) = \sum_x p(\mathbf{x} \mid \mathbf{y}, \theta_a) \log p(\mathbf{x}, \mathbf{y} \mid \theta) - \sum_x p(\mathbf{x} \mid \mathbf{y}, \theta_a) \log p(\mathbf{x} \mid \mathbf{y}, \theta). \quad (18)$$

Introducing compact notation in terms of an auxiliary function $Q(\theta \mid \theta_a) = \sum_x p(\mathbf{x} \mid \mathbf{y}, \theta_a) \log p(\mathbf{x}, \mathbf{y} \mid \theta)$, we have

$$\begin{aligned} &\log p(y \mid \theta) - \log p(y \mid \theta_a) \\ &\quad = Q(\theta \mid \theta_a) - Q(\theta_a \mid \theta_a) + \sum_x p(\mathbf{x} \mid \mathbf{y}, \theta_a) \log \frac{p(\mathbf{x} \mid \mathbf{y}, \theta_a)}{p(\mathbf{x} \mid \mathbf{y}, \theta)}. \end{aligned} \quad (19)$$

The last term on the *r.h.s.* is the Kullback distance, and hence is ≥ 0. Therefore

$$\log p(y \mid \theta) - \log p(y \mid \theta_a) \geq Q(\theta \mid \theta_a) - Q(\theta_a \mid \theta_a). \quad (20)$$

If we estimate $\theta_{a+1} = \arg\max_\theta Q(\theta \mid \theta_a)$ in Eq. (15) then $\log p(y \mid \theta_{a+1}) \geq \log p(y \mid \theta_a)$ by Eq. (20). Thereby we improve on θ_a because the likelihood is increased. The algorithmic steps can now be stated as follows:

Start: Get an estimate $\theta_a = (\alpha_l^a, \alpha_2^a, \ldots, \alpha_L^a; \varphi_1^a, \varphi_2^a, \ldots, \varphi_1^L)$.

Step E: Calculate the conditional expectation
$Q(\theta \mid \theta_a) = \sum_x p(\mathbf{x} \mid \mathbf{y}, \theta_a) \log p(\mathbf{x}, \mathbf{y} \mid \theta)$.

Step M: Determine $\theta_{a+1} = \arg\max_\theta Q(\theta \mid \theta_a)$.
Let $\theta_{a+1} \rightarrow \theta_a$
Go to Step E.

Convergence of this algorithm has been studied in Boyles *et al.*[15] and Wu *et al.*[86] The expression $Q(\theta \mid \theta_a)$ can be further analyzed to derive cases that lead to unique solutions. Another approach to EM algorithm is in terms of free energy and the Boltzmann-Gibbs distribution,[6] with the function to be optimized being the free energy of the model in terms of a parameter like θ.

There exist various applications of EM algorithms to computational biology, of which we highlight some here. Let us now consider n sequences (say, fragments of DNA sequences) s_l, all of length $N + 1$. Assume that all of them contain some special patterns called motifs (*i.e.*, some particular arrangement of the nucleotides) of length W. Consider that (i) the exact position of the pattern in the fragment is not known, (ii) the pattern may be altered in a sequence, *i.e.*, there are mutations, and (iii) there are no insertions or deletions. All sequences contain a single copy of the motif between two random sequences. We will mathematically model this situation to demonstrate the use of EM algorithm. An extensive survey of pattern detection in bio-sequences is provided in Brazma *et al.*[16] and Rigoutsos *et al.*[66]

Let us represent the position of occurrence of the pattern by a hidden random variable X, which is integer-valued, and the probabilities be denoted as $\alpha_t = P(X = t), t \in (0, 1, \ldots, N - W + 1)$. Consider W independent random variables Y_{t+i}, $i = 0, 1, \ldots, W - 1$, each assuming values in a finite discrete

alphabet set $A = s_1, s_2, s_3, \ldots, s_K$. Here, the s_i's can represent the alphabet set of nucleotides A, C, T, G. Let the output symbol probabilities be denoted by $b_i(s_l) = P(Y_{t+i} = s_l \mid X = t), l = 1, \ldots, K; i = 0, 1, \ldots, W-1$. These symbols correspond to the letters in the sequence. We have

$$P(Y_t = s_{lt}, \ldots, Y_{t+W-1} = s_{lt+W-1} \mid X = t) = \prod_{i=0}^{W-1} b_i(s_{lt+i}). \tag{21}$$

The random sequence $(Y_t, \ldots, Y_{t+W-1})$, along with the probability distribution, is called a motif of length W. The distribution $B_i = b_i(s_1), b_i(s_2), \ldots, b_i(s_K)$ defines a probability distribution for the motifs.

As we have represented the sequence and the motif as a three-part model, *viz.*, one random part followed by a motif and then another random part, therefore we need to calculate the probability for the random parts as well. Let this be given by another probability distribution $p = (p_1, p_2, p_3, \ldots, p_K)$. For the sequence $s = s_{j_0} \cdots s_{j_{t-1}} \mid s_{j_t} \cdots s_{j_{t+W-1}} \mid s_{j_{t+W}} \cdots s_{j_N}$ (the separators denoting the three regions), the probability is computed as

$$P(Y_0^1, \ldots, Y_N^l \mid X^l = t, p, B_i) = \prod_{j=0}^{t-1} p_{lj} \prod_{i=0}^{W-1} b_i(s_{lt+i}) \prod_{j=t+W}^{N} p_{lj}. \tag{22}$$

The available information is (i) a training set of n sequences $s_{(l)}$ $[(l)$ denoting the set $l_i]$ with length $N+1$, and (ii) motifs of length W. The hidden information is the starting position of the occurrence of these motifs. We have a mixture of probabilities

$$p(s_{(l)}) = P\big(Y_0^{(l)} = s_{j_0}^{\,l}, \ldots, Y_N^{(l)} = s_{j_N}^{\,l} \mid X^l = t, p, B_i \times \alpha_t\big). \tag{23}$$

This is a mixture model with the random variables X and Y taking the forms $Y = A \times A \times A \times A \times A \times \cdots \times A$ and $X = \{0, 1, \ldots, N - W + 1\}$.

The problem is to estimate the parameters and optimize them, as in the E and M steps of the algorithm. This is done using the MEME[2,4,18] approach. This works with continuous replacement of motifs after identification, by applying a new mixture model each time. This method has a limitation in the sense that the motifs are not re-estimated. Application of EM algorithm to mixture problems monotonically increases the likelihood function, the algorithm converges to a local maxima of the function in most cases and get stuck there. This is a major problem in applying the algorithm to a multiple component mixture model.

We discuss now a recent application of EM algorithms for the problem of multiple motif discoveries.[14] This demonstrates an improvement in the application of EM algorithms to mixture models, as compared to the MEME approach. The authors apply a recently discovered greedy method of incremental learning for Gaussian mixtures.[57,82] The method fixes a stopping criteria or a limit on the desired number of motifs, and the learning proceeds on an incremental fashion until convergence. The algorithm selects an initialization for the parameters of a new motif by performing a global search over the input sub-strings, combined with a local search

based on partial EM steps for fine tuning of the parameters of the new component. A hierarchical clustering based on *kd*-tree technique[11,81] is employed for partitioning the input dataset of sub-strings, thereby reducing the time complexity of the algorithm. This approach, in contrast to MEME, is able to effectively fit multiple-component mixture models. This is achieved through a combined scheme of global and local search, which overcomes the existing problem of poor initialization of EM that frequently gets stuck on local maxima of the likelihood function. The procedure also does a better exploration of the dataset, resulting in the discovery of larger groups of motifs.

4.2. *Forward backward*

The Forward Backward algorithm is generally applied to the case of scoring a standard HMM. It also forms the mathematical basis for other dynamic programming algorithms used in sequence analysis, and was originally proposed for speech recognition problems.[10] The algorithm is actually a combination of two, *viz.*, the forward and the backward algorithms. It is also referred to, in a variant form, as the Baum Welsch algorithm.[10] The algorithm is generally used to compute the likelihood of a given observed DNA or protein sequence.

Assume we have a model $\lambda = (T, E, \pi(0))$. Our problem is to calculate the simultaneous probability for a sequence of emitted symbols $\mathbf{o} = o_0, o_1, \ldots, o_N$ conditioned on the given model. Using Eq. (11), the probability is expressed as

$$P(Y_0, Y_1, \ldots, Y_n; \lambda) = \sum_{j_0=1}^{J} \cdots \sum_{j_n=1}^{J} \pi_{j_0}(0) \times e_{j_0}(0) \prod_{l=1}^{n} p_{j_{l-1}|j_l} \times e_{j_l}(l). \tag{24}$$

Let us denote this probability by L_N. Note that this expression has an exponential growth of operations in N in the summation. The forward-backward algorithm allows its evaluation in such a way that the computational requirement becomes linear in the sequence length $L + 1$. The idea is based on splitting the expression into a forward and a backward variable, as

$$L_N = P(Y_0, Y_1, \ldots, Y_n; \lambda) = \sum_{j=1}^{J} \alpha_n(j) \times \beta_n(j). \tag{25}$$

The forward variable α is defined as the simultaneous probability of the emitted sequence up to time $n \leq N$ and of the Hidden Markov Chain being in the state j at time n. So α is defined as

$$\alpha_n(j) = P(Y_0 = o_0, Y_1 = o_1, \ldots, Y_n = o_n \mid X_n = j). \tag{26}$$

Similarly the backward variable is defined for the other half of the time slice, as the probability of the emitted subsequence from time $n + 1$ to N conditioned on the model being in state j at time n. Mathematically,

$$\beta_n(j) = P(Y_{n+1} = o_{n+1}, \ldots, Y_N = o_N \mid X_N = j). \tag{27}$$

The idea of the problem is to set up a recursive relationship for the two variables and follow with an iteration to solve the problem in the standard manner of dynamic programming.

The recursion relations are given as

$$\sum_{i=1}^{J} \alpha_n(i) \times p_{i|j} \times e_j(o_{n+1}) = \left[\sum_{i=1}^{J} \alpha_n(i) \times p_{i|j} \right] \times e_j(o_{n+1}) \tag{28}$$

and

$$\beta_n(j) = \sum_{i=1}^{J} e_i(o_{n+1}) \times \beta_{n+1}(i) \times p_{j|i}. \tag{29}$$

The algorithmic steps are pretty simple and Koski[47] provides a detailed description. Both these algorithms scale as $O(n^2)$, thereby overcoming the evaluation problem. The probability of the model being in state i at time t, given an observation sequence o and model λ, is given as

$$P(X_t = i \mid \mathbf{o}, \lambda) = \frac{\alpha_i(t)}{\beta_i(t)}. \tag{30}$$

This reduces the exponential computational complexity of the original problem to one that is linear in sequence length.

Let us now discuss the situation in terms of the HMM model formulated in Sec. 4, particularly with reference to Fig. 2. We are essentially trying to determine *'how likely is a given sequence'* to occur in case of a simple state transition. We illustrate a numerical approach to this problem, by finding the likelihood of a small sequence $TAGA$. The probability of the sequence needs to be calculated, given a model λ. One possibility follows the jumps $0 \rightarrow 1 \rightarrow 3 \rightarrow 5$. The probability is calculated from Eq. (9), using the matrices in Sec. 4, as

$$\begin{aligned} P(TAGA, \lambda) &= a_{01} \times e_1(T) \times a_{11} \times e_1(A) \times a_{13} \times e_3(G) \times a_{33} \times e_3(A) \times a_{35} \\ &= 0.5 \times 0.3 \times 0.2 \times 0.4 \times 0.8 \times 0.3 \times 0.4 \times 0.2 \times 0.6. \end{aligned}$$

Practically we calculate the likelihood of the sequence $TAGA$ by observing that we are trying to find the probability of the event that we are in state five, having observed four symbols. This is possible if we are either in state three and observe four symbols or we are in state four and observe four symbols, and in both cases transit to state five. This is the origin of the recursion, whereby we use the prior states to get the probabilities for the next states and move one step at a time. Using Eqs. (28) and (30), and simplifying them, we have $f_5(4) = f_3(4) \times a_{35} + f_4(4) \times a_{45}$, where $f_A(k)$ denotes state A upon observing k symbols. We know the symbols we are supposed to observe, *viz.* $TAGA$, and use this information and the boundary conditions to recursively calculate the probabilities. Here the boundary conditions are simple, $f_0(0) = 1, f_1(0) = 0, f_5(0) = 0$, with their meanings being self-evident.

A likely sequence of steps is (i) going from state 0 to state 1 and emitting a symbol T, (ii) doing a self transition and emitting a symbol A, (iii) making

a transition to state 3 and emitting a symbol G, and (iv) doing a self transition and emitting a symbol A, before undergoing a transition to state 5. The other likely possibility is (i) going from state 0 to state 2 and emitting a symbol T, (ii) doing a self transition and emitting a symbol A, (iii) making a transition to state 4 and emitting a symbol G, and (iv) doing a self transition and emitting a symbol A, before undergoing a transition to state 5. The functions are evaluated as

$$\begin{aligned} f_1(1) &= e_1(T) \times (f_0(0) \times a_{01} + f_1(0) \times a_{11}), \\ f_2(1) &= e_2(T) \times (f_0(0) \times a_{02} + f_2(0) \times a_{22}), \\ f_1(2) &= e_1(A) \times (f_0(1) \times a_{01} + f_1(1) \times a_{11}), \end{aligned} \tag{31}$$

and so on. The numeric values can be determined from the transition matrix and the emission vectors, which we show as states. The rest of the lengthy calculation can be done by repeating the process while using the recursion relations.

4.3. *Viterbi*

Viterbi algorithm[34] solves the problem of computing the probability of the most likely path in a model λ that generates a string S as well as the path itself. This belongs to a class of dynamic algorithms that solves the problem by backtracking along the state path. It works by gradually replacing calculations over all possible paths with those over the most likely paths associated with each sequence. An initial guess of the model parameters is refined by observation, thereby reducing the errors of fitting the given data using gradient descent for minimizing an error measure.

The Viterbi algorithm can be considered as a special form of the Forward-Backward algorithm, where only the maximum path is considered at each time step instead of all paths. For each state in an execution path, the algorithm computes the forward probability of arriving at that state and the backward probability of generating the final state of the model. This optimization reduces the computational load and allows the recovery of the most likely state sequence. The amount of storage is proportional to the number of states, and the amount of computation to the number of transitions. So this algorithm works well for sequences having sharp peaks, like in the case of modelling protein families. However, for DNA sequences the results are not good because of lack of such sharp peaks. Viterbi learning and algorithm have been extensively used in probabilistic sequence analysis problems and profile HMM's. Computing the Viterbi path of a sequence is generally called the problem of aligning a sequence to a model. Its application to RNA structure prediction is described in Sec. 10.

The algorithm makes a number of assumptions. First, both the observed events and hidden events must be in a sequence. Secondly, these two sequences need to be aligned, and an observed event needs to correspond to exactly one hidden event. Third, computing the most likely hidden sequence up to a certain point t must only depend on the observed event at point t, and the most likely sequence at point $t-1$.

Generally the parameters in Viterbi algorithm come out to be very small, and the calculations are mostly done using logarithms.

The probabilistic information about a given sequence is given by the posterior probability, defined as

$$\widehat{\pi}_j(n \mid N) = P(X_n = j \mid Y_0 = o_0, \ldots, Y_n = o_N), \quad j = 1, \ldots, J. \tag{32}$$

The alignment problem of HMM's can be solved by finding for each $n = 0, 1, 2, \ldots, N$ the value of arg $\max_{1 \le j \le J} \widehat{\pi}_j(n \mid N)$. Let us define

$$\delta_n(j) = \max_{j_0, \ldots, j_{n-1}} P(Y_0 = o_0, \ldots, Y_N = o_n, X_0 = j_0, \ldots, X_n = j). \tag{33}$$

This measures the maximum probability along a single subsequence of states accounting for the first $n + 1 \le N$ emitted symbols. The property of conditional independence of HMM can be applied using this expression, to get a multiplicative score for the sequence. Viterbi algorithm utilizes this idea along with the concept of backtracking from dynamic algorithm, to reconstruct the best possible state sequence from a series of probabilistic observations on a state sequence. The condition is given by the Bellman optimality principle[47]

$$\delta_n(j) = \left[\max_{i=1,\ldots,J} \delta_{n-1}(i) \times p_i \mid j \right] \times e_j(o_n). \tag{34}$$

At each iteration for each state, the transaction from the earlier state with the best score is generated and the product is built up. This procedure yields the subsequence $\delta_n(j)$, termed the *survivor*, and is denoted by $\psi_n(j) = \arg\max_{i=1,\ldots,J} \delta_{n-1}(i) \times p_{i|j}$. Viterbi algorithm provides a recipe to solve this expression.

There are four major steps, consisting of start, recursive, termination and traceback states.[47] The start state is an initialization step. This is followed by a recursive state, where a state is calculated recursively in terms of its previous states and the surviving states are stored to build up a matrix of likely states. The termination state applies the condition to terminate a Viterbi walk. Finally, the traceback step reconstructs the path by adding up all the surviving recursive steps.

A typical example of application of this algorithm is for parsing a query sequence into its component exons and introns. The probability that the model will produce a given sequence is also computed by the Viterbi algorithm. This represents determining the possibility that a given DNA sequence contains a gene.

The Viterbi algorithm, when applied to solve the learning problem of HMM's,[84] is called the Viterbi *Learning* algorithm. In this case, an initial HMM model is chosen, and a probability matrix is constructed for state transitions either from known values or heuristics. The Viterbi algorithm is then used to align the sequence to the HMM. A set of statistical counts of the number of occurrences of states and transition is counted, followed by a relative frequency estimation, and the model is then recursively calibrated. The processes are repeated until the estimates converge. This provides an estimate of the model parameters for the maximum probability

case. The parameters that are estimated include

- $n_i(k)$ = the number of times state i occurs in the alignment of the sequence;
- $n_{i|j}(k)$ = the number of transitions from i to j in the sequence alignment; and
- $m_{j|l}(k)$ = the number of times the symbol o_l is emitted in state j by the sequence.

Then $n_i(k)$, $n_{i|j}(k)$ and $m_{j|l}(k)$ are computed over all possible k, and the relative frequencies are calculated. We refer the reader to a good implementation of this algorithm at *http://coen.boisestate.edu/ssmith/biohw/CompCode/Viterbi.txt*, using Matlab.

5. Application of HMM to Protein Modelling and Sequence Analysis

Application of HMMs to computational biology problems is quite an active field of research and has been mainly concentrated to protein family profiling, protein binding site recognition and gene finding in DNA.[47,51] Baldi and Brunak[6] define three main groups of problems in computational biology for which HMMs have been applied, namely the problem of multiple alignments of DNA sequences, the problem of finding patterns in biological data, and the large set of classification problems. Although HMMs provide a good probabilistic model of sequence and position specific parameters, but the computational complexity of a large scale model becomes very high due to the large number of parameters involved. The hard optimization problem that HMM tries to solve is replaced by an equally hard parameterization problem.

One of the earliest applications was by Churchill,[24] who applied HMM to the modelling of the coding and non-coding regions of DNA. This was extended to DNA segmentation problem for the yeast genome.[64] HMM's have also been applied to EM algorithms for determination of biopolymer sequences and protein binding sites in DNA,[18,55] mapping of genetic linkage map,[52] protein secondary structure prediction,[1,79] and protein modelling.[7,49]

Gene-prediction HMMs model the process of pre-mRNA splicing and protein translation. A DNA sequence is the input and the output is a parse tree of exons and introns on the DNA sequence, from which the protein sequence of the gene can be predicted. This implies a correct labelling of each element in the sequence as belonging to either a coding region, non-coding region or intergenic region. The standard problems of HMM are solved using gene finding programs [see Sec. 6].

Profile HMM,[48] which is a generalization of profile analysis,[38] is a very important example of application of HMM to pattern finding problem in protein families. This model has its basis in the approximate common substring (ACS) problem, which tries to find out approximate common substrings in a set of randomly chosen strings. Biologically, these are the motifs in a sequence family and correspond to the strongly conserved regions. However the motifs may not be exactly identical, and the probability of point (single nucleotide) mutation exists. This model does

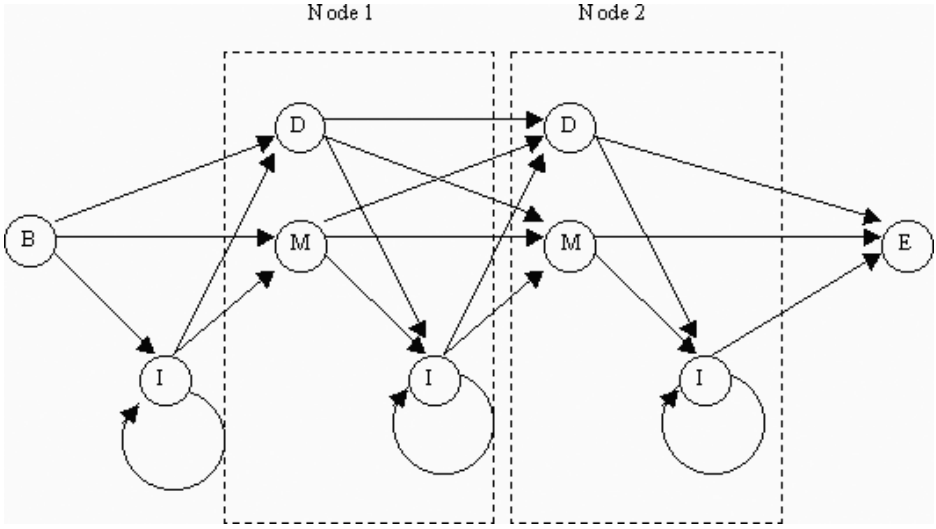

Fig. 3. HMM architecture for detecting motifs in DNA.

not incorporate insertion and deletion, and this is where the HMM provides a better modelling. Biological sequences being very prone to insertion and deletion, the HMM appears to be a better model in mapping motifs in a sequence family.

Extensive studies have been done by Krogh *et al.*,[49] on the globin protein family, for finding motifs by applying HMM. Profile models are designed to detect distant similarities between different DNA sequences along the evolutionary chain. This tries to model the biological proposition of sequence similarity in an evolutionary hierarchy. An example of HMM architecture to model this situation is shown in Fig. 3. It has a simple left-to-right structure in which there is a repetitive set of three states, depicted as match (M), delete (D), and insert (I) in the figure. The match state corresponds to a consensus amino acid for this position in the protein family. The delete state is a non-emitting state, and represents skipping this consensus position in the multiple alignment. The insert state models the insertion of any number of residues after this consensus position. The standard problems of HMM's are overcome mainly by using the profile HMM modelling programs [see Sec. 6].

6. Databases and Softwares for HMM

A library of Profile HMM's[49] (statistical models of the primary structure consensus of a sequence family) is nowadays maintained in Public databases and new sequences are searched against this sequence library. HMM databases are stored as concatenated single HMM files. The most comprehensive library of profile HMM is the PFAM database.[76,77]

Software tools for searching HMM databases and for sequence modelling are freely available. Some of these tools are targeted to biological problems only, while some are libraries to be used with other mathematical modelling tools. A good number of tools for HMM also exist in the speech recognition community. Gollery[36] provides a good survey on HMM databases.

HMMER[29] is a freely distributable implementation of profile HMM software for protein sequence analysis. The current version is HMMER 2.3.2 (3 Oct 2003) and runs on Unix, Macintosh and Windows. Sequence Alignment and Modelling System (SAM) is a collection of software tools for linear HMM modelling (*http://www.cse.ucsc.edu/research/compbio/sam.html*). SAM runs essentially on Unix, and the latest version SAM 3.4 is also freely downloadable.

Meta-MEME[3,5] is a software toolkit for building and using motif-based HMMs of DNA and proteins. The input to Meta-MEME is a set of similar protein sequences, as well as a set of motif models discovered by MEME. Meta-MEME combines these models into a single, motif-based HMM, and uses this to produce multiple alignment of the original set of sequences and to search a sequence database for homologs. The HMM's generated by Meta-MEME's differ from those produced by SAM and HMMER in the sense that the former are motif-based. The toolkit consists of five primary programs for doing five different types of jobs. There are other utility programs, including a converter to change a Meta-MEME linear HMM into an HMMER format. Meta-MEME has been tested on SunOS, Solaris, DEC Alpha and SGI Irix systems. All code is written in ANSI C and can be easily ported to other systems.

Matlab is a versatile mathematical modelling tool, and there exist various extensible libraries that can be added to it as toolkits. A freely available toolbox with Matlab (*http://www.ai.mit.edu/∼murphyk/Software/HMM/hmm.html*) supports inference and learning for HMMs with discrete, Gaussian, or mixture outputs. The Gaussians can be full, diagonal, or spherical (isotropic). It handles discrete inputs, while the inference routines support filtering, smoothing, and fixed-lag smoothing.

The HTK toolkit (*http://htk.eng.cam.ac.uk/*) is also a useful tool. Although the code is copyrighted by Microsoft, the toolkit is freely downloadable, the latest version being 3.2.1. HTK toolkit consists of a set of library modules and tools available in C source form. The distributed version of HTK works on Solaris, IRIX, HPUX, Linux, Windows 2000 and NT. The tools provide facilities for speech analysis, HMM training, testing and results analysis. The software supports HMM's using both continuous density mixture Gaussians and discrete distributions, and can be used to build complex HMM systems. The toolkit is primarily targeted to the speech recognition problem domain, but has also been used for sequence analysis.

GENSCAN is another HMM based sequence analysis tool that is used for the identification of complete exon/intron structures in genomic DNA's.[17] The GENSCAN Database and webserver are now maintained at MIT

(*http://genes.mit.edu/GENSCAN.html*), and it is freely available for academic use. Executables are currently available for the following Unix platforms: Sun/Solaris, SGI/Irix, DEC/Tru64 and Intel/Linux.

7. Preliminaries from Grammars

All formal languages have a grammar, which defines the production rules of the language. Language grammars have been applied to computational biology,[6] particularly to the RNA folding problem. The whole biochemical behavior of a protein depends upon the three dimensional configuration it converges to, taking into account local interactions between atoms and molecules within itself as well as in the surrounding media. This is also termed protein folding, and involves the 3D *tertiary* structure. While the *primary* structure is represented as 1D sequence of amino acids, the *secondary* structure is typically 2D.

Given a defined alphabet with a set of some predefined symbols, and a set of production rules defining how the alphabets in the language combine, one can generate all possible expressions supported by the language. Thus expressions not obeying the legal rules of the language can be discarded. Grammars can be applied to RNA's and DNA's, because they have a well-defined alphabet consisting of the four nucleotides. The possible combinations observed can give us an idea of the production rules in the language of the nucleotides.

Alphabets are defined as a finite, non-empty set of symbols usually written as Σ. An example may be the set of binary alphabets $\Sigma = \{1, 0\}$ or the set of nucleotides $\Sigma = \{A, C, T, G\}$. A combination of these alphabets is called a string or word, like 001111 is a string for the binary alphabets and $AACTGGA$ is a string for the nucleotides. The empty string is defined as a string with no symbols and is denoted by ε. We denote Σ^* as the set of all possible strings from the set Σ, and this is generally not finite.

A language L is defined as a subset of the set Σ^*. Any language follows a set of rules for production of strings, by concatenating the alphabets in the set. The production rules give us the power to parse a string and check whether or not it is a valid member of the language.

We provide here a brief introduction to formal language theory and types of grammars. Hopcroft *et al.*[41] and Harrison[40] give a good coverage on the subject. We will use results from computational linguistics, as discussed in Robert[69] and Wetherell.[85]

A formal grammar for a language is usually defined as a four tuple $G = (V, T, R, S)$. Here V represents the set of variables, also called non-terminals or syntactic categories, and each variable represents a language or a set of strings. T denotes the terminals or the finite set of symbols that forms the string. R represents the production rules that give a recursive definition of the language, with the production symbol being $\rightarrow$. S is the start symbol, which is a variable that represents the language being defined. Productions are further defined as consisting of a

variable called head, and a string of zero or one terminal symbols called body. Let us consider a simple example.

Palindromes are strings that read same from both ends, such as $AATAA$ or $TTAATT$, etc. Palindromes have interesting applications in RNA structure determination. Let us consider the alphabet set $\{A, T\}$. Denoting the palindrome by X, the production rules (R) will be

$$X \to \varepsilon \,|\, A \,|\, T \,|\, AXA \,|\, TXT.$$

The grammar G is $G = (\{X\}, \{A, T\}, R, X)$. Any language $L = L(G)$ is said to be generated by the grammar G.

Chomsky[20,21] has given a general theory of grammar classification called the Chomsky hierarchy. Grammars are classified into four categories namely regular grammars (RG), context free grammars (CFG), context sensitive grammars (CSG) and unrestricted or phrase structure grammar (REG). Let any capital letter like A, B denote nonterminals, small letters like a, b represent terminals, and Greek letters like β, δ denote any string of terminals or non-terminals including the null string. Here we have $V = \{A, B\}$ and $T = \{a, b\}$.

RG's have production rules of the form $A \to aA$ or $A \to a$, which means that one can go from a nonterminal to a single letter or a single letter followed by a variable. So strings can grow only in one direction. Production rules of CFG's are of the form $A \to \beta$. A typical example of CFG's is the Palindromes.

The term "context-free" comes from the feature that the variable A can always be replaced by a, in no matter what context it occurs. CFG's are important because they are powerful enough to describe the syntax of programming languages, and almost all programming languages are defined via CFG's. These are also simple enough to allow the construction of efficient parsing algorithms which, for a given string, determine whether and how it can be generated from the grammar. They can be typically expressed in *Chomsky Normal Form* (CNF). Because of the simple form of production rules in CNF grammars, this normal form has both theoretical and practical implications. For instance, given a CFG, one can use the CNF to construct a polynomial-time algorithm,[a] which decides whether a given string is in the language represented by that grammar or not.

CSG's have productions of the form $\beta \mathrm{A} \delta \to \beta \gamma \delta$. If S is the start symbol then the rule $S \to \varepsilon$ is allowed, provided S does not occur on the right hand side of any other production rule. A typical example is copy language, of the form $xxyzxxyz$, where a string is split into two substrings which are exact copies of each other. REG's have productions of the form $\beta \mathrm{A} \delta \to \gamma$.

RG's generally show short range dependencies and hence are good models for HMM's; CFG's represent nested and long range dependencies and hence can model RNA's; CSG's show crossing dependencies while REG's demonstrate all type of dependencies. Figure 4 summarizes these relationships.

[a]This is the CYK algorithm used in RNA parsing, which we discuss in Sec. 9.2.

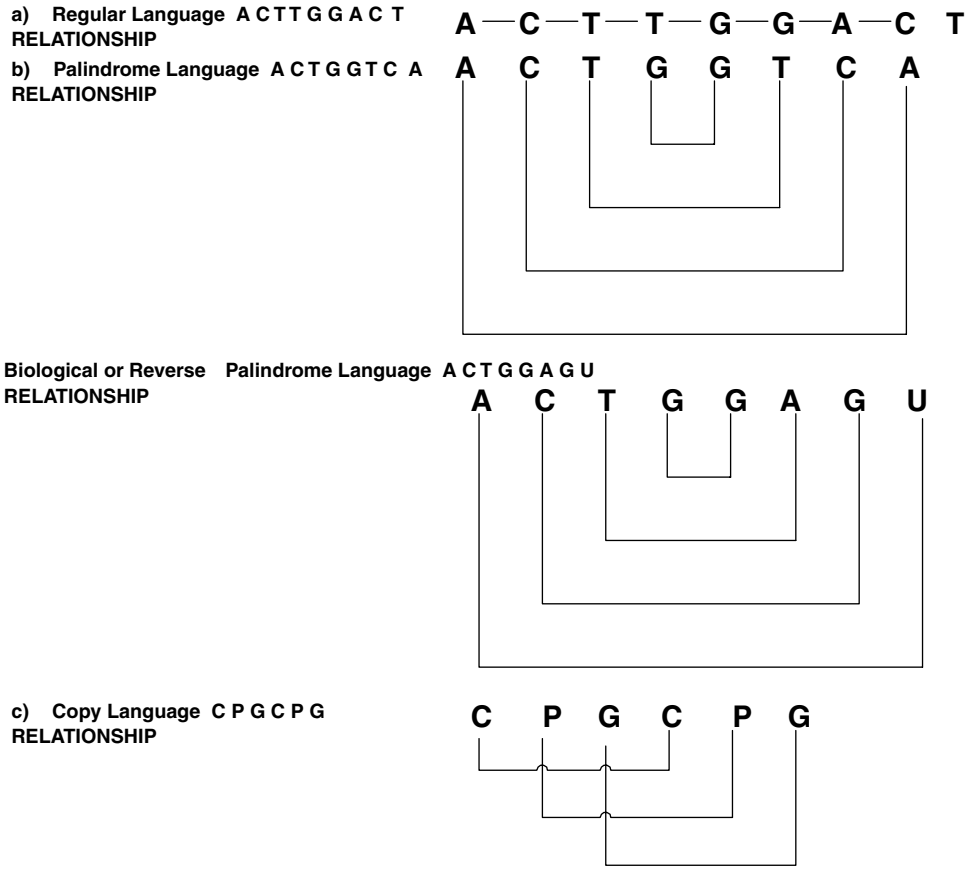

Fig. 4. Relationship rules for (a) Regular, (b) Palindrome, and (c) Copy language grammars.

Let us consider the sequence of events of a regular grammar generating a string. Suppose we need to generate the string GCGCGCTG. The production rules will be

$$
\begin{aligned}
S &\to gA \\
A &\to cB \\
B &\to gC \\
C &\to cD \\
D &\to gE \\
E &\to gF \\
F &\to cG \mid aD \mid cD \\
G &\to tH \\
H &\to g.
\end{aligned}
$$

Proceeding from the start symbol we generate

$$gA \rightarrow gcB \rightarrow gcgC \rightarrow gcgcD \rightarrow gcgcgE \rightarrow gcgcggF.$$

There are three possible expansions. Considering the first one, we get

$$gcgcggF \rightarrow gcgcggcG \rightarrow gcgcgctH \rightarrow gcgcgctg.$$

This is a valid string in the language defined by the above production rule. Considering the second possibility, we generate

$$gcgcggF \rightarrow gcgcggaD \rightarrow gcgcggagE \rightarrow gcgcggaggF$$
$$\rightarrow gcgcggaggcG \rightarrow gcgcggaggctH \rightarrow gcgcggaggctg.$$

Similar patterns will be generated for the other cases. Diagrammatic representation of derivation is often provided graphically, in terms of a construct called *parse tree*. A derivation "$\Rightarrow$" represents zero or more derivation steps "$\rightarrow$", and they can be leftmost or rightmost derivations depending upon whether one always replaces the leftmost or the rightmost variables by one of its production bodies.

A parse tree shows how the symbols of a terminal string are grouped into substrings belonging to the language of the grammar. Let $G = (V, T, R, S)$ be a grammar. The parse trees for G are trees with the following conditions:

(1) Each interior node is labelled by a variable in V.
(2) Each leaf is labelled by either a variable, a terminal or ε. However if the leaf is labelled by ε, then it must be the only child of its parent.
(3) If an interior node is labelled Y and its children are labelled $X_1, X_2, X_3, \ldots, X_k$ from left, then $Y \rightarrow X_1X_2X_3 \cdots X_k$ is a production in R.

The yield of a parse tree is a string, that we get by concatenating its leaves by looking from the left. This is a terminal string such that all its leaves are either labelled with a terminal or ε. Parse trees are very important for RNA's and this is discussed in Sec. 10. Algorithms can be constructed to solve various categories of problems with parse trees. However, all the algorithms do not have exact or finite solutions.

8. Grammars and HMMs

Grammars constitute a very natural model for biological sequence analysis, as they can be very easily represented as strings. HMM's are mainly restricted to modelling local interactions and cannot map long range correlations. Grammars provide the formalism for doing that. Although it is possible in principle to construct any sequence from the alphabet set, yet nature allows only a reduced set and grammars provide rules for these allowed sentences from the base alphabets. Searls made an extensive study of this formalism,[74] by mainly working on a variation of definite clause named *string variable grammar*.[75] A parser called GenLang has been developed for the grammar, and results on the parsing of Eukaryotic protein encoding genes have been reported.[27]

Table 1. Comparison of HMM and SCFG.

HMM	CFG
Hidden States	Non-Terminals
Transition Matrix	Rewriting Rules
Emission Matrix	Terminals
Probabilities	Probabilities

Generally palindrome grammars are applied for modelling the structure in RNA palindromes. Recursive palindromes are used to map repeats in DNA, and secondary stem structure of RNA.[56] Context sensitive copy languages are used to model DNA direct repeats. This relationship is depicted in Fig. 4.

We will not discuss the string variable grammar formalism any further, but concentrate on stochastic context free grammars (SCFG's). These are a natural extension of HMM's for higher order correlations in sequences. HMM's can be viewed as a stochastic version of regular languages, and SCFG's as a stochastic version of context-free languages.[42] Table 1 lists an explicit mapping between HMM's and SCFG's. SCFG's have been mainly used to model nested correlation in RNA secondary structure. One of the main problem of using SCFG is the absence of efficient algorithms and the intense computational requirements of the existing ones. An application of SCFG's to RNA structure prediction is described in detail in Sec. 10.

Let G be an SCFG in which all production rules $a \rightarrow b$ are assigned a probability $P_G(a \rightarrow b)$, such that the sum of probabilities of all possible production rules from any non-terminal is 1. The corresponding stochastic grammar defines a probability distribution over the set of finite strings over the finite alphabet Σ. The probability $P_G(s)$ of G generating string s is the sum of the probabilities of all possible derivation of s from a start symbol. The probability of a derivation is given by the product of the probabilities of all the derivation rules in the derivation sequence. The corresponding SCFG can be transformed to an equivalent SCFG in CNF. The problem of finding $P_G(s)$ can be solved by the Inside Outside algorithm, while the most likely parse of s in G is determined by the Cocke–Younger–Kasami (CYK) algorithm. These are explained in Secs. 9.1 and 9.2.

SCFG's generally solve three classes of problems, *viz.*,

Alignment: Given a parameterized SCFG, what is the optimal alignment of a sequence to it.
Scoring: Given a parameterized SCFG, what is the sequence of that model.
Training: Given a set of sequences, how to estimate the probability parameters of an SCFG.

Algorithms for HMMs and grammars have a natural relationship to each other in terms of complexity and methodology. Considering sequences having average

Table 2. Comparison of algorithms for HMM and SCFG.

Problem	HMM	CFG
Optimal Alignment	Viterbi	CYK
EM Parameter Estimation	Forward Backward	Inside Outside
$P(s \mid \lambda)$	Forward	Inside

observed length N and number of different non-terminals M, generally the memory complexity for HMM is $O(MN)$ and for SCFG is $O(MN^2)$. The corresponding time complexities are $O(M^2N)$ and $O(M^3N^3)$ respectively. However both cases involve polynomially bound algorithms, and hence are solvable. For the SCFG, the alignment problem for SCFG is modelled using the CYK algorithm, the scoring problem is handled using the Inside-Outside algorithm, and the training problem is modelled using the EM algorithm. Table 2 lists the maps between different algorithms for the HMM and SCFG.

9. Algorithms for Grammars

In this section, we outline two of the widely used and generalized algorithms for SCFG's, *viz.*, Inside Outside and Cocke Younger Kasami.

9.1. *Inside outside*

The most popular algorithms for the estimation of the probabilities of a context free grammar are the Inside Outside algorithm and the Viterbi algorithm, both employing Maximum Likelihood approaches. The difference between the logarithm of the likelihood of a string and the logarithm of the likelihood of the most probable parse of a string is upper bounded linearly by the length of the string and the logarithm of the number of non-terminal symbols.[73]

The use of the Inside Outside algorithm for the estimation of the probability distributions of stochastic context free grammars in language modelling is restricted, due to the time complexity per iteration and the large number of iterations that are needed to converge.[54] Its application to RNA structure prediction is described in Sec. 10.

Let us now develop the steps of this algorithm. We will consider a CNF SCFG with N different nonterminals $S_1, \ldots, S_N$, where the start terminal is denoted by S_1. The production rules are of the form $W_x \rightarrow W_y W_z$ and $W_x \rightarrow a$, where a is a terminal symbol and W_i's are nonterminals. Using Eqs. (6)–(7), the transition and emission probabilities for these productions are $p_x(y, z)$ and $e_x(a)$. We consider the sequence $\mathbf{o} = (o_1, o_2, \ldots, o_L)$.

The Inside algorithm calculates the probability of a sequence defined by an SCFG. The probability $\alpha(i, j, x)$ of a parse subtree, rooted at the nonterminal S_x, is calculated for subsequence $o_i, \ldots, o_j$ over all x, i, j. This involves walking along

an $L \times L \times N$ dynamical programming matrix following standard procedures. The computational complexity is $O(L^3N^3)$ in time and $O(L^2N)$ in space. The steps of this algorithm are similar to the Viterbi algorithm of Sec. 4.3, and consist of similar stages involving initialization, recursion and termination.

The Outside algorithm does the same thing from the other end of the sequence. Here we consider the probability β of a complete parse tree rooted at the start nonterminal for the complete sequence o, excluding all parse subtrees for the sequence $o_1, o_2, \ldots, o_j$ rooted at the nonterminal W_x over all x, i, j. The Outside algorithm recursively walks its way inward from the largest excluded subsequence, while the Inside algorithm walks its way outward from the smallest subsequence. The complexity orders are the same as that in the Inside algorithm. The Inside and Outside variables can be used to re-estimate the probability parameters of an SCFG by the EM algorithm.[54]

9.2. *Cocke–Younger–Kasami*

The CYK algorithm[43] deals with the parsing problem for CFG's. It determines whether a given string can be generated by a given context-free grammar and, if so, how it can be generated. The standard version of CYK can only recognize languages defined by context-free grammars in CNF. Although CYK can be extended to parse grammars that are not in CNF in some cases however, the algorithm then becomes much more complicated.[19] The application of CYK algorithm to RNA structure prediction is highlighted in Sec. 10. Now we describe the CYK algorithm steps for a CFG in CNF.

Let G be a CFG in CNF, and L be the language generated by G. Let n be the number of alphabets or grammar symbols of G, denoted as $x_1, x_2, \ldots, x_m$. Assume that the start symbol is x_1, where $x_1, \ldots, x_r$ are variables, and that $x_{r+1}, \ldots, x_m$ are the terminals. Consider a string S and let $s[j, d]$ be a substring of length l starting from the jth symbol of S. Let $B[i, j, d]$ be the boolean array element, where x_i derives $s[j, d]$. The steps of the CYK algorithm are as follows:

Step 1. Initialize all $B[i, j, d]$ to false.

Step 2. *For* all i from $r + 1$ to m and *For* all j from 1 to n, *If* $x_i = s[j, 1]$ *Then* assign $B[i, j, 1]$ to be true.

Step 3. *For* each production of the form $x_k \rightarrow x_i$, where k is between 1 and j (*i.e.*, x_k is a variable) and i is between $r+1$ and m (*i.e.*, x_i is a terminal), *For* each j from 1 to n *If* $B[i, j, 1]$ is true *Then* assign $B[k, j, 1]$ to be true.

Step 4. Execute the following for all d, starting with $d = 2$ and ending with $d = n$.

Step 4.1. Execute the following for every production of the form $x_k \rightarrow x_k x_q$.

Step 4.2. *For* all j from 1 to $n - d + 1$ and *For* all s from $j + 1$ to $j + d - 1$ *If* $B[k, j, s - j]$ and $P[q, s, d + j - s]$ are true *Then* assign $B[i, j, d]$ to true.

Step 5. *If* $B[1, 1, n]$ is true *Then* return Yes *Else* return No. If response is Yes then the string is a member of the language, otherwise it is not.

The worst case asymptotic time complexity of CYK is $\Theta(n^3)$, where n is the length of the parsed string, thereby making it the most efficient algorithm for recognizing a context-free language.

10. Application of SCFG to RNA Structure Prediction

The main domains of application of stochastic grammars in Bioinformatics include protein secondary structure prediction,[35,60] RNA secondary structure prediction,[45,68,72] and gene finding.[17,50,58] In this section, we will mainly consider the problem of RNA secondary structure modelling. These secondary structures are formed by creation of hydrogen bonds between donor and acceptor sites on the nucleotides A, C, G, U. The complementary bases, $C - G$ and $A - U$ form stable base pairs, while in the weaker $G - U$ pair, the bases bond in a skewed fashion. These constitute the canonical base pairs. There are other non-canonical base pairs as well.

Prediction of secondary structure is based upon two energy assumptions, *viz.*, that (i) the total free energy of a given secondary structure is the sum of the free energies of its loops, and (ii) the thermodynamic free energy of one loop is independent of the free energies of all the other loops in the structure. The secondary structure with the lowest free energy is always the most stable one. RNA also forms pseudoknots,[59] but these cannot be inferred from energy calculations. RNA secondary structures show long range correlation and can be modelled well with SCFG.[80] Figure 5 depicts the various structural elements in RNA.[b]

Many computational models have been developed for predicting RNA structures based on energy calculations and stochastic grammars. However, it is still not known how to assign energies to the loops created by pseudoknots, and dynamic programming methods that compute minimum energy structures break down. Covariance methods, on the other hand, are able to predict pseudoknots from aligned, homologous RNA sequences.

A standard grammar for RNA will be based on the four alphabets set (A, C, G, U), where X represents any nonterminal, with the following productions.

- $X \rightarrow XX$, describing the branched secondary structure,
- $X \rightarrow aXa$, describing the base pairing in RNA,
- $X \rightarrow aX$, $X \rightarrow X$, describing multiple alignments, and
- $X \rightarrow a$.

Generally, SCFG-based algorithms are computationally intensive, with structural alignment of an RNA to a sequence being $O(N^3)$ on memory and $O(N^4)$ on time for a sequence of length L, as compared to $O(N^2)$ requirement for standard sequence alignment algorithms. Dynamic algorithms for structure calculations based on energy principles have also been developed.[63,87]

[b] *http://www.bioinfo.rpi.edu/∼zukerm/Bio-5495/RNAfold-html/node2.html.*

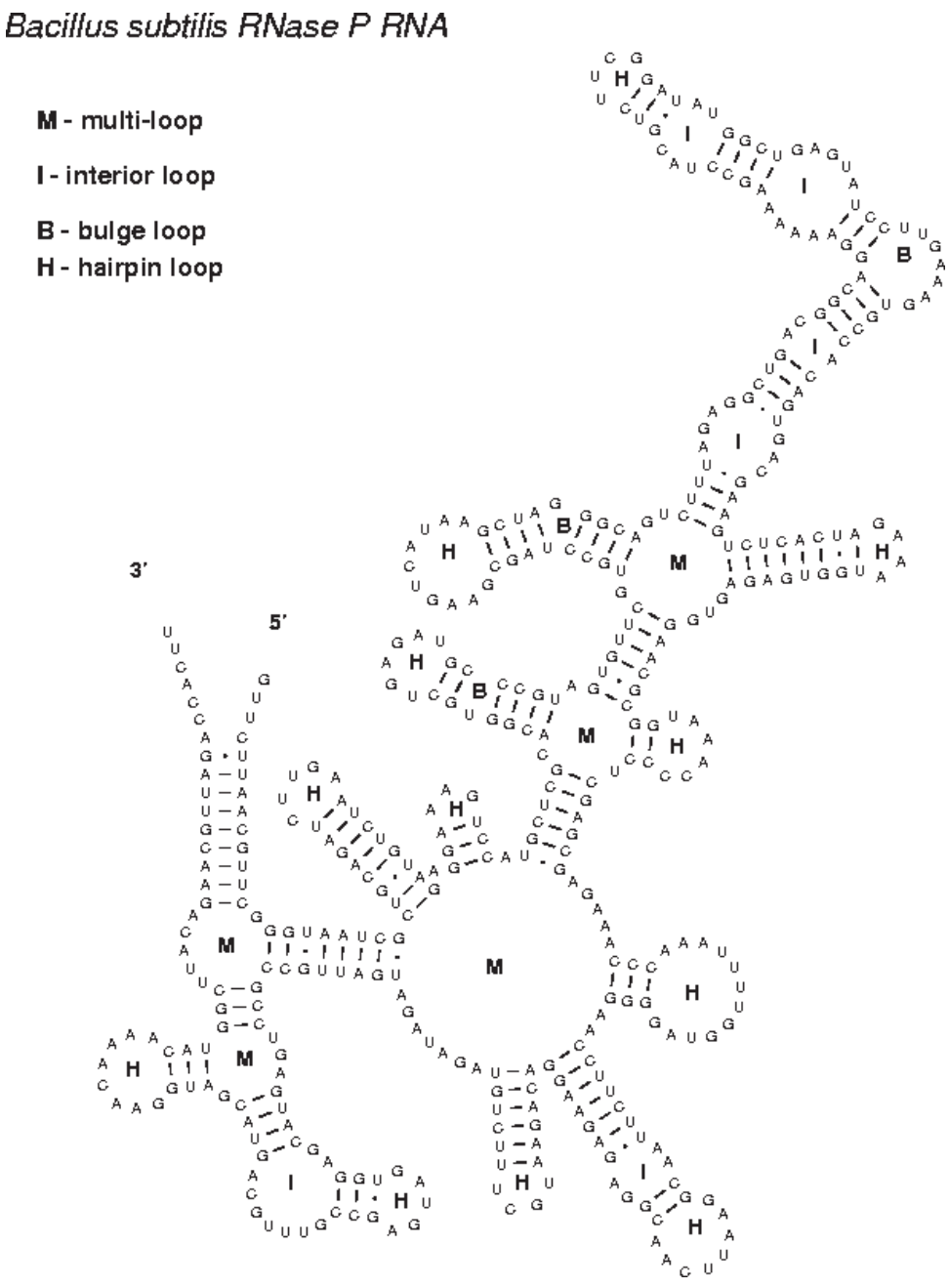

Fig. 5. RNA structure.

Knudsen and Hein[45] developed an efficient algorithm, termed KH-99 for structural prediction in RNA sequences. The algorithm uses an SCFG to produce a prior probability distribution of RNA structures. The Inside Outside algorithm described in Sec. 9.1 is used to calculate the posterior probability of the structure given an alignment and a phylogenetic tree. This probability is based on individual probabilities for alignment columns and pairs of columns for base pairs. The most likely structure is found by using the CYK algorithm of Sec. 9.2. This algorithm is mostly useful for a limited number of sequences, due to its large computation time and problems with gaps. The authors have further extended the algorithm by treating gaps as unknown nucleotides, and reported an improved version called Pfold.[46] This algorithm improves on the KH-99 in terms of speed, robustness, computing efficiency and treatment of gaps.

Sakakibara *et al.* have done important work in applying SCFG to RNA structure prediction,[72] involving both primary and secondary structures. A generalized version of the Forward Backward algorithm (Sec. 4.2), based on tree grammars, is used to train unaligned sequences. This gives a better performance than the standard Inside Outside algorithm of Sec. 9.1. The model is tested on tRNA and gives good results in predicting secondary structures of new sequences.

Pseudoknots in RNA cannot be modelled by a single context free grammar.[68,78] However, formal transformational grammar and other algorithms for modelling the situation have been developed.[31,67] Covariance model, involving a probabilistic approach, is used to construct the secondary structure and primary sequence consensus of RNA's from existing aligned or unaligned RNA sequences. The model is based on an ordered tree, which can capture all the pairwise interactions of an RNA secondary structure. But non-nested pairs like pseudoknots or base triples cannot be modelled. It is actually a generalization of HMM, that is similarly described by a set of states, and a set of emission and transition probabilities. The basic problem of aligning an RNA sequence is handled using dynamic programming algorithms. This is similar to the Inside Outside algorithm with the Viterbi assumption (Sec. 4.3), that the probability of the model emitting the sequence is approximately equal to the probability of the single best alignment of the model to the sequence.

RNA structure modelling using SCFG has predicted important results, but it has its limitations in terms of computing power, size of sequences to be compared and the type of structures to be predicted. Application of higher order grammars, like graph grammar, will probably be able to better model long-range correlations.

11. Databases and Softwares for SCFG

Software for SCFG are very few in numbers and have been mostly developed privately in laboratories. All the SCFG tools that are available in the bioinformatics domain are targeted to RNA secondary structure prediction. A comparative analysis of the various grammars has been recently done.[70] A good

listing of various types of softwares available for SCFG modelling is provided in *http://www.isp.pitt.edu/information/toolboxes.html*.

RNA modelling, in comparison to that for HMM, is considerably underdeveloped. This is mostly because the algorithms are much more difficult and their convergence properties are not good. It requires huge amount of computational resources to calculate RNA folding, and is generally done using supercomputers. RNA databases are also not that widely available or maintained like (say) the gene or the protein databases, since wet lab work with RNA is comparatively a more difficult subject. The best place to search for information about RNA databases and software is RNA world at IMB, Jena (*http://www.imb-jena.de/RNA.html*). Some of the RNA databases include RNase P sequence/structure database at University of Indiana, Group I intron structures at University of Colorado, rRNA WWW server at University of Antwerp, RDP, the Ribosomal Database Project at Michigan State University, 16s RNA structures at University of Colorado, 23s RNA structures at University of Colorado, RNA editing website at UCLA, Physarum mitochondrial RNA editing at University of Texas at Dallas, and RNA Secondary Structures at the Gutell Lab. at University of Texas, Austin.

A comprehensive package of softwares for modeling of RNA (*http://www.genetics.wustl.edu/eddy/software/*) has been developed by Eddy *et al.* at Washington University. The packages available include ATV, RESEARCH, Infernal, TRANASCAN-SE, QRNA, RNABOB, PKNOTS and NCRNASCAN. All these softwares are available free of cost and are downloadable. The packages mostly run on UNIX and Linux environments. They are mostly C library routines, which can be used as plugins in structure manipulation and prediction algorithms. These algorithms use SCFG and covariance model.

Another group of packages is based on maintaining an energy-based RNA folding server (*http://www.ibc.wustl.edu/∼zuker/rna/form1.cgi*). Folding of RNA is done using a related method that utilizes thermodynamic information, instead of base identity statistics.

12. Conclusion

The aim of this tutorial has been to provide an insight into the use of HMM's and Grammars to biological problems, mostly for sequence analysis, pattern discovery and structure modelling. We have tried to show how the grammar formalism is a natural extension of the HMM, for cases where stochasticity is introduced in the process. Biological problems have been shown to follow both the models, depending on the variables in the problem. Algorithms for solving these have been discussed, along with problems regarding exact solutions and their approximations.

There remain many open problems, like the incorporation of structural information into HMM formalism and construction of complex models. The biological problems, handled in the existing framework, are mostly simple in nature and a more precise modelling can be done using extensions of HMM's.

Grammars have been shown to be powerful generalizations of HMM's and provided good results for RNA structures. However, pseudoknots and other such structures have no known explanation in this regular framework. Different extensions proposed in literature generate approximate solutions to this problem. Higher order mapping in the form of Graph grammars can also be a very important field of investigation. The graph grammar formalism may provide clues to problems in long range correlation, as well as higher order structures in proteins and other biological structures.

An extensive amount of research has been done in the field of computational analysis and approximate solutions for algorithms, but much more work is still needed before the huge biological data sets can be properly mapped. Biological data analysis for pattern search will also benefit from discovery of long range correlation mechanism, thereby throwing new light on unknown structures and behavioral patterns.

We hope this tutorial will provide biologists a rigorous and complete introduction to HMM's and Grammars, while giving a concise idea of applying machine learning techniques to biological problems.

Acknowledgement

The authors gratefully acknowledge the anonymous referees, whose suggestions definitely helped improve the quality of this survey.

References

1. Asai K, Hayamizu S, Onizuka K, HMM with protein structure grammar, *Proceedings of the Hawaii International Conference on System Sciences*, Los Alamitos, CA, Vol. 84, pp. 783–791 IEEE, Computer Society Press, 1993.
2. Bailey T, *Discovering Motifs in DNA and protein sequences: The Approximate Common Substring Problem*, Ph.D. thesis, Department of Computer Science and Engineering, University of California, San Diego, 1995.
3. Bailey TL, Elkan C, Fitting a mixture model by expectation maximization to discover motifs in biopolymers, *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, Menlo Park, CA, pp. 28–36, AAAI Press, 1994.
4. ——, Unsupervised learning of multiple motifs in biopolymers using EM, *Machine Learning* **21**:51–80, 1995.
5. Bailey TL, Gribskov M, Combining evidence using p-values: Application to sequence homology searches, *Bioinformatics* **14**:48–54, 1998.
6. Baldi P, Brunak S, Bioinformatics the machine learning approach, 2nd ed., MIT Press, Boston, 2001.
7. Baldi P, Chauvin Y, Hunkapillar T, McClure M, Hidden Markov models of biological primary sequence information, *Proceedings of the National Academy of Sciences*, *USA* **91**:1059–1063, 1994.
8. Baum LE, An inequality and associated maximization techniques in statistical estimation for probabilistic functions of Markov processes, *Inequalities* **3**:1–8, 1972.

9. Baum LE, Petrie T, Statistical inference for probabilistic functions of finite state Markov chains, *Ann Math Stat* **37**:1554–1563, 1966.
10. Baum LE, Petrie T, Soules G, Weiss N, A maximization technique occuring in the statistical analysis of probabilistic functions of Markov Chains, *Ann Math Stat* **41**:164–171, 1970.
11. Bentley JL, Multidimensional binary search trees used for associative searching, *Comm Assoc Comput Machin* **18**:509–551, 1975.
12. Bilmes J, A Gentle tutorial on the EM Algorithm and its application to parameter estimation for Gaussian Mixture and Hidden Markov Models, *Tech. Report* ICSI-TR-97-021, University of Berkeley, 1997.
13. Birney E, Hidden Markov Models in biological sequence analysis, *IBM J Res Dev* **45**:755–763, 2001.
14. Blekas K, Dimitrios IF, Likas A, Greedy mixture learning for multiple motif discovery in biological sequences, *Bioinformatics* **19**:607–617, 2003.
15. Boyles R, On the convergence of the EM algorithm, *J Roy Stat Soc Ser* **B45**: 47–50, (1983).
16. Brazma A, Jonasses L, Eidhammer I, Gilbert D, Approaches to the automatic discovery of patterns in biosequences, *J Comput Biol* **5**:277–303, 1998.
17. Burge C, Karlin S, Prediction of complete gene structure in human genomic DNA, *J Mol Biol* **268**:78–94, 1997.
18. Cardon LR, Stormo GD, Expectation maximization algorithm for identifying protein binding sites with variable length from unaligned DNA fragments, *J Mol Biol* **223**:159–170, 1992.
19. Chappelier JC, Rajman M, A generalized CYK algorithm for parsing stochastic CFG, in *Proceedings of 1st Workshop on Tabulation in Parsing and Deduction (TAPD98)* (Paris), 1998.
20. Chomsky N, Three models for the description of language, *IRE T Inform Theor* **2**:113–124, 1956.
21. ——, On certain formal properties of grammars, *Inform Cont* **2**:137–167, 1959.
22. Chow YS, Tiecher H, *Probability Theory*, Springer Verlag, Berlin, 1998.
23. Chung KL, *Markov Chains with Stationary Transition Probabilities*, Springer Verlag, Berlin, 1967.
24. Churchill GA, Stochastic models for heterogeneous DNA sequences, *Bulletin of Mathematical Biology* **51**:79–91, 1989.
25. Claus V, Ehrig H, Rozenberg G (eds.), Graph grammars and their application to Computer Science and Biology, Lecture Notes in Computer Science, Vol. 73, Springer Verlag, New York, 1979.
26. Dempster AP, Laird NM, Rubin DB, Maximum likelihood from incomplete data via the EM algorithm, *J Roy Stat Soc* **B39**:1–38, 1977.
27. Dong S, Searls DB, Gene structure prediction by linguistic methods, *Genomics* **23**:540–551, 1994.
28. Dubrin R, Eddy RS, Krogh A, Mitchison G, *Biological Sequence Analysis*, Cambridge University Press, London, 1998.
29. Eddy SR, *Hidden Markov Model and large scale genome analysis*, Transaction of American Crystallographic Association (1997).
30. ——, Profile Hidden Markov Models, *Bioinformatics* **14**:755–763, 1998.
31. Eddy SR, Durbin R, RNA sequence analysis using covariance models, *Nucleic Acids Research* **22**:2079–2088, 1994.
32. Feller W, *An introduction to probability theory and its applications*, Vol. I,II, John Wiley, New York, 1968,1971.

33. Flappan E, *When Topology Meets Chemistry*, Cambridge University Press, London, 2000.
34. Forney GD Jr., The Viterbi algorithm, *Proceedings of IEEE* **61**:268–278, 1973.
35. Goldman N, Thorne JL, Jones DT, Using evolutionary trees in protein secondary structure prediction and other comparative sequence analyses, *J Mole Biol* **263**: 196–208, (1996).
36. Gollery M, Specialized Hidden Markov model databases for microbial genomics, *Comparative and Functional Genomics* **4**:250–254, 2003.
37. Grate L, Hughey R, Karplus K, Sjolander K, Stochastic modeling techniques, understanding and using Hidden Markov Models, *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology (St. Louis)*, AAAI Press, 1996.
38. Gribskov M, McLachlan AD, Eisenberg D, Profile analysis and detection of distantly related proteins, *Proceedings of the National Academy of Sciences*, USA **84**:4355–4358, 1987.
39. Gusfield D, *Algorithms on Strings,Trees and Sequences*, Cambridge University Press, London, 1999.
40. Harrison MA, *Introduction to Formal Language Theory*, Addison-Wesley, New York, 1978.
41. Hopcroft JE, Motwani R, Ullman JD, *Introduction to Automata Theory, Languages and Computation*, Pearson Education, Singapore, 2001.
42. Jagota A, Lyngsø RB, Pedersen CNS, Comparing an HMM and SCFG, *Proceedings of the 1st Workshop on Algorithms in Bioinformatics*, WABI 01 (Denmark), 2001, pp. 69–84.
43. Kasami T, An Efficient Recognition and Syntax Algorithm for Context-Free Languages, *Tech. report, Air Force Cambridge Research Lab*, Cambridge, 1965.
44. Kenney JG, Snell JL, *Finite Markov Chains*, Academic Press, New York, 1966.
45. Knudsen B, Hein J, RNA secondary structure prediction using stochastic context free grammars and evolutionary history, *Bioinformatics* **15**:446–454, 1999.
46. ——, Pfold: RNA secondary structure prediction using stochastic context-free grammar, *Nucleic Acid Research* **13**:3423–3428, 2003.
47. Koski T, *Hidden Markov Models for Bioinformatics*, Kluwer, Netherlands, 2001.
48. Krogh A, *Computational Methods in Molecular Biology*, Ch. 4, pp. 45–63, Elsevier, 1999, pp. 45–63.
49. Krogh A, Brown M, Mian IS, Sjolander K, Haussler D, Hidden Markov Model in computational biology: applications to protein modeling, *J Mole Biol* **235**:1501–1531, 1994.
50. Krogh A, Mian IS, David H, A Hidden Markov Model that finds genes in *E.coli DNA*, *Nucleic Acids Research* **22**:4768–4778, 1994.
51. Kulp DC, *Protein Coding Gene Structure Prediction using Generalized Hidden Markov Model*, Ph.D. thesis, Department of Computer Science and Engineering, University of California, Santa Cruz, 2003.
52. Lander ES, Green P, Construction of multilocus genetic linkage maps in humans, *Proceedings of the National Academy of Sciences, USA* **84**:2363–2367, 1987.
53. Lange K, *Mathematical and Statistical Methods for Genetic Analysis*, Springer Verlag, Berlin, 1997.
54. Lari K, Young SJ, The estimation of stochastic context free grammars using the Inside-Outside algorithm, *Computer Speech and Language* **4**:35–36, 1990.
55. Lawrence CE, Reilly AA, An Expectation Maximization algorithm for the identification and characterization of common sites in unaligned biopolymer sequences, *Protein* **7**:41–51, 1990.

56. Lewin B, *Genes VII*, Oxford University Press, London, 2000.
57. Li JQ, Barron AR, *Mixture density estimation*, Advances in Neural Information Processing Systems, Vol. 12, MIT Press, 2000, pp. 279–285.
58. Lukashin AV, Bodovsky M, Genmark.HMM: New solutions for gene finding, *Nucleic Acids Research* **26**:1107–1115, 1998.
59. Lyngsø RB, Pedersen CNS, *Pseudoknots in RNA secondary structures*, Proceedings of the Fourth Annual International Conference on Computational Molecular Biology (Tokyo, Japan), 2000, pp. 201–209.
60. Mamitsuka H, Abe N, Predicting location and structure of beta-sheet regions using stochastic tree grammars, *Proceedings of 2nd International Conference on Intelligent Systems for Molecular Biology*, Vol. 263, 1994, pp. 276–284.
61. McLachlan GJ, Krishnan T, *The EM Algorithm and Extensions*, John Wiley & Sons, New York, 1997.
62. Norris JR, *Markov Chains*, Cambridge University Press, London, 1997.
63. Nussinov R, Pieczenik G, Griggs JR, Kleitman DJ, Algorithms for loop matchings, *SIAM Journal of Applied Mathematics* **35**:68–82, 1978.
64. Peshkin L, Gelfand MS, Segmentation of yeast DNA using Hidden Markov Model, *Bioinformatics* **15**:980–986, 1995.
65. Rabiner LR, A tutorial on Hidden Markov Models and selected applications in speech recognition, *Proceedings of the IEEE* **77**:257–286, 1989.
66. Rigoutsos L, Floratos A, Parida L, Gao Y, Platt D, The emergence of pattern discovery techniques in computational biology, *Metabolic Engineering* **2**:159–177, 2000.
67. Rivas E, Eddy SR, A dynamic programming algorithm for RNA structure prediction including pseudoknots, *J Mole Biol* **285**:2053–2068, 1999.
68. ——, The language of RNA: A formal grammar that includes pseudoknots, *Comparative and Functional Genomics* **16**:334–340, 2000.
69. Robert CB, *Computational Linguistics*, MIT Press, Boston, 1989.
70. Robin DD, Sean RE, Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction, *BMC Bioinformatics* **5**: 2004.
71. Sakakibara Y, Brown M, Hughey R, Mian SI, Sjlander K, Underwood RC, Haussler D, *Recent methods for RNA modeling using stochastic context-free grammars*, Proceedings of the Asilomar Conference on Combinatorial Pattern Matching (NY), Springer-Verlag, 1994.
72. Sakakibara Y, Brown M, Underwood RC, Mian IS, Haussler D, Stochastic context free grammar for tRNA modelling, *Nucleic Acid Research* **22**:5112–5120, 1994.
73. Sanchez JA, Benedi JM, Casacuberta F, Advances in Structural and Syntactical Pattern Recognition, pp. 50–59, Advances in Structural and Syntactial Pattern Recognition, Springer Verlag, Heidelberg, 1996, pp. 50–59.
74. Searls DB, The linguistics of DNA, *American Scientist* **80**:579–591, 1992.
75. ——, String variable grammar: A logic grammar formalism for the biological language of a DNA, *Journal of Logic Programming* **24**:73–102, 1995.
76. Sonnhammer EL, Eddy SR, Birney E, Bateman A, Durbin R, Pfam: Multiple sequence alignments and HMM-profiles of protein domains, *Nucleic Acid Research* **26**:320–322, 1998.
77. Sonnhammer EL, Eddy SR, Durbin R, Pfam: A comprehensive database of protein families based on seed alignments, *Proteins* **28**:405–420, 1997.
78. Tabasaka JE, Cary RB, Gabow HN, Stormo GD, An RNA folding method capable of identifying pseudoknots and base triples, *Bioinformatics* **8**:691–699, 1998.

79. Tanaka H, Ishikawa M, Asai KA, Konagaya A, *Hidden Markov Model and iterative aligners*, Proceedings of 1st International Conference on Intelligent Systems for Molecular Biology (Menlo Park), AAAI Press, 1993, pp. 395–401.
80. Underwood RC, *The application of stochastic context-free grammars to folding, aligning and modeling homologous RNA sequences*, Tech. report, UCSC-CRL-94-14, University of California, Santa Cruz, 1993.
81. Verbeek JJ, Vlassis N, Krose B, *Efficient greedy learning of Gaussian mixture*, (Amsterdam), In Proceedings of the 13th Belgium-Dutch Conference on Artificial Intelligence BNAIC'01, Vol. 12, Amsterdam, 2001, pp. 251–258.
82. Vlassis N, Likas A, A greedy EM algorithm for Gaussian mixture learning, *Neural Processing Letter* **15**:77–87, 2002.
83. Wang TLJ, Shapiro AB, Shasha D (eds.), *Pattern Discovery in Biomolecular Data*, Oxford University Press, London, 1999.
84. Waterman MS, *Introduction to Computational Biology*, Chapman & Hall, London, 1995.
85. Wetherell CS, Probabilistic languages a review and some open questions, *Comp. Surveys* **12**:361–379, 1980 .
86. Wu J, On the convergence properties of the EM algorithm, *The Annals of Statistics*, pp. 95–103, 1983.
87. Zuker M, Computer prediction of RNA structure, *Methods in Enzymology* **180**:262–288, 1989.

Sushmita Mitra is a Professor at the Machine Intelligence Unit, Indian Statistical Institute, Kolkata. From 1992 to 1994 she was in the RWTH, Aachen, Germany as a DAAD Fellow. She was a Visiting Professor in the Computer Science Departments of the University of Alberta, Edmonton, Canada in 2004, Meiji University, Japan in 1999 and 2004, and Aalborg University Esbjerg, Denmark in 2002 and 2003. Dr. Mitra received the *National Talent Search Scholarship* (1978–1983) from NCERT, India, the *IEEE TNN Outstanding Paper Award* in 1994 for her pioneering work in neuro-fuzzy computing, and the *CIMPA-INRIA-UNESCO Fellowship* in 1996.

She is the author of two books published by John Wiley, guest edited two special issues of journals, and has more than 70 research publications in referred international journals. According to the Science Citation Index (SCI), two of her papers have been ranked 3rd and 15th in the list of *Top-cited papers in Engineering Science* from India during 1992–2001. Her current research interests include data mining, pattern recognition, soft computing, image processing, and Bioinformatics.

Shibaji Mukherjee received the B.Sc (Hons) and M.Sc degree in Physics from University of Calcutta and M.S degree in Physics from Northeastern University, Boston in 1989, 1993 and 1995. He is an executive body member of Association for Studies in Computational Biology (ASICB), Calcutta. He is responsible for planning and coordinating programs in Computational Biology at ASICB. He is presently working as a Technical Manager at STC Systems, India. He heads the Composite Applications group in STC. He is responsible for managing and coordinating integration solutions development for protocols and frameworks. His areas of research interest include Machine Learning, Computational Biology, Algorithms and Distributed Computing. He can be reached at mshibaji@acm.org