Overlapping Genes Detected by Hidden Markov Models in PRISM

Master Thesis

Supervisor: Henning Christiansen

Students: Yuan Zhang，HongBo Liu

Department of Communication, Business and Information Technologies

Roskilde University

Abstract

As more new species have been discovered, the researchers can get biological sequences of species from DNA sequencing institute by sending species samples to the institute. Nowadays, finding properties and structures in biological sequences data is essential for research and analysis in several areas such as gene finding in genetics area. A number of gene finder tools have been developed, but they give quite different results for detecting genes. We use a computational method for modeling and analysis that combines statistic and machine learning with logic programming technology. We propose to use logic-statistical modeling system-PRISM for building HMM models and making predictions. PRISM is embedded with statistical and machine learnng techniques such as Viterbi and EM learning. So we can run Viterbi computations in PRISM to make gene predictions for a gven sequence.

We make several experiments, in which we develop different PRISM models for detecting non overlap genes or overlapping genes. We use an approach called coordinating HMMs (CHMMs) that is to run separate models in parallel, which is capable of describing arbitrary overlapping structures. We make three coordinated PRISM models and run them separately and independently, but at the same time produce identical sequence. In this way, based on separate analyses at the same time to the same sequence by these coordinated models, all possible overlapping genes in different reading frames could be detected with considering their ‘signals’ such as promoter, Shine Dalgarno sequence. Our test experiments based on an artificial sequence containing three overlapping genes confirm that the inferences of existence of overlapping genes can be performed by this approach. Our experiments in the PRISM system indicate that logic-statistical modeling method provides more expressive power and ease of modeling.

Table of Contents

1. Introduction

As many organisms have been completely sequenced in the past few years, a more accurate and fast tool is necessary for gene prediction via dealing with biological sequences data. Though several gene prediction tools have already been developed, the results they give are quite different (Christiansen and Dahmcke, 2007) and some very small genes and overlapping genes on DNA reverse strand are still missed by them (Siebenlist and Gilbert, 1980). Overlapping genes may be encoded opposite each other by the same piece of DNA in prokaryotic cells (Silby and Levy, 2008). Nevertheless, it is difficult to detect overlapping genes in prokaryotes. Different orientations of overlapping genes are not considered by GeneMark (Lukashin and Borodovsky, 1998) that is commonly used gene finder tool because of lacking of sufficient data for obtaining statistical models of overlapping genes. Biologists usually use gene finding tools to find genes first. And then, they go to lab to do some experiments to prove it and find gene functions. If a tool can detect overlapping genes, it will contribute to the biologists detect some more important genes that are ignored by traditional tools. Hidden Markov models (HMMs) have been applied for traditional tools to predict genes based on DNA sequences. PRogramming In Statistical Modeling (PRISM) is a powerful tool for building complex statistical models, which is developed based on logic programming (Sato and Kameya, 2001). PRISM is embedded with statistical and machine learnng techniques such as Viterbi and EM learning. So we can run Viterbi computations in PRISM to make gene predictions for a gven sequence. Our approach is to make a main HMM model and implemented it in PRISM. This main model contains three sub-models which detect the genes on the different reading frames[1] respectively. The overlapping case can be described by the model via coordinating of the three sub-models. Consequently, the existence of overlapping genes inferences can be performed by an HMM in PRISM.

2. Specification of the problem

As more new species have been discovered, the researchers can get biological sequences of species from DNA sequencing institute by sending species samples to the institute. Nowadays, it becomes important effectively use these biological sequences data to predict unknown genes in order to guide further biological experiments. There are still some overlapping genes that can not be found by the

[1] Reading frame is a way to read DNA sequence depending on which base is the first base for starting reading in the sequence.

traditional tools, such as Ecoparse (Krogh, *et al.*, 1994), GeneMark (Lukashin and Borodovsky, 1998), EasyGene (Larsen and Krogh, 2003), Glimmer (Salzberg, *et al.*, 1998), and so on. As 1 or 4 bases overlaps in the same strand are common in prokaryotes, Ecoparse can predict 1 or 4 bases overlapping genes in the same strand (Krogh, *et al.*, 1994). GeneMark can also predict some overlapping genes in the same strand. However, it is very difficult for them to predict overlapping genes in the reverse strands. The gene overlaps cause several difficulties for a high accuracy prediction. That is, it might be hard to exactly predict the start condon and ribosome binding site of a gene fall into the overlap region where oligonucleotide statistic may not fit o regularly used models (Lukashin and Borodovsky, 1998). Some bacterial genomes of prokaryotes contain overlapping genes which encoded in the same strand or in the different strand with other gene. In the bacterium *Pseudomonas fluorescens,* ten overlapping genes were discovered (Silby and Levy, 2008) and one of them, *iiv14* was encoded opposite to the gene *pfl_0939* by the same piece of DNA (Fig. 1). They did some lab experiments and found that *iiv14* encodes a protein that functions to promote colonization of soil. Their findings indicate that bacterial genomes have more genes than that currently annotated, and some overlapping genes have escaped detection because they occupy the same piece of DNA with other genes.

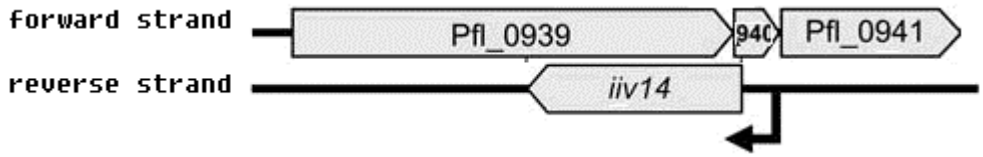


Fig. 1 overlapping gene *iiv14* was found in the DNA reverse strand (adapted from Silby and Levy, 2008)

These ignored overlapping genes may be functionally important for regulating bacterial cell cycle. An unconventional approach is required for detecting these overlapping genes, which will greatly contribute to deduce and identify some new genes that have not been annotated yet.

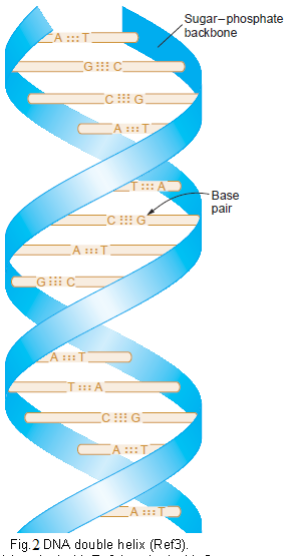


Fig. 2 DNA double helix (Ref3). A is paired with T, C is paired with G

3. Genetic foundations

3.1 The basic concept

There are huge amounts of different kinds of organisms on earth. Biologists study organism similarities and differences based on the analysis of their DNA sequences. DNA is genetic material that maintains organism's characteristics

during evolution. It is composed of four nucleotides which are represented by A, T, C and G (Griffiths, *et al*., 2004). Two strands are twisted together forming DNA double helix. Base A is always complementary to base T and Base C is always complementary to base G (Fig.2).

DNA molecule contains many genes and each gene is a specific sequence that carries the genetic information required for making into a functional protein (Fig.3). Proteins are made up of a series of amino acid molecule (Griffiths, *et al*., 2004). As the figure 4 shows, a codon is a triplet of bases that codes for a specific amino acid (Nelson and Cox, 2005). The beginning of a gene contains a start codon, which is 'ATG' in most of the time. But sometimes 'GTG' and 'TTG' work as start codon, for example, bacterium *E. coli* uses 83% ATG, 14% GTG and 3% TTG as start codon (Blattner, *et al*. 1997). There are three different stop codons which are 'TAG', 'TAA' and 'TGA' at the end of a gene (Nelson and Cox, *et al*., 2004).

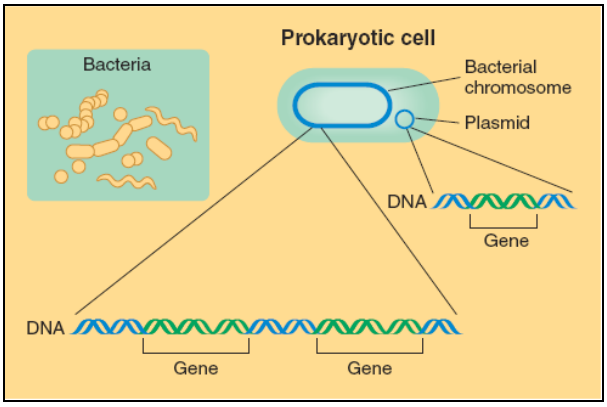


Fig. 3 DNA molecule contains many genes (Griffiths et al., 2004)

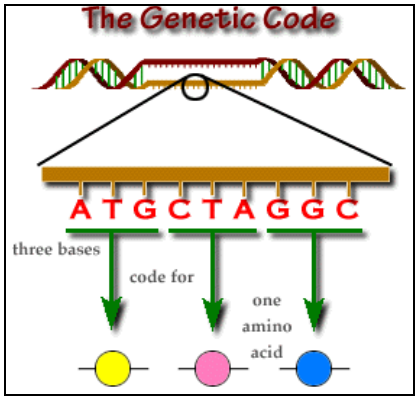


Fig. 4 Three bases code for one amino acid (Tan S., 2008)

In principle, a piece of DNA contains six reading frames; three in each strand (Fig. 5). Reading frame is a way to read DNA sequence depending on which base is the first base for starting reading in the sequence (Griffiths, *et al.*, 2004).

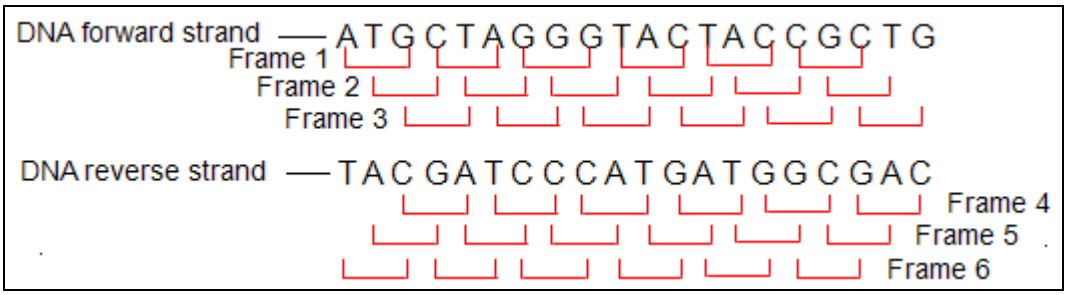


Fig. 5 DNA double strands with six reading frames. In the DNA forward strand, frame 1 starts with the first base 'A', frame 2 starts with the second base 'T' and frame 3 starts with the third base 'G'. The DNA reverse strand is read on the opposite orientation, frame 4 starts with the first base 'C', frame 5 starts with the second base 'A' and frame 6 starts with the third base 'G'.

3.2 The common features of prokaryotic genes

Start codon and stop codon are in the same reading frame, which is a characteristic of the coding region of a gene. However, prediction of a prokaryotic gene is not only based on finding start codon and stop codon. Subsequences in front of start codon or after the stop codon may contain some particular kinds of signals, such as promoter, etc, which is another characteristic for detecting a real gene (Griffiths, *et al*., 2004). A signal represents a portion of DNA that contains several nucleotides, which are complementary with nucleotides of another molecule. In this way, the signal can be bound by some molecules, which plays an important role during gene expression.

Gene expression includes DNA transcription and DNA translation. DNA transcription is a process of making messenger RNA (mRNA) from a DNA template. Translation is synthesis of a protein from an mRNA template. mRNA is a molecule that is required as a messenger for DNA to be translated into a protein. The site in front of start codon may contain two specific sequences that can be recognized by a molecule called sigma factor. This site is termed promoter and two specific sequences are termed Gilbert Box (GB) sequence and Pribnow Box (PB) sequence respectively (Siebenlist and Gilbert, 1980). While sigma and RNA polymerase[2] bind to this promoter site, the DNA double strand opens and DNA transcription starts. This transcription start site is designated as +1. Another signal is called Shine Dalgarno sequence (SD) which is located near start codon. The space between SD and start codon is about 5-10 bases (Shine and Dalgarno, 1975). While Shine Dalgarno sequence is bound by a ribosome[3] molecule and ribosome reads start codon, DNA translation starts. The third signal is terminator, which is located in the position after stop codon. DNA transcription stops while RNA polymerase reaches the terminator signal. Some terminators require Rho protein to disrupt the further transcription, whereas others function independently of Rho protein, which form a structure to cause transcription stops. The former is termed rho-dependent terminator and the latter is termed rho-independent terminator. The rho-independent (Fig 6) is also called hairpin loop, which is the region that contains self complimentary sequences and forms a hairpin structure. The Rho dependent region has a sequence of about 40 to 60 bases that is rich in C bases and poor in G bases (Nelson and Cox, 2005).

[2] RNA polymerase is an enzyme that produces RNA.
[3] Ribosome is a molecule that functions in DNA translation process.

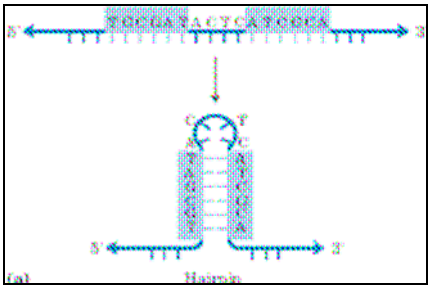


Fig 6 Palindromic single DNA strand forms a hairpin structure (Nelson and Cox, 2005). There are two highlight subsequences on the top single strand. The first base 'T' of left highlight is complementary to the last base 'A' of right highlight, the second base 'G' of left highlight is complementary to the second very right base 'C', and so on.

Some common features of prokaryotic genes (Fig 7) can be captured in terms of the following signals.

1). Promoter site generally contains two boxes, Gilbert Box (GB) - TTGACA and Pribnow Box (PB) - TATAAT. They are located in position -35 and -10 respectively. The space between them is 17 bases (Hawley and McClure, 1983; Harley and Reynolds, 1987 and reviewed in DeHaseth, *et al*., 1998).

2). Shine Dalgarno sequence (SD) -AGGAGG is found 5-10 bases in front of start codon (Shine and Dalgarno, 1975; Hannenhalli, *et al*., 1999).

3).Terminator site generally includes two kinds of signals - hairpin loop and rho dependent signal. Hairpin loop has palindromic DNA sequences about 30-60 bases after the stop codon, followed by around 8 T bases (Wilson and von Hippel, 1995; De Hoon, *et al*., 2005).

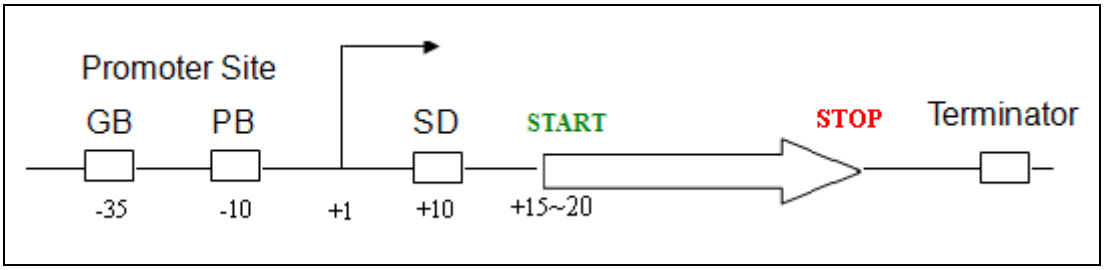


Fig.7 some common features of prokaryotic genes. DNA transcription start site is designated as +1. The relative positions to start site of some signals and the coding region of a gene are showed in a DNA strand. The black arrow denotes the start position of the gene expression. Empty box with arrows denotes the coding region including start codon and stop codon of a gene. Empty boxes denote the signals - two boxes GB and PB in promoter site, shine dalgarno sequence - SD and terminator sequence respectively. The numbers under the DNA strand are the relative positions to the gene expression.

3.3 The different overlapping patterns

3.3.1 Two genes overlapping patterns

Two genes overlapping patterns are showed up based on study of overlapping genes in the genomes of *Mycoplasma genitalium* and *Mycoplasma pneumoniae* (Fukuda, *et al*., 1999). Two adjacent genes are regarded as overlapping genes due to their coding regions in the different reading frames. The coding region includes start codon, stop codon and several other codons between them. The two genes are overlapping each other in two ways - same-strand overlap (Fig. 8a, 8b) and different-strand overlap (Fig.8c, 8d, 8e, and 8f).

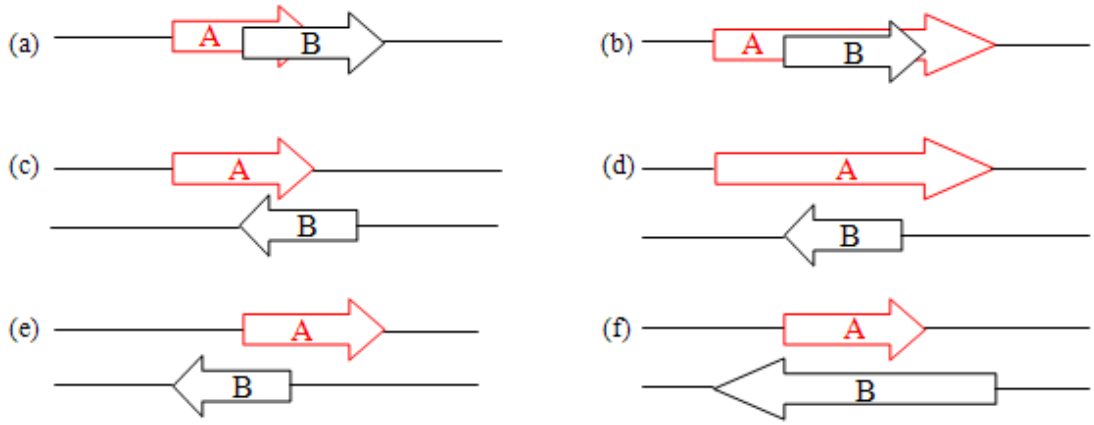


Fig.8 Six types of overlapping patterns for gene A and B. Empty boxes with arrows indicate the coding region of the genes, genes A and B are in red and black, respectively. (a) and (b) indicate same-strand overlapping pattern. (c), (d), (e) and (f) indicate different-strand overlapping pattern. (a) indicates gene A overlaps with part of sequences of gene B. (b) indicates gene A overlaps with the whole sequences of gene B, that is, gene B is included in gene A. (c) indicates the end part of sequences of gene A and B overlap each other. (d) indicates gene A overlaps with the whole sequences of gene B, that is, gene B is included in gene A. (e) indicates the head part of sequences of gene A and B overlap each other. (f) indicates gene B overlaps with the whole sequences of gene A, that is, gene A is included in gene B.

3.3.2 Six genes overlapping pattern

In above section, we illustrated two genes overlapping patterns that obtained from the article (Fukuda, *et al*., 1999). Many published articles about overlapping genes only considered two genes overlap each other in a segment of DNA. In addition, two genes' respective 'signals' such as PB, GB and SD overlap each other that were not considered. Actually, in a segment of DNA, it could be more than two genes, but at most six genes in the different reading frames that overlap each other. Furthermore, these genes' signals could also be overlap each other. We illustrate all possible genes

overlap each other in a segment of DNA (Fig 9).

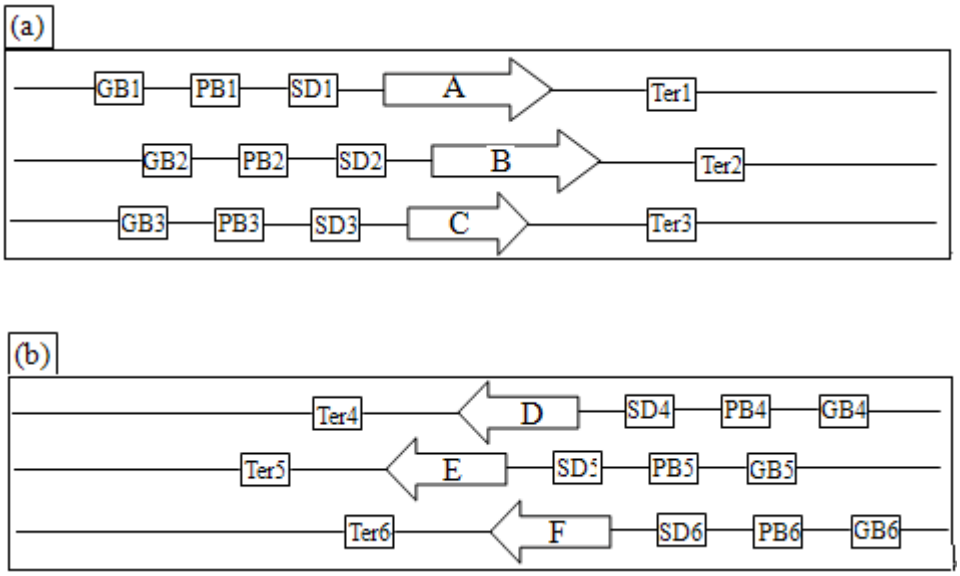


Fig 9 Six genes overlapping pattern. A segment of DNA contains six genes, gene A, B, C, D, E and F in the different reading frames that overlap each other. GB represents Gilbert Box, PB represents Pribnow Box, SD represents Shine Dalgarno sequence and Ter represents terminator sequence. (a) Illustrates three genes of DNA direct strand overlap each other, gene A, B and C are in the reading frame 1, 2 and 3 respectively. (b) Illustrates three genes of DNA reverse strand overlap each other, gene D, E and F are in the reading frame 4, 5 and 6 respectively.

These six genes – geneA, B, C, D, E and F are in the different reading frames and they overlap each other. Gene A is in the reading frame 1; Gene B is in the reading frame 2 and so on. These genes have their own signals - Gilbert Box (GB), Pribnow Box (PB), Shine Dalgarno (SD) and terminator sequence (Ter).

4. Computational models for sequence analysis

4.1 Hidden Markov models

4.1.1 Applications of HMM

A Hidden Markov Model (HMM) is a state-based statistical model, which transitions stochastically from state to state and emits a single symbol from each state based on emission probabilities of that state. HMM has been successfully applied to many problems in computational biology and in other fields, such as speech recognition field (Rabiner, 1989). Finding genes in DNA sequences is one of the problems in computational

biology. During the past several years HMM has become the most successful model used in gene finding. It was first applied to gene finding in prokaryotes by Krogh *et al,* since 1994 (Krogh, *et al*., 1994). They used HMM to find genes in the bacterium *E. coli* DNA sequences. Later on several gene finding programs was developed in order to predict genes more accurate and reliable. The GeneMark.hmm program is one of these gene finding programs, which was developed by Lukashin and Borodovsky in 1998 (Lukashin and Borodovsky, 1998). It applied generalized HMM to find genes in bacterial genomes. This HMM-based program provides more accurate and reliable gene prediction via several years update.

4.1.2 Definition of HMMs

An HMM consists of a set of hidden states, a set of observable letters, state transition probabilities and emission probabilities that are probability of emitting each letter in each state. Therefore it contains two stochastic processes, one is the process of transition between states, and another one is the process of emitting an output letter. The former is a hidden process and the latter is an observed process (Rabiner and Juang, 1986).

The elements of an HMM are formally defined by Lawrence Rabiner (Rabiner, 1989). An HMM is characterized by the following elements:

1. Set S of N states, $S = \{S_1, S_2, \ldots, S_N\}$.
2. Set V of M observation symbols, the output symbol. $V = \{V_1, V_2, \ldots, V_M\}$.
3. Set A of state transition probabilities, $A = a_{ij}$ where a_{ij} is the probability of a transition from the state i to the state j.

$$a_{ij} = P(q_{t+1} = S_j \mid q_t = S_i),\ 1 \le i, j \le N.$$

4. Set B of symbol emission probabilities in the state j. $B = b_j(k)$, where $b_j(k)$ is the probability of emitting symbol k at the state j.

$$b_j(k) = P(V_k \mid q_t = S_j),\ 1 \le j \le N,\ 1 \le k \le M.$$

5. Set π, the initial state distribution $\pi = \{\pi_i\}$ where π_i is the probability that state i is a start state.

$$\pi_i = P(q_1 = S_i),\ 1 \le i \le N.$$

The above is the definition of an HMM. In order to facilitate indication of the complete parameter set of the model, a compact notation is usually used as follows:

$$\lambda = (\pi, A, B)$$

We draw a simple HMM model (Fig. 10) so as to be able to better understand the above definition.

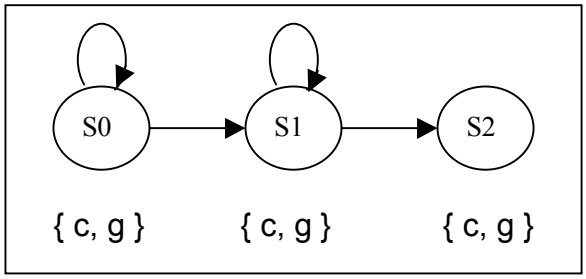


Fig.10 A simple HMM has three states S0, S1 and S2.
Each state emits a letter 'c' or 'g'.

This HMM has three states S0, S1 and S2 where S0 is initial state and S2 is final state. In each state, the HMM emits a letter either 'c' or 'g'. The model can move from the state S0 to S1 or move from the state S0 to S0. And the model can move from the state S1 to S2 or move from the state S1 to S1. So we can use the following notation to define this HMM.

Given a model $\lambda = (\pi, A, B)$, where N = 3, M = 2, $a_{11}, a_{12}, a_{22}, a_{23}$, any of them can not be zero, $a_{11} + a_{12} = 1$, $a_{22} + a_{23} = 1$, $b_1(c) + b_1(g) = 1$, $b_2(c) + b_2(g) = 1$, $b_3(c) + b_3(g) = 1$, $\pi = \{1, 0, 0\}$.

Let us make a detailed interpretation about the above notation of the HMM.

$\lambda = (\pi, A, B)$: The model λ has three parameters π, A and B.

N = 3: The model contains three states S0, S1 and S2.

M = 2: The model emits two letters 'c' or 'g'.

$a_{11}\ a_{12}, a_{22}\ a_{23}$ can not be zero:
The probabilities of moving from the state S0 to S0, the probabilities of moving from the state S0 to S1, the probabilities of moving from the state S1 to S1 and the probabilities of moving from the state S1 to S2 are not all zero.

$a_{11} + a_{12} = 1$: The sum of the probabilities of moving from the state S0 to S0 and the probabilities of moving from the state S0 to S1 should be one.

$a_{22} + a_{23} = 1$: The sum of the probabilities of moving from the state S1to S1 and the probabilities of moving from the state S1 to S2 should be one.

$b_1(c)+b_1(g)=1$:The sum of the probability of emitting letter 'c' and emitting letter 'g' at the state S0 should be one.

$b_2(c)+b_2(g)=1$:The sum of the probability of emitting letter 'c' and emitting letter 'g' at the state S1 should be one.

$b_3(c)+b_3(g)=1$:The sum of the probability of emitting letter 'c' and emitting letter 'g' at the state S2 should be one.

$\pi = \{1, 0, 0\}$: The probability of the state S0 as start sate is one, therefore the state S0 is the initial state and S1 and S2 are not.

4.1.3 Three basic problems for HMMs

Solving three basic problems of HMMs is crucial for real-world applications. The three problems and the corresponding solutions are summarized by Lawrence Rabiner (Rabiner, 1989). These problems are as follows:

Problem 1: Given the observation sequence $O = O_1 O_2 \ldots O_T$, where each observation O_T is one of the letters from V and T is the number of observations in the sequence , and a model $\lambda = (\pi, A, B)$, how to compute $P(O \mid \lambda)$, the probability of the observed sequence, given the model.

Problem 2: Given the observation sequence $O = O_1 O_2 \ldots O_T$, and the model λ, how to choose a state sequence $Q = q_1 q_2 \ldots q_T$ which is the best state path to explain the observations.

Problem 3: How to adjust the model parameters $\lambda = (\pi, A, B)$ to maximize the probability of the observation sequence given the model. This problem allows us to optimize the model parameters so as to best explain how a given observation sequence comes about.

The formal mathematical solutions to each of the three problems are Forward Algorithm, Viterbi Algorithm and Expectation-Modification (EM) Algorithm. We will introduce them in details that are inspirited from the citations (Rabiner, 1989; Boyle, 1984) in the following sections respectively.

4.1.4 Forward algorithm

The first problem is an evaluation problem, that is to evaluate how well a given model matches a given observation sequence. It is very useful for dealing with such a case-choose a model from several models, which best matches the given observation sequence. The most straight forward thought is to consider all possible state sequences of length T. Given a model, the probability of observation sequence is calculated by adding up a joint probability over all possible sate sequences. This joint probability is derived from multiplying two probabilities, that is, the probability of observation sequence for a given state sequence and a model multiplied by the probability of the state sequence for a given model. The corresponding formula is defined (Rabiner, 1989) as follows:

$$P(O \mid \lambda) = \sum_{allQ} P(O \mid Q, \lambda) P(Q \mid \lambda)$$

$$= \sum_{q_1, q_2, \ldots, q_T} \pi_{q1} b_{q1}(O_1) a_{q1q2} b_{q2}(O_2) \ldots a_{q_{T-1}q_T} b_{q_T}(Q_T)$$

where $P(O \mid \mathrm{Q}, \lambda) = \prod_{t=1}^{T} P(O_t \mid \mathrm{q}_t, \lambda)$ (Probability of an observation sequence)

$$= b_{q_1}(O_1) \cdot b_{q_2}(O_2) \cdots b_{q_T}(O_T)$$

$P(Q \mid \lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \cdots a_{q_{T-1} q_T}$ (Probability of a state sequence)

$O = O_1\ O_2 \ldots O_T$ (An observation sequence)
$Q = q_1\ q_2\ \ldots q_T$ (A state sequence)

It is obvious that this exhaustive evaluation method is extremely complex. It requires on the order of $2T*N^T$ calculations where T is number of observations, N is number of states, there are N^T possible state sequences since N possible states can be reached at every t = 1, 2,…, T. Let us use an example of HMM to illustrate it. Like the figure 10 of the model, this HMM also contains three states (N = 3) s0, s1, s2 and each state outputs symbol 'c' or 'g'. If the observation sequence O is 'c, g, g, g, c, c, c, g, g, c' with length 10 (T = 10), the possible state sequence Q has 3^{10} types of state combinations, like Q = s0 s1 s2 s1 s0 s2 s2 s1 s0 s1 is one of all types. The number of calculations required could be on the order of $20*3^{10}$. Imagine that the length of the observation sequence is 100, it requires on the order of $20*3^{100}$ calculations. The computation is quite complex by this method. Therefore, an efficient algorithm is required for solving this problem, which is Forward Algorithm.

The forward algorithm defines forward variable α_t (i), its formula is showed as follows, which is the probability that all the symbols until time t have been generated and the state is S_i at time t.

$$\alpha_t\ (i) = P\ (O_1\ O_2 \ldots O_t,\ q_t = S_i \mid \lambda)$$

The forward variable α_t (i) can be induced using the following recursive procedure:
1). Initialization:

$$\alpha_1\ (i) = \pi_i b_i\ (O_1),\ 1 \le i \le N$$

Initially we are in the state S_i and generate O_1.

2). Induction:

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t\ (i)\ a_{ij}]\ b_j\ (O_{t+1}),\ 1 \le t \le T-1,\ 1 \le j \le N$$

We can arrive at the state Sj from any of the previous state S_i with probability α_{ij} and generate O_{t+1}. Notice that calculating the probability α at time t+1 only uses the partial probability α at time t. However, in order to get the probability of time t, we have to calculate the probability at time t-1. So it is a recursive procedure that we need to

calculate the partial probabilities from time t=1 until t =T.

3). Termination:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Finally, given an HMM, the probability of the observation sequence is obtained by summing of all partial probabilities at time T.

In order to obtain the probability of the observation sequence, the forward algorithm calculates the partial probabilities (α) at time t = 1, 2,..., T, and sums all partial probabilities (α) at t = T. This is a recursive calculation that can reduce the complexity of calculation. So the computational complex of the forward algorithm is extremely lower than the exhaustive evaluation since its calculation is linear in T (Boyle, 1984). So it only requires on the order of N^2T calculation. Using the same example above, N = 3 and T = 100, only 900 calculations are required, which is quite less than $20*3^{100}$ calculations in exhaustive evaluation method.

The following is an example that we make here to illustrate about the recursive procedure of the forward algorithm.

Given an HMM $\lambda = (\pi, A, B)$. It has three states S_1, S_2, S_3 and each state outputs symbol 'c' or 'g'. The probabilities of state transition $a_{11} = 0.5$, $a_{12} = 0.25$, $a_{13} = 0.25$, $a_{21} = 0.25$ $a_{22} = 0.5$, $a_{23} = 0.25$, $a_{31} = 0.25$, $a_{32} = 0.25$, $a_{33} = 0.5$. The probabilities of emission symbol $b_1(c)$, $b_1(g)$, $b_2(c)$, $b_2(g)$, $b_3(c)$, $b_3(g)$ all equals 0.5. The initial state distribution $\pi_1=0.5$, $\pi_2=0.25$, $\pi_3=0.25$. If the observation sequence O is 'c, g, g, g, c, c, c, g, g, c' with length 10 (t = 10), what is the probability of this observed sequence, given the above model? So we can solve this problem by the recursive forward algorithm.

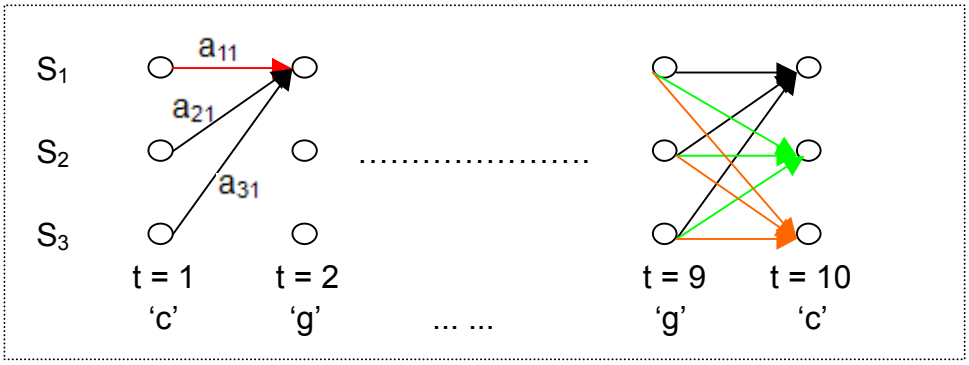


Fig.11 illustration of the recursive procedure for the forward algorithm
It includes 10 observations and three states S_1, S_2 and S_3.
t represents observations, S represents states.

First, we can calculate the initial probabilities at t = 1 by the first step's formula.
$\alpha_1(S_1) = \pi_1 * b_1(c) = 0.5 * 0.5 = 0.25$

$\alpha_1(S_2) = \pi_2 * b_2(c) = 0.25 * 0.5 = 0.125$
$\alpha_1(S_3) = \pi_3 * b_3(c) = 0.25 * 0.5 = 0.125$

Second, we uses the probabilities at t = 1 to calculate the probabilities at t = 2 by the second step's formula. In the figure 11, we can see that the state S_1 at t = 2 can be reached from three possible states at t=1 (three arrows are showed from t=1 to t=2). Thus the probabilities for the state S_1 at t = 2 can be calculated as follows:

$\alpha_2(S_1) = [\alpha_1(S_1) * a_{11} + \alpha_1(S_2) * a_{21} + \alpha_1(S_3) * a_{31}] * b_1(g)$
$= (0.25*0.5 + 0.125 * 0.25 + 0.125 * 0.25) * 0.5$
$= 0.094$

So we can use this way to calculate $\alpha_2(S_2)$ and $\alpha_2(S_3)$, the probabilities for the state S_2 and S_3 at t = 2. Consequently, we can use all partial probabilities at t =2 to calculate the probability at t = 3 and so on until t = 10. That is when we recursively get all partial probabilities at t = 9, we can use it to calculate $\alpha_{10}(S_1)$, $\alpha_{10}(S_2)$ and $\alpha_{10}(S_3)$. These probabilities for the different states at t=10 can be derived from the different path from the previous time t = 9 (these paths for each state are showed by black arrows, green arrows and orange arrows)

Finally, the probability of the observation sequence given the HMM is the sum of the partial probabilities at t = 10. That is $\alpha_{10}(S_1) + \alpha_{10}(S_2) + \alpha_{10}(S_3)$.

4.1.5 Viterbi algorithm

The second problem of HMM is to find the most likely state path way of an HMM based on the observation sequence. Viterbi Algorithm is used to solve this problem. It is similar except for the backtracking step to forward algorithm calculation. The major difference is the Viterbi algorithm calculates the probability of the most probable path to a state at time t, which is albe to choose the maximal probability, whereas the forward algorithm calculates the total probability of the all probable paths. Viterbi Algorithm is also a recursive calculation that can reduce the complexity of calculation. And it only requires on the order of N^2T calculation. The following is the recursive procedure of the Viterbi algorithm to find the best state sequence.

1). Initialization:

$$\delta_1(i) = \pi_i b_i(O_1),\ 1 \le i \le N$$
$$\phi_1(i) = 0.$$

2). Induction:

$$\delta_t(j) = \max_{1 \le i \le N}[\delta_{t-1}(i) a_{ij}] b_j(O_t), \qquad 2 \le t \le T,\ 1 \le j \le N$$

$$\phi_t(\mathrm{j}) = \arg\max_{1 \le \mathrm{i} \le \mathrm{N}} [\delta_{t-1}(i) a_{ij}], \qquad 2 \le t \le T,\ 1 \le j \le N.$$

3). Termination:

$$p^* = \max_{1 \le i \le N} [\delta_T(i)]$$

$$q_T^* = \arg\max_{1 \le \mathrm{i} \le \mathrm{N}} [\delta_T(i)].$$

4). Path (a state sequence) backtracking:

$$q_T^* = \phi_{t+1}\left(q_{t+1}^*\right), \qquad \mathrm{t} = \mathrm{T} - 1, \mathrm{T} - 2, \ldots, 1.$$

Let's use the previous example (See Fig. 11) to illustrate the recursive procedure of the Viterbi algorithm.

First, we can calculate the initial probabilities at t = 1 by the first step's formula.

$\delta_1(S_1) = \pi_1 * b_1(c) = 0.5 * 0.5 = 0.25$
$\delta_1(S_2) = \pi_2 * b_2(c) = 0.25 * 0.5 = 0.125$
$\delta_1(S_3) = \pi_3 * b_3(c) = 0.25 * 0.5 = 0.125$

Second, we uses the probabilities at t = 1 to calculate the probabilities at t = 2 by the second step's formula. In the figure 11, we can see that the state S_1 at t = 2 can be reached from three possible states at t=1 (three arrows are showed from t=1 to t=2). We choose the path with the maximal probability as a partial best path. In order to remember the partial best path, we use an array $\phi_t(j)$ to do this. ϕ is a back pointer which points to the previous state that optimally generates the current state (Boyle, 1984). For example, $\phi_2(S_1)$ points to the state S_1 at t = 1 since the best path is the red arrow in the figure 11. Thus the probabilities for the state S_1 at t = 2 can be calculated as follows:

$\delta_2(S_1) = \max [\delta_1(S_1) * a_{11}, \delta_1(S_2) * a_{21}, \delta_1(S_3) * a_{31}] * b_1(g)$
where $\delta_1(S_1) * a_{11} = 0.25 * 0.5 = 0.125$,
$\delta_1(S_2) * a_{21} = 0.125 * 0.25 = 0.03125$
$\delta_1(S_3) * a_{31} = 0.125 * 0.25 = 0.03125$
So we choose the maximum value 0.125.
$\delta_2(S_1) = 0.125 * 0.5 = 0.0625$
$\phi_2(S_1)$ records the partial best path, that is, from the state S_1 at t = 1 to the state S_1 at t = 2 (the red arrow is the best path in the figure 11)

So we can use this way to calculate $\delta_2(S_2)$ and $\delta_2(S_3)$, the partial probabilities for the state S_2 and S_3 at t = 2. $\phi_2(S_2)$ and $\phi_2(S_3)$ record the partial best paths. Consequently, we can use all partial probabilities at t =2 to calculate the probability at t = 3 and so on until t = 10. That is, we recursively get all partial probabilities $\delta_{10}(S_1)$, $\delta_{10}(S_2)$, $\delta_{10}(S_3)$

and the partial best paths $\phi_{10}(S_1)$, $\phi_{10}(S_2)$, $\phi_2(S_3)$ at t = 10.

Third, we can choose the one with maximum value among them by the termination step. If $\delta_{10}(S_2)$ is the maximal probability, the state at time t = 10 in the state sequence q_{10} should be S_2.

Finally, we can calculate the state sequence $q_9, q_8, \ldots, q_1$ via the path backtracking step. q_9 can be calculated by $\phi_{10}(S_2)$, which points to a state at t = 9 from the state S_2 at t = 10 based on the partial best path record. Consequently, we can use the partial best path record $\phi_9(q_9)$ to get q_8 and so on until q_1. Thus, $q_1, q_2, \ldots, q_{10}$ is obtained by this way, which is the most probable sequence of hidden states given the observation sequence and the HMM.

4.1.6 Expectation-Modification (EM) algorithm

Expectation-Modification (EM) algorithm is a method to adjust the model parameters to maximize the probability of the observation sequence given a HMM (Rabiner, 1989). The standard EM is the Baum-Welch algorithm, which uses an iterative procedure as follows:

1). Initialize the distribution parameters:

$\overline{\pi_i} = \gamma_1(i)$ indicates the expected number of times in the state S_i at time t = 1.

2). Iterative procedure includes expectation step (E-step) and modification step (M-step).

E-step: estimate the expected value of the unknown parameters, given the current parameter estimate.

$$\xi_t(i,j) = \frac{P(q_t = S_i, q_{t+1} = S_j, O \mid \lambda)}{P(O \mid \lambda)} = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O \mid \lambda)}$$
$$= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}$$

$\xi_t(i, j)$ indicates the probability of being in the state S_i at time t and the state S_j at time t+1.

$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$ indicates the probability of being in the state S_i at time t.

$\sum_{t=1}^{T-1} \gamma_t(i)$ indicates the expected number of transitions from S_i.

$\sum_{t=1}^{T-1} \xi_t(i,j)$ indicates the expected number of transitions from S_i to S_j.

M-step: re-estimate the distribution parameters to maximize the likelihood of the data. The following is a set of re-estimate formulas for π, A and B which are derived from the above formulas in E-step.

$\overline{\pi_i} = \gamma_1(i)$ indicates the expected number of times in the state S_i at time t = 1.

$\overline{a_{ij}} = \dfrac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$ $\sum_{t=1}^{T-1} \xi_t(i,j)$ indicates the expected number of transitions from S_i to S_j.

$\sum_{t=1}^{T-1} \gamma_t(i)$ indicates the expected number of transitions from S_i.

$\overline{b_j}(k) = \dfrac{\sum_{t=1,\, o_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$ $\sum_{t=1,\, o_t = v_k}^{T} \gamma_t(j)$ the expected number of times in the state *j* and observe V_k.

$\sum_{t=1}^{T} \gamma_t(j)$ the expected number of times in the state *j*.

Let's also use the previous example of the figure 11 to illustrate the iterative procedure of the EM algorithm. Now, we define the model $\lambda = (\pi, A, B)$, so we can use it to obtain those values in E-step and thereby compute the parameters via the formulas in M-step. Thus, the re-estimated model is defined as $\overline{\lambda} = (\overline{\pi}, \overline{A}, \overline{B})$, the parameters found on M-sep are used to start another E-sep, repeat this procedure until we find a new model $\overline{\lambda}$ which has maximum likelihood estimates of parameters.

4.2 Extentions of HMMs

Hidden Markov models (HMMs) have been known as one of the most widely used tools for modeling. However, the potentials of some new extended HMMs still need to be exploited. Subsequently, higher order HMMs, factorial HMMs, interpolated markov models (IMMs) and so on emerge as extensions of HMMs for overcoming the problems that can't be solved by the traditional HMMs.

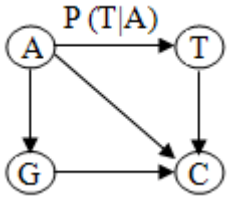
4.2.1 Higher order HMMs

Higher order HMMs process is a stochastic process in which the model moves from state to state depending only on the previous n states. So in an n-order HMM, the probability distribution of the next observation depends on the previous n observations:

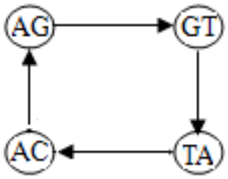
$$P(O_T \mid O_{T-1}O_{T-2}\ldots O_1) = P(O_T \mid O_{T-1}\ldots\ldots, O_{T-n})$$

Where $O = O_1 O_2 \ldots O_T \ldots$ is an observation sequence

Consider a first-order Markov model for sequences of four letters A, T, C and G, the probability of the next letter depends on what the previous letter generated was. We built this model by making a state for each letter (see the figure on the right). In this figure, we can see that the probability of the emission of letter T only depends on the previous letter A, which is P (T | A).



Consider a second-order Markov model for sequences of four letters A, T, C and G. A sequence AGTAC can be translated to a sequence of pairs AG-GT-TA-AC. Then we made four state Markov model that is part of second-order HMM (See the figure on the right), each state only have transitions to states whose first letter matched their second letter. In this figure, the state AG can only move to the next state GT because the first letter of the state GT is equal to the second letter of the state AG.



The framework of higher order models is more convinient than the first order models, although theoretically they are equivalent. Higher-order Markov models are particularly effective in extracting statistical characteristics of a given sequence. GeneMark used a fifth-order Markov model to achieve higher accuracy for detecting genes.

4.2.2 Factorial HMMs

Factorial HMMs (FHMMs) were first introduced by Ghahramani (Ghahramani and Jordan, 1996) and extend HMMs by allowing the modeling of several stochastic random processes decoupled. FHMMs are a multiple layers structure. In the figure 12, a FHMM with three layers is showed. We can see that each layer has independent state transitions but the observation at time step t can depend upon all state variables at that time step. For example, the observation Y_t depends on a combination of the

three states S_t^1, S_t^2 and S_t^3. This means the observation Y_t is determined by a conditional probability P ($Y_t | S_t^1, S_t^2, S_t^3$).

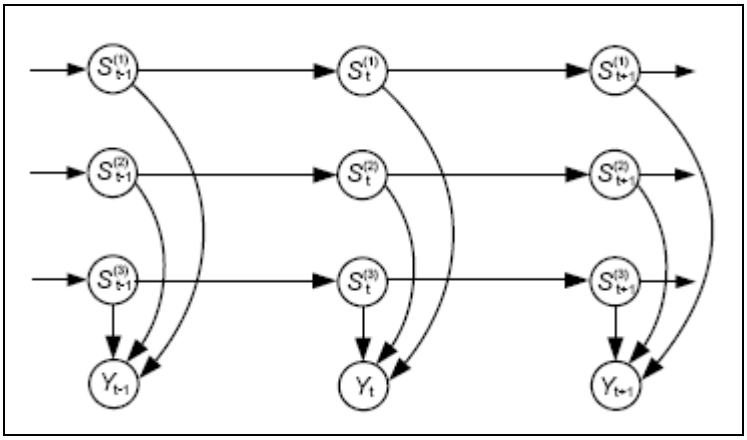


Fig.12 Factorial Hidden Markov Model (Ghahramani and Jordan, 1996). The FHMM represents the conditional independent relations with three layers. The circles $\{S_t\}$ are hidden states and the circles $\{Y_t\}$ are observation symbols.

In the figure 12, the factorial HMM consists of M=3 underlying Markov models. We assume that 3, 7, 112 different states each, the FHMM can be translated to a HMM with 3 * 7 * 112 = 2352 states. In order to describe observations, both FHMM and its equivalent HMM need a table with 2352 entities. More precisely, if each of the underlying Markov models of FHMM has state sets SS1, SS2, SS3, the state set of equivalent HMM is the Cartesian product SS1 * SS2 * SS3.

In the paper (Ghahramani and Jordan, 1996), authors conducted Bach's chorale experiment to investigate whether factorial HMMs provide an advantage over a regular HMM in modeling a real time series data set. It has been shown that FHMMs provide a more satisfactory model in three points of view. First, the models can be considered with significant larger state spaces. Second, in the factorial HMM, the large state spaces do not need excessively large numbers of parameters relative to the number of data. Finally, the factorial HMMs may achieve significantly better predictors than a single HMM. So the factorial HMM provides an advantage over traditional HMMs in modeling of predicting the complex patterns.

4.2.3 Interpolated Markov models

A higher order (k^{th}-order) HMM uses the k previous bases to predict the next base. It seems too difficult to accurately learn such models when there are no sufficient training data to accurately estimate the probability of each base occurring after every possible combination of k preceding bases. The number of probabilities we must estimate from the data increases exponentially when k increases. In order to overcome

this problem, a HMM based gene program called GLIMMER was developed in 1998 (Salzberg, *et al*., 1998). It uses interpolated Markov models (IMMs) as a framework to capture dependencies between nearby bases in a given sequence.

GLIMMER uses IMM-based method to make predictions based on contexts (oligomers) of variable lengths in a DNA sequence. Oligomer is termed as short, single stranded DNA fragments. It uses a linear combination of probabilities acquired from several lengths from 1 to 8 of oligomer to make predictions. And then, it gives high weigths to ligomers that occur frequently and low weights to those that do not. So IMMs can use a longer oligomer to make a prediction, which is more flexible and more powerful than higher-order Markov methods. Since oliomers with 8 bases length occur frequently, the IMM can use this longer oligomer to make a prediction. Once the weights are computed, GLIMMER evaluates new sequences by computing the proability of generating a sequence.

4.3 The PRISM system

PRISM (PRogramming In Statistical Modeling) is a logic-based statistical modeling language that integrates logic programming extended with random variables and parameter learning (Sato, 2007). The system is an extension of the B-Prolog system and developed by Sato and Kameya (Sato and Kameya, 2001). PRISM can estimate the probability distribution by using the EM algorithm to best explain the data from a given set of observed data. So PRISM appears as a powerful modeling environment, which includes HMMs, stochastic context free grammars (SCFGs), discrete Bayesian networks, etc., all embedded in a declarative language.

4.3.1 The PRISM programs

PRISM is a probabilistic extension of B-Prolog, extended with discrete random variables called msw's. Known models such as HMM and SCFG can be expressed in PRISM in straightforward ways. PRISM is embedded with statistical and machine learning techniques such as Viterbi and EM learning. So PRISM can run Viterbi computations to make predictions for a given sequence and the system can learn probabilities for the random variables of a model from a given sequence. PRISM programs are executed in a top-down left-to-right manner. The clauses of a PRISM program can be classified into three parts: modeling part, utility part and declarations. Modeling part corresponds to the definition of the model. It includes the definitions of all probabilistic predicates and of some non-probabilistic predicates which are called from probabilistic predicates. The following is an example of modeling part in a PRISM program from PRISM 1.12 user's manual chapter 7.1.

```
hmm(L):-              % To observe a string L:
    str_length(N),     %  Get the string length as N
```

```
    msw(init, S),      %  Choose an initial state randomly
    hmm(1,N,S,L).      %  Start stochastic transition (loop)

hmm(T,N,_,[]):- T>N,!. % Stop the loop
hmm(T,N,S,[O|Y]) :-    % Loop: The state is S at time T
msw(out(S),O),         % Output O at the state S
msw(tr(S),Next),       % Transit from the state S to Next.
T1 is T+1,             % Count up time
hmm(T1,N,Next,Y).      % Go next (recursion)
str_length(10).        % String length is 10
```

The modeling part is described above with several clauses. It expresses a probabilistic generation process for an output string in the HMM. For various probabilistic inferences, the modeling part works both in sampling execution and explanation search. The utility part is just a usual Prolog program with the system's built-ins.

Declarations are composed of several predefined predicates to give additional information to the system, which includes target declarations (observable probabilistic predicates), multi-valued switch declarations (outcome spaces of switches), etc. The following example form is used in the target declaration:

```
target(hmm,1).
```

This target declaration declares that the goals of the form *hmm (L)* will be observed, where *L* is a string to be generated.

The following example form is used in multi-valued switch declarations:

```
values(init, [s0,s1,s2]).   % state initialization
values(out (_), [c, g]).    % letter emission
values(tr(_), [s0,s1,s2]).  % state transition
```

The above three clauses declare three types of switches: the state initialization switches 'init' chooses 's0', 's1' or 's2'as an initial state to start, the letter emission switches 'out()' chooses 'c' or 'g' as an emitted letter at each state, and the state transition switches 'tr()' chooses 's0','s1'or 's2' as the next state.

The following extended form is also used in multi-valued switch declarations:

```
values_x(s, [1-10]).
```

The above clause declares a switch 's' whose value is chosen from a list of number 1 to number 10.

4.3.2 PRISM built-in utilities

PRISM possesses incomparable flexibility and powerful functionality. It provides several built-in utilities, such as random switches, sampling, probability calculation, Viterbi, hindsight computation, EM learning and so on. We focus on random switches, Viterbi and EM learning, which are mainly used in our PRISM models.

4.3.2.1 Random switches

Random switches are the most characteristic of PRISM, which make probabilistic choices. The built-in *msw (init, X)* makes probabilistic choice, that is, a random switch init and the variable *X* will achieve one of the possible values when this call is executed. To use the switch init, a multi-valued switch declaration must be predefined in the PRISM program, for example, *value (init, [s0, s1, s2])*. The probabilistic behavior of the random switch init is specified by the parameter $\theta_{init,X}$. This parameter can be set by using the built-in *set_sw(init,[0.11,0.26,0.63])*.That is, this will set 0.11 to the parameter of the first value of switch *init*, and set 0.26 and 0.63 to the parameter of the second and the third value respectively, where the order of values follows the predefined multi-valued switch declaration.

4.3.2.2 Viterbi computation

The efficient algorithm-Viterbi is well-known, which computes the most probable path of hidden state transitions given a string. By the Viterbi computation, we can use the built-in *viterbif/1* to get the most probable explanation *E** and its probability *P (E)* for a given goal *G*. If we run *viterbif (G)*, it will display the Viterbi probability and the Viterbi explanation for *G*. The following is an example of running Viterbi algorithm.

```
| ?- viterbif(hmm([c,c,c,c,c,g,g,g,g,g])).

hmm([c,c,c,c,c,g,g,g,g,g]
  <= hmm(1,10,s0,[c,c,c,c,c,g,g,g,g,g]) & msw(init,s0)
hmm(1,10,s0,[c,c,c,c,c,g,g,g,g,g])
  <= hmm(2,10,s0,[c,c,c,c,g,g,g,g,g]) & msw(out(s0),c) & msw(tr(s0),s0)
hmm(2,10,s0,[c,c,c,c,g,g,g,g,g])
  <= hmm(3,10,s0,[c,c,c,g,g,g,g,g]) & msw(out(s0),c) & msw(tr(s0),s0)

  …omitted…

hmm(10,10,s2,[g])
   <= hmm(11,10,s0,[]) & msw(out(s2),g) & msw(tr(s2),s0)
```

```
hmm(11,10,s0,[])

Viterbi_P = 0.0000001457835261
```

The first line is a command line in PRISM, which is used to call built-in predicate *viterbif*. The several followed lines indicate the Viterbi explanation for *hmm (L)*. It describes the state transition and emission processes that the model can generate the observed sequence *L*. The last line shows the probability of the most probable path that the model can generate this sequence.

4.3.2.3 Parameter learning

PRISM offers parameter learning, which is used to estimate the probability distributions to best explain observed data by using the EM algorithm. This power is suitable for several applications such as learning parameters of stochastic grammars, training stochastic models for gene sequence analysis, user modeling and so on.

Parameter learning is called maximum likelihood estimation (ML estimation). In ML estimation, the system tends to find the parameters θ that maximize the likelihood, the product of probabilities of given observed goals. If the observed goals are incomplete data, the system provides the utility of EM learning. It conducts learning in two phases: the first phase searches for all the explanations for observed data, and the second phase finds an ML estimate of θ by using the EM algorithm. The EM algorithm is an iterative process, which includes three steps-initialization step, expectation step and maximization step. The initialization step is used to initialize the parameters as $\theta^{(0)}$, and iterate the next two steps until the likelihood converges. The expectation step is used to compute the expected occurrences $\widehat{C}_{i,v}$ of msw(i,v) under the parameters $\theta^{(m)}$. The maximization step uses the expected occurrences to update each parameter by $\widehat{\theta}_{i,v}^{(m+1)} = \widehat{C}_{i,v} / \sum_{v'} \widehat{C}_{i,v'}$ and increases m by one. When the likelihood converges, the system will save the estimated parameters to its internal database, so we can make further probabilistic inferences based on these parameters.

We assume that we built a HMM in the system already like the model hmm (L) in §4.3.1. And then we can make a simple and artificial learning experiment as follows:

```
hmm_learn(N):-
set_params,!,                  % Set parameters manually
get_samples(N,hmm(_),Gs),!,    % Get N samples
learn(Gs).                     % learn with the samples

set_params :-
set_sw(init, [0.85,0.12,0.03]),
set_sw(tr(s0), [0.1,0.6,0.3]),
```

```
set_sw(tr(s1), [0.3,0.5,0.2]),
set_sw(tr(s2), [0.4,0.5,0.1]),
set_sw(out(s0),[0.3,0.7]),
set_sw(out(s1),[0.7,0.3]),
set_sw(out(s2),[0.7,0.3]).
```

In this experiment, we first give some predefined parameters to the HMM by `set_params`, and let the program to generate N samples under the parameters by using built in predicate `get_samples()`that calls `hmm/1` N times. Then we can learn the parameters from such sampled strings by `learn()`.

5. Evaluation of HMM based gene finders

Many gene prediction programs are currently available and are free to use them. In the past 15 years, number of competitive programs has been developed, updated in order to achieve 100% accuracy predictions. First, we give a general overview of different gene finding programs for predicting the prokaryotic genes, but we are not going to describe the detailed mathematical methods and algorithms. Later on, we use a data set to evaluate the accuracy of these programs for predicting the overlapping genes.

5.1 Introductions of gene finders

A HMM was first applied to gene finding in prokaryotes in 1994. It was used to find genes in the bacterium *E. coli* DNA sequences. Since only one strand was modeled, the HMM was applied twice to find genes in both strand-forward strand and reverse strand. They also considered the overlapping genes that 1 or 4 bases overlaps between the stop codon of one gene and the start codon of another gene. Two overlap HMMs were used to deal with the overlapping genes. They used this HMM based approach to predict about 80% of known *E. coli* genes, but about 5% genes are missed completely (Krogh, *et al*.,1994). Since then, a number of gene finding programs have been developed in order to predict more accurate and reliable genes.

The GeneMark.hmm program was developed in 1998. It applied generalized HMM to find genes in bacterial genomes. Both DNA strands-direct and reverse were dealed with simultaneously. The possibility of gene overlapping, at that time, was not considered because the authors thought no sufficient data were provided to obtain statistical model of overlapping genes in several possible orientations. This program was evaluated on several test data sets including 10 completely bacterial genomes.

The gene finding accuracy was improved since the ribosome binding site model was used to find exact gene starts (Lukashin and Borodovsky, 1998). From 1998 until now GeneMark.hmm has been updated in several times with different versions to improve the accuracy of gene prediction. It is a standard tool for gene prediction in new prokaryotic genomic sequences and provides more accurate and reliable gene prediction. The major characteristic of GeneMark.hmm is that two models, Typical and Atypical model are used to predict the different types of genes in parallel. They named 'Typical' and 'Atypical' that is because it was shown that the most of *E. coli* genes belong to the cluster of typical gene, while many genes that were potentially horizontally transferred into *E. coli* genome belong to the cluster of atypical gene. The atypical model can predict more genes that escape detection by the typical model.

Another gene finding program is EasyGene that was developed by Thomas Schou Larsen and Anders Krogh in 2003 (Larsen and Krogh, 2003). It is based on a HMM that is estimated with the Baum-Welch algorithm for a new genome. It is done by searching from protein matches in a new genome to get a good training set. Then the HMM parameters are estimated from the data set. The HMM is used to score all the open reading frames (ORFs)[4] which is also called coding region in the genome and the score is converted to R-value which indicates if it is a real gene or not.

Glimmer was first developed in 1997 (Salzberg, *et al.*, 1998). It uses interpolated Markov models (IMMs) to find coding regions. This IMM-based method detects genes based on a variable context. It solves the problem that occurs in the higher order Markov model, which is the number of probabilities that are estimated from the data increase exponentially. Glimmmer creates six IMMs for each possible reading frame and use them to score entire ORFs. Glimmer was updated into new version Glimmer 2.0 in 1999. It is much better than the first version for resolving the overlapping genes. It discards the genes with shorter ORF if another gene scores the highest wih the longest ORF on the overlap regions (Delcher, *et al*., 1999).

5.2 Evaluation results of gene finders

To evaluate the accuracy of the above programs, we prepared a data set that consists of four fragments of DNA sequences (Fig. 13). The results of running on different programs are showed in the table 1 (the raw results in Appendix 1).

[4] ORF is part of genomes that contains a sequence of bases encoding protein.

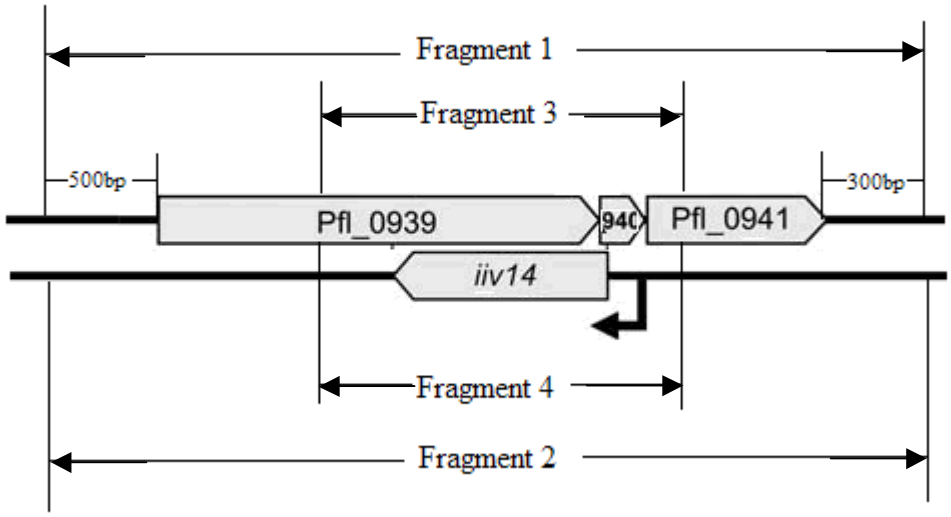


Fig.13 A data set consists of four fragments of DNA sequences. The sequences of fragments 1 and 3 are derived from the top strand by reading the bases from the left to right. The sequences of fragments 2 and 4 are derived from the down strand by reading the bases from the right to left.

The following is the respective position of the four fragments in the bacterium genome (*Pseudomonas fluorescens Pf0-1*), the numbers refer to nucleotides'positions in the genome:

Fragment 1: 1090732-1094788 (4057bp)
Fragment 2: 1094788-1090732 (4057bp)
Fragment 3: 1092032-1093714 (1683bp)
Fragment 4: 1093714-1092032 (1683bp)

Table 1 Evaluated gene prediction programs

Programs	Fragment 1	Fragment 2	Fragment 3	Fragment 4
	A B C O	A B C O	B O	B O
GeneMark	Yes Yes Yes No	Yes Yes Yes No	Yes No	Yes No
EasyGene	No No No No	No No No No	No No	No No
GLIMMER	No No No No	No No No No	No No	No No

'A' represents Pfl_0939 gene, 'B' represents Pfl_0940 gene,
'C' represents Pfl_0941 gene, 'O' represents iiv14 gene.

From the results, we can see that EasyGene and GLIMMER both predict several different genes comparing with GeneMark. For the fragment 1 and 2, EasyGene gives the totally different results. This means, given a DNA sequence, EasyGene will predict different genes for the direct strand and reverse strand. The genes predicted by EasyGene and GLIMMER are not identical with the four annotated genes. GeneMark gives the same results for the fragment 1 and 2, three genes Pfl_0939, Pfl_0940 and

Pfl_0941 can be more precisely predicted. For the fragment 3 and 4, GeneMark finds the same gene Pfl_0940. For the overlapping gene iiv14, none of three programs can predict it.

6. HMMs design

6.1 General HMM

In the previous sections, we summarized the common features of prokaryotic genes (Fig. 7). We build a general HMM (Fig. 14) based on those features.

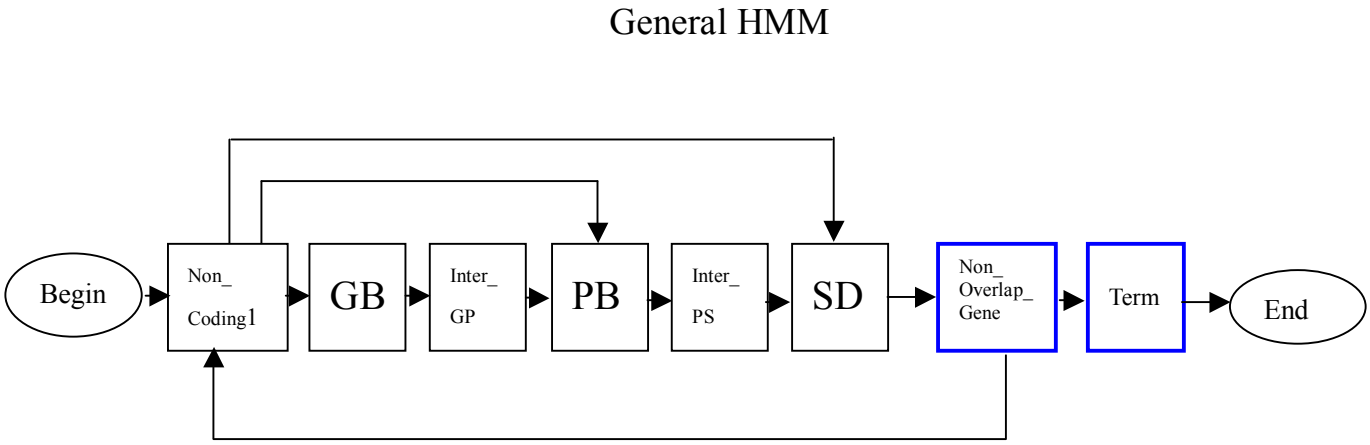


Fig.14. General HMM

The hidden states of the model are represented as boxes and ovals. Arrows represents the transitions between two states. Two states ‘’Begin’ and ‘End’ indicate the starting state and the ending state respectively. Blue boxes represent sub-models including ‘Non_Overlap_Gene’ and ‘Term’. ‘GB’ state indicates Gilbert Box; ‘PB’ state indicates Pribnow Box., ‘SD’ state indicates Shine Dalgarno sequence. Three states ‘Non_Coding1’, ‘Inter_GP’ and ‘Inter_PS’ indicate the sequences between two states.

First of all, we define ‘Begin’ and ‘End’ as states to describe the starting and ending of emission sequences respectively. Since there are four signals in the common features, and then we define them as states or sub-model respectively in the HMM. ‘GB’, ‘PB’ and ‘SD’ are defined as states and ‘Term’ is defined as sub-model. In the promoter site, Gilbert Box and Pribnow Box are located closely. So we define ‘GB’ and ‘PB’ as state to indicate these two boxes. The sequence of Shine Dalgarno-AGGAGG is used to predict the starting position of gene translation. We define ‘SD’ as a state to emit this sequence. As we know, terminator signal includes hairpin loop sequence and rho-dependent sequence. Thus we define ‘Term’ as a sub-model containing several

inner states to emit different sequences. For the coding region, we define 'Non_Overlap_Gene' as another sub-model to describe a gene sequence. In addition, 'Non_Coding1', 'Inter_GP' and 'Inter_PS' are defined as transition states to emit the sequences between the above states or sub-models. The sequences between 'Begin' and 'GB' are represented as 'Non_Coding1' state. The sequences between 'GB' and 'PB' are represented as 'Inter_GP' state. The sequences between 'PB' and 'SD' are represented as 'Inter_PS' state.

State transitions of the HMM are represented by arrows in the figure. One of transition patterns tends to pass through all states from left to right, that is, starting with 'Begin' state and ending with 'End' state. Transition arrows are showed in the middle of the model. Other transition pattern could be used to miss a few interim states therefore pass through directly the rest of states. 'Non_Coding1' state can be directly moving to 'PB' state without passing through 'GB' state, because a subclass of *E. coli* promoters functions quite well without recognizing GB region (Reviewed by DeHaseth, *et al*., 1998). 'Non_Coding1' state can be directly moving to 'SD' state without passing through 'GB' and 'PB', which is because several genes may share one promoter and each may have its own SD in front of start codon. 'Non_Overlap_Gene' sub-model may go back to 'Non_Coding1' state since several genes can share one terminator. 'Term' sub-model may go back to 'Non_Coding1' state, which is because a completed genome sequence contains one or more transition patterns.

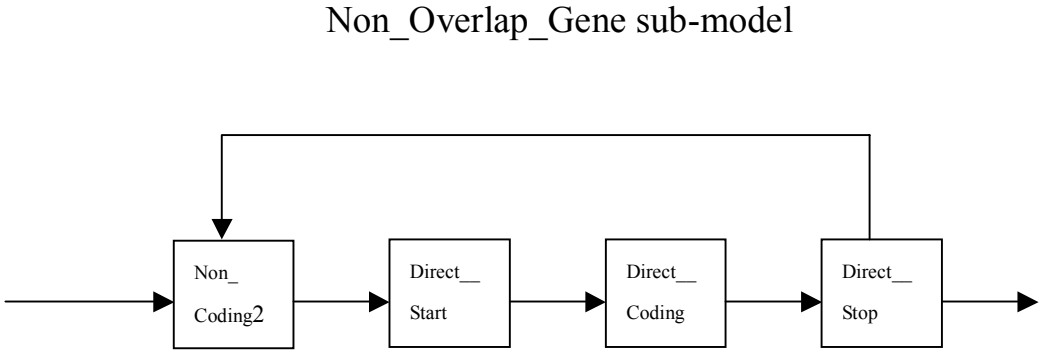


Fig.15. Non_Overlap_Gene sub-model

The model describes the genes are located in the DNA direct strand.

The states 'Direct_Start', 'Direct_Coding' and 'Direct_Stop' indicate start codon, coding sequences and stop codon of a gene in the DNA direct strand respectively. The state 'Non_Coding2' indicates the sequences in front of start codon. The transitions of the model are represented as arrows.

As we mentioned before, the coding region of a gene includes start codon, stop codon and several other codons between them. So we define 'Direct_Start', 'Direct_ Stop' and 'Direct_ Coding' as states to indicate the sequences of start codon, stop codon and several other codons respectively. As we know, the space between SD and start codon

is about 5-10 bp (Shine and Dalgarno, 1975), thus we define ‘Non_Coding2’ state to emit the sequences with 5-10 bases. State transitions of this sub-model are represented by arrows in the figure 15. The first state is ‘Non_Coding2’, which is moved to from an outer state. The transition continues to next three states in order. The last state is ‘Direct_Stop’, which will be moved to an outer state or moved back to ‘Non_Coding2’. We define this recursive transition due to many genes may be followed one by one without any signal emission between them.

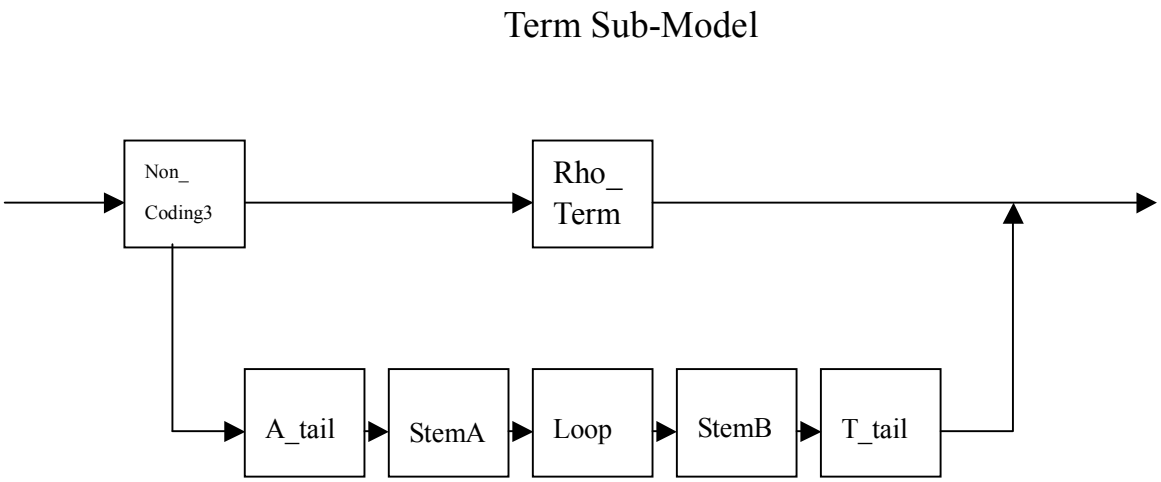


Fig.16. Term Sub-Model

‘Rho_Term’ indicates the Rho dependent sequence. Five states ‘A_tail’, ‘StemA’, ‘Loop’, ‘StemB’ and ‘T_tail’ indicate the specific sequences in the hairpin loop terminator. ‘Non_Coding3’ indicate the starting state that is moved to from outside state.

As we mentioned before, terminator signal includes rho-independent terminator called hairpin loop and rho-dependent terminator. The Rho dependent region has a sequence of about 40 to 60 bases that is rich in C residues and poor in G residues. So we define ‘Rho_Term’ as an inner state in ‘Term’ sub-model. The length of emitting sequences is about 30-60bp. The probability of emitting ‘C’ is the highest and the probability of emitting ‘G’ is the lowest. Hairpin loop terminator is more complicated than rho-dependent. First, Hairpin loop has palindromic DNA sequences about 30-60 bases after the stop codon, followed by around 8 ‘T’ bases. 15 bases on the 3' ends are called 'T-tail', while 15 bases on the 5' ends are called 'A-tail' (Kingsford, *et al*., 2007). Structure of hairpin loop terminator is displayed in the figure 17. So we define five states ‘A_tail’, ‘StemA’, ‘Loop’, ‘StemB’ and ‘T_tail’ based on this structure. Moreover, we find an exactly terminator sequence in *Bacillus subtilis* from an article (De Hoon, *et al*., 2005) and we draw a figure (Fig. 18). So we decide the length of ‘A_tail’ and ‘T_tail’ state emission is 6-15 bp, give a highest probability of ‘A’ in the ‘A_tail’ state and a highest probability of ‘T’ in the ‘T_tail’ state.
The length of ‘Loop’ state is 3-8 bp and high probabilities of ‘A’ and ‘T’ are required in this state. The length of ‘StemA’ and ‘StemB’ is 6 bp. The first base position of

'StemA' corresponds to the sixth base position of 'StemB'; the second corresponds to the fifth, and so on. Finally, the Term sub-model is showed in the figure 16. The model starts with 'Non_Coding3' state, which can be moved into from an outer state.

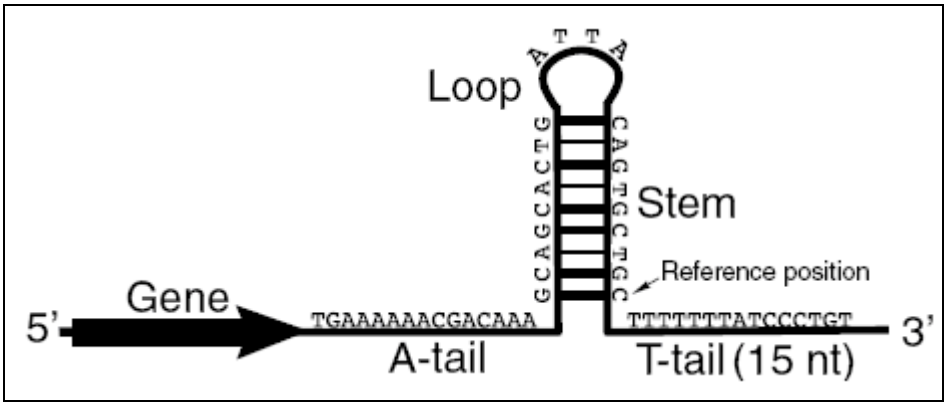


Fig. 17 Structure of hairpin loop terminator (adapted from Kingsford *et al*., 2007)
G-C pairing is rich in stem. A-tail contains several A bases and T-tail contains several T bases

A A A A A A T G G G G G T G T T A T C C C C C A T T T T T T
A-tail > > > > > > > loop < < < < < < T-tail
Stem Stem

Fig. 18 The terminator sequence downstream of *yqfT* gene in *Bacillus subtilis*
A-tail contains 6 A bases and T-tail contains 6 T bases.
Five G-C pairing is in the stem region.

6.2 Two genes overlap HMMs

In the previous sections, we illustrated the different overlapping patterns for two genes (Fig.8). We can design two_genes_overlap models based on those patterns. Since there are six overlapping patterns, six models are required for describing them respectively. In general，these six models have a common characteristic, that is, all of them contain the same states, 'Begin', 'End', 'Non_Coding', 'Inter_Region', 'Start_Codon_A', 'Coding_A' , 'Stop_Codon_A', 'Start_Codon_B', 'Coding_B' and 'Stop_Codon_B'. The difference of these six models is they have the different order of state transition due to the different overlapping patterns. For example, comparing pattern (a) and pattern (b) (Fig. 8), gene A and gene B partly overlap each other in pattern (a), whereas gene B entirely overlaps with gene A in pattern (b). Therefore, stop codon of gene A is encountered precede stop codon of gene B in the HMM of pattern (a), whereas stop codon of gene B is prior to stop codon of gene A in the HMM of pattern (b). In brief, 'Stop_Codon_A' state is prior to 'Stop_Codon_B' sate

in the HMM of pattern (a), in contrast, 'Stop_Codon_B' state is prior to 'Stop_Codon_A' sate in the HMM of pattern (b). Two_Genes_Overlap HMM of pattern (a) and (b) are showed as follows:

6.2.1 Two genes overlap HMM of pattern (a)

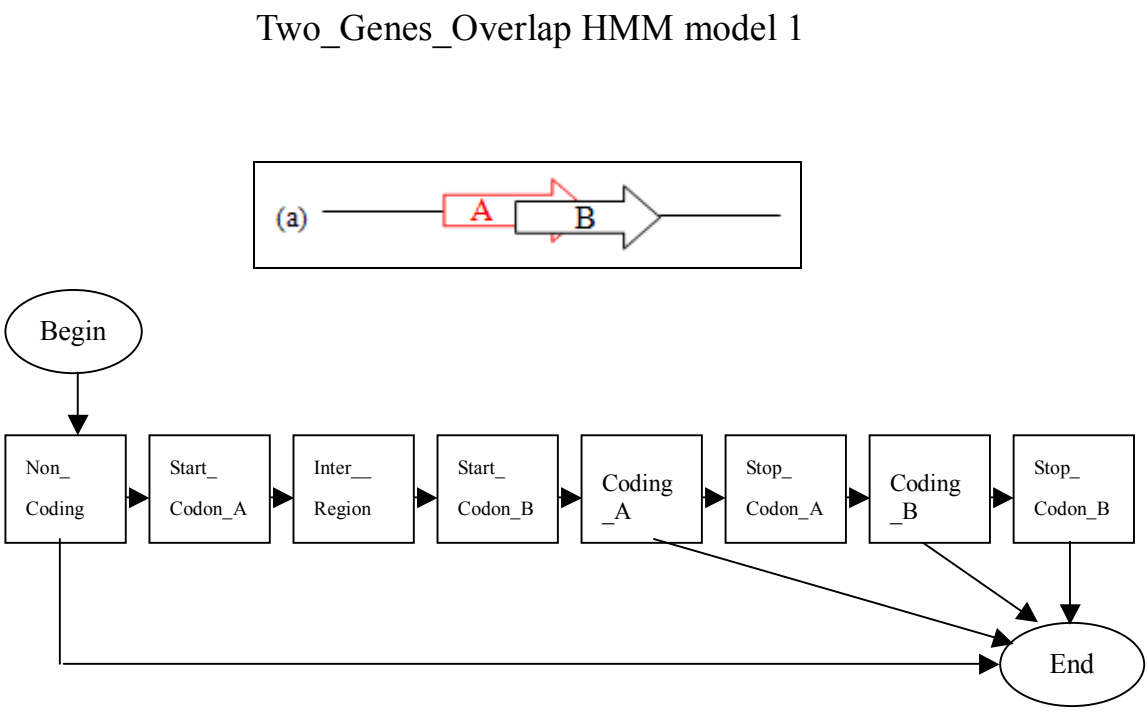


Fig.19. Two_Genes_Overlap HMM model 1

The model describes two genes overlap in the same strand, see pattern (a).

Two states ''Begin' and 'End' indicate the starting state and the ending state respectively. The states 'Non_Coding', 'Inter_Region' indicate the sequences between two states. The states 'Start_Codon_A', 'Coding_A' and 'Stop_Codon_A' indicate start codon, coding sequences and stop codon of gene A. The states 'Start_Codon_B', 'Coding_B' and 'Stop_Codon_B' indicate start codon, coding sequences and stop codon of gene B. The transitions of the model are represented as arrows.

Description of Overlap_Gene model_1:

The two genes gene A and gene B are localized in the same DNA direct strand. They are in the different reading frames, gene A is in frame 1 and gene B is in frame 2 or frame 3. The length between two start codons can be chosen from a list [1,4,7,10....], which will ensure gene A and B are in the reading frame 1 and frame 2 or 3.

6.2.2 Two genes overlap HMM of pattern (b)

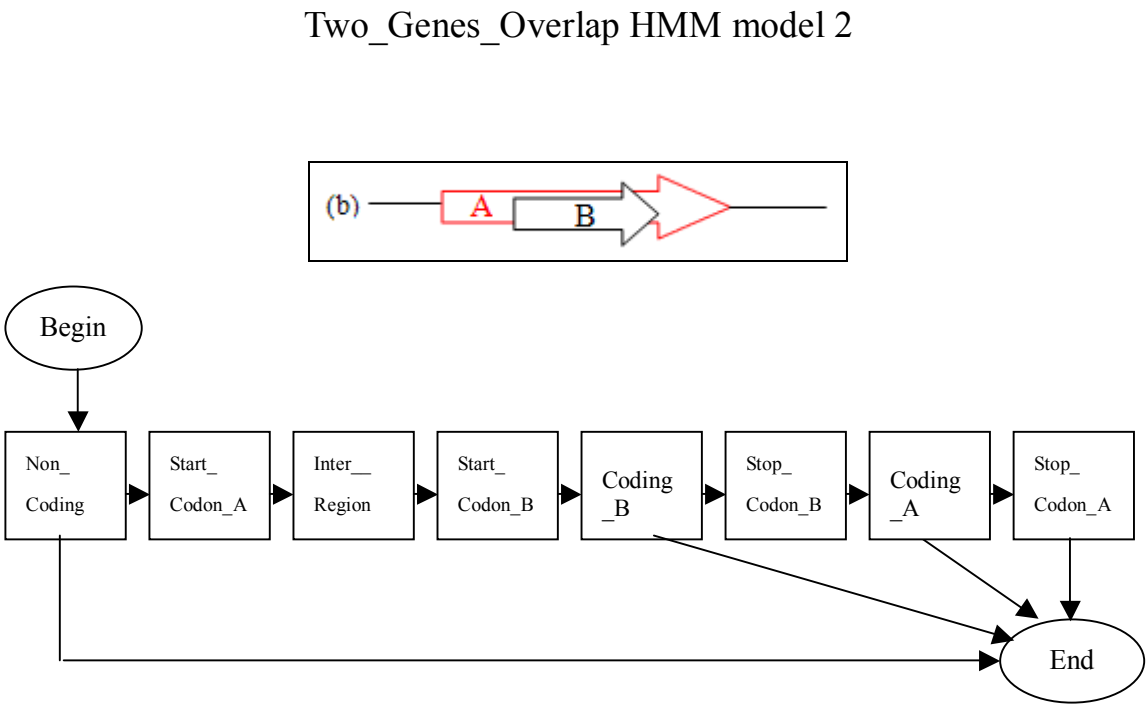


Fig.20. Two_Genes_Overlap HMM model 2

The model describes two genes overlap in the same strand, see pattern (b).

Two states ‘’Begin’ and ‘End’ indicate the starting state and the ending state respectively. The states ‘Non_Coding’, ‘Inter_Region’ indicate the sequences between two states. The states ‘Start_Codon_A’, ‘Coding_A’ and ‘Stop_Codon_A’ indicate start codon, coding sequences and stop codon of gene A. The states ‘Start_Codon_B’, ‘Coding_B’ and ‘Stop_Codon_B’ indicate start codon, coding sequences and stop codon of gene B. The transitions of the model are represented as arrows.

Description of Overlap_Gene model_2:

The two genes gene A and gene B are localized in the same DNA direct strand. They are in the different reading frames, gene A is in frame 1 and gene B is in frame 2 or frame 3. The length between two start codons can be chosen from a list [1,4,7,10....], which will ensure gene A and B are in the reading frame 1 and frame 2 or 3 respectively.

Comparing the above two models, we can easily see a little different that is the order of state transition. We have to build different HMMs for the rest of overlap patterns in order to detect two genes overlap each other. The rest of HMMs are not showed here since they have a little different with the above models.

7. Implementation of HMMs in PRISM

7.1 General PRISM model

As described in §6.1, the general HMM model we consider has several states and four emission letters 'a', 't', 'c' and 'g'. First of all, we classify these states into two groups based on their characteristic of letter emission. One is called 'position emission', letter emission at these states depends on a random switch named *emit (_Q, _P)*. The states in this group are 'GB', 'PB', 'SD', 'Direct_Start', 'Direct_stop', 'StemA' and 'StemB'. These states emit a certain number of letters at each time. For example, 'GB', 'PB', 'SD', 'StemA' and 'Stemb' states always emit six letters at each time. 'Direct_Start' and 'Direct_stop' always emit three letters at each time.

In the PRISM program (Appendix 2), we declare a random switch *emit (_Q, _P)* by the values declaration, which may choose one of four values 'a', 'c', 't' and 'g'. The following *set_sw* declaration assigns probabilities to the different possible values. In the modeling part *hmmGF(Q, Seq)*, the msw calls are used to make probabilistic choices.

```
values(emit(_Q,_P),['a','c','t','g']).
set_sw(emit('direct_start',1),[0.8,0.01,0.1,0.09]),
set_sw(emit('direct_start',2),[0.01,0.01,0.97,0.01]),
set_sw(emit('direct_start',3),[0.01,0.01,0.01,0.97]),
hmmGF(Q,Seq):- msw(trans(Q),Q1),
               msw(emit(Q1,1),N1),
               msw(emit(Q1,2),N2),
               msw(emit(Q1,3),N3).
```

In the above program, when the call *msw(emit(Q1,1),N1)* is executed, the variable N1 will be aligned to one of the possible values for a random switch *emit(Q1,1).* The three letters will be generated at the 'Direct_Start' state by calling the three msw's.

The other group is called 'length emission', letter emission at these states depends on a random switch named *period* that may take one of the possible values as emission length. The states in this group are 'Non_Coding1', 'Inter_GP' and 'Inter_PS', 'Non_Coding2, 'Direct_Coding', 'Non_Coding3', 'Rho_Term', 'A_tail', 'T-tail' and 'Loop'.
In the PRISM program (Appendix 2), we make several declarations for the constant values first:

```
non_coding_length(10).
```

```
inter_GP_length(21).
inter_PS_length(30).
direct_coding_length(250).
rho_length(60).
tailA_length(15).
tailT_length(15).
loop_length(10).
```

The above numbers in the brackets are maximum value of emission length for the different states, which can be used by the following values_x declaration. For example, the clause *inter_GP_length(21)* represents that the 'inter_GP' state emits at most 21 letters at each time. In the following values_x declaration, a random switch named *period(inter_GP)*, which may take one of the possible values from number 15 to 21 as an emission length of the 'inter_GP' state. The clause *inter_GP_length(L)* means to get the string length as L, corresponding to the constant value declaration *inter_GP_length(21),* L will be aligned to number 21.

```
values_x(period(inter_GP),[15-L]):-
        inter_GP_length(L).
```

In addition, the 'Direct_Coding' state is a little different from the other states about choosing the emission length. The number of letter emission at 'Direct_Coding' state should be divided by 3. So in the following values_x declaration, a predicate named *list_mod_3(L1,L)* is called to get a list of numbers `[3,6,9,12...]` that will be aligned to the variable L. Thus, a random switch *period(direct_coding)* may choose one of these numbers as an emission length of the 'Direct_Coding' state.

```
values_x(period(direct_coding),L):-
      direct_coding_length(L1),
      list_mod_3(L1,L).
```

Let's take 'Non_Coding1' state as an example to illustrate the process of generating a sequence based on the probabilistic event. First, we make several declarations:

```
target(hmmGF,1).
non_coding_length(10).                  % constant value 10
values_x(period(non_coding1),[0-L]) :-  % get emission length <= 10
        non_coding_length(L).            % get maximum length L = 10
values(trans(begin),[non_coding1]).      % state transition
values(emit(_Q),['a','c','t','g']).      % letter emission
set_sw(trans(begin),[1.0]),              % set probability to parameter
set_sw(emit('non_coding1'),[0.01,0.80,0.18,0.01]),
```

The above first clause is a target declaration, which declares that the event *hmmGF*

(Seq) will be observed, where *Seq* is a string to be generated. The second clause is used to set constant value 10, which means the maximum of emission length at the 'Non_Coding1' state is 10. The third clause is a multi-valued switch declaration, which tends to get an emission length of the 'Non_Coding1' state by a random choice. The followed two values declarations declare two types of switches: state transition switch called *trans* and letter emission switch called *emit*. The last two *set_sw* declarations assign probabilities to the different possible values corresponding to the two values declarations.We then proceed to the modeling part that is described as follows:

```
hmmGF(Seq) :- hmmGF(begin,Seq).    % To observe a string Seq
hmmGF(Q,Seq):-
       msw(trans(Q),Q1),           % Randomly choose the next state
       msw(period(Q1),L),          % Randomly choose emission length
       sequence_out(Q1,L,Seq-Seq1),% Generate a string with above length
       hmmGF(Q1,Seq1)).            % Go next (recursion)
```

This modeling part expresses a probabilistic generation process for an output sequence as the event *hmmGF(Seq)* depends on msw's. In the event *hmmGF,* a predicate named *sequence_out* is called to generate a string by outputting a letter in one loop.

```
sequence_out(_Q, 0,Seq-Seq) :-         % Stop the loop
        !.
sequence_out(Q, L,[N|Seq]-Seq1) :-     % Loop: State is Q with Length L
         !,
        msw(emit(Q),N),                % Output N at the state Q
        L1 is L-1,                     % Count down the length L
        sequence_out(Q, L1, Seq-Seq1). % Go next (recursion)
```

In a part of setting probabilities of the program, we used the statistic data from a published article, which has been figured out from experimental data. An article 'Analysis of *E. coli* promoter sequences' has been published by Harley and Reynolds since 1987. They have analyzed 263 promoters with two boxes GB and PB. The space between two boxes is about 15-20 bp. The distribution of specific bases at each position is displayed in the figure 21. We can set probabilities of two states GB and PB based on these values.

	T	T	G	A	C	A	T	A	T	A	A	T
T	78	82	15	20	10	24	82	7	52	14	19	89
G	10	5	68	10	7	17	7	1	12	15	11	2
C	9	3	14	13	52	5	8	3	10	12	21	5
A	3	10	3	58	32	54	3	89	26	59	49	3

Fig.21 The distribution of specific bases at each position (Harley and Reynolds, 1987)

The left part of the above figure represents six bases of GB box and the right part represents six bases of PB box. The left first column shows the distribution of base 'A', 'C', 'T' and 'G' at the first position of GB. A is 3 %, C is 9%, T is 78% and G is 10%. So we can set probabilities for the first position of GB in terms of these percentages in the program as follows:

```
set_sw(emit('gb',1),[0.03,0.09,0.78,0.10]),
```

In this way, we can set probabilities for six positions of GB and PB in the program based on statistic data of the figure 21.

```
set_sw(emit('gb',2),[0.10,0.03,0.82,0.05]),
set_sw(emit('gb',3),[0.03,0.14,0.15,0.68]),
set_sw(emit('gb',4),[0.58,0.12,0.20,0.10]),
set_sw(emit('gb',5),[0.31,0.52,0.10,0.07]),
set_sw(emit('gb',6),[0.54,0.05,0.24,0.17]),
set_sw(emit('pb',1),[0.03,0.08,0.82,0.07]),
set_sw(emit('pb',2),[0.89,0.03,0.07,0.01]),
set_sw(emit('pb',3),[0.26,0.10,0.52,0.12]),
set_sw(emit('pb',4),[0.59,0.12,0.14,0.15]),
set_sw(emit('pb',5),[0.49,0.21,0.19,0.11]),
set_sw(emit('pb',6),[0.03,0.05,0.89,0.03]),
```

7.2 Two genes overlap PRISM models

As described in §6.2, six different HMMs have a common characteristic, that is, all of them contain the same states, 'Begin', 'End', 'Non_Coding', 'Inter_Region', 'Start_Codon_A', 'Coding_A' , 'Stop_Codon_A', 'Start_Codon_B', 'Coding_B' and 'Stop_Codon_B'. The difference of these six models is they have the different order of state transition due to the different overlapping patterns. So these states can also be classified into two groups based on their characteristic of letter emission. 'Start_Codon_A', 'Stop_Codon_A', 'Start_Codon_B' and 'Stop_Codon_B' states belong to a group called 'position emission' because these states always emit three letters at each time.

In the PRISM program (Appendix 4), we declare a random switch *emit(_Q, _P)* by the values declaration, which may choose one of four values 'a', 'c', 't' and 'g'. The following *set_sw* declaration assigns probabilities to different possible values. As we know, the three types of start codon are 'atg', 'gtg' and 'ttg'. So we set a high probability 0.8 to the first value 'a' of the random switch *emit* at the first position of the 'start_codon_A' state. We set a high probability 0.97 to the third value 't' of the random switch *emit* at the second position of the 'start_codon_A' state. We set a high

probability 0.97 to the fourth value 'g' of the random switch *emit* at the third position of the 'start_codon_A' state.

```
values(emit(_Q,_P),['a','c','t','g']).
set_sw(emit('start_codon_A',1),[0.8,0.01,0.1,0.09]),
set_sw(emit('start_codon_A',2),[0.01,0.01,0.97,0.01]),
set_sw(emit('start_codon_A',3),[0.01,0.01,0.01,0.97]),
```

The rest of states such as 'Non_Coding', 'Inter_Region' 'Coding_A' and 'Coding_B' states belong to a group called 'length emission' because letter emission at these states depends on a random switch named *period* that may take one of the possible values as emission length.

In the PRISM program, we make several declarations for the constant values first:

```
non_coding_length(10).
coding_A_length(20).
coding_B_length(30).
inter_length(20).
```

In the following values_x declaration, a random switch named *period_frame2 (inter_region)*, which may take one of the possible values. A predicate named *list_frame_2(L1,L)* is called to get a list of numbers `[1,4,7,9...]` that will be aligned to the variable L. Thus, the random switch *period_frame2(inter_region)* may choose one of these numbers as an emission length of 'Inter_region'.

```
values_x(period_frame2(inter_region),L):-
      inter_length(L1),
      list_frame_2(L1,L).
```

In the modeling part *hmmGF(Q, Seq),* the msw calls are used to make probabilistic choices. As described in §6.2.1, this PRISM model is used to detect two genes- geneA and geneB overlap each other in the same DNA direct strand. So they must be located in the different reading frame, gene A is in frame 1 and gene B may be in frame 2 or frame 3. Thus, we declare a random switch *whichFrame(begin),* which may take one of the possible frames. We then call *msw(whichFrame(Q),F)* in the beginning of the modeling part, which may randomly choose a reading frame that will be align to the variable F.

```
values(whichFrame(begin),['frame2','frame3']).

hmmGF(Q,Seq):-
       msw(whichFrame(Q),F),
       ( F = 'frame2' -> hmmGF_frame2(Q,Seq)
```

```
        ;
         hmmGF_frame3(Q,Seq)).
```

If ‘frame2’ is randomly chosen, which means gene A is in the reading frame 1 and gene B is in the reading frame 2, so we need to consider the emission length of ‘Inter_region’, ‘Coding_A’ and ‘Coding_B’. According to the features of six reading frames in the figure 5 of §3.1, we make sure that the emission length of ‘Inter_region’ ‘Coding_A’ and ‘Coding_B’should be chosen from the list [1,4,7,10,…], [2,5,8,11,…] and [1,4,7,10,…] respectively. In contrast, if gene A is in the reading frame 1 and gene B is in the reading frame 3, the emission length of ‘Inter_region’ ‘Coding_A’ and ‘Coding_B’should be chosen from the list [2,5,8,11,…], [1,4,7,10,…] and [2,5,8,11,…] respectively. This can be done by the following declarations:

```
values_x(period_frame2(inter_region),L):-    % Choose length from L
      inter_length(L1),
      list_frame_2(L1,L).                    % L is a list of [1,4,7,10,…]

values_x(period_frame2(coding_A),L):-
      coding_A_length(L1),
      list_frame_3(L1,L).                    % L is a list of [2,5,8,11,…]

values_x(period_frame2(coding_B),L):-
      coding_A_length(L1),
      list_frame_2(L1,L).                     % L is a list of [1,4,7,10,…]
```

In the values_x declarations, two predicates named *list_frame_2(L1,L)* and *list_frame_3(L1,L)*are called for assigning to L a list of [1,4,7,10,…] and a list [2,5,8,11,…] respectively.

7.3 General PRISM model with annotations

As described in §7.1, general PRISM model is used to detect all possible genes in the DNA direct strand. In order to test this program, we obtain annotated sequences (Appendix 7.1) from NCBI (the National Center for Biotechnology Information) website. And then we can test this program to find out whether the annotated sequences contain any genes or not. When we run the general PRISM program and use Vitrbif predicate to get the most probable path way for a given sequence, we observed that the Viterbi explanation and proof tree are so complicated. Since we can not easily and clearly see the transition path, we are unable to come to the conclusion that the given sequence contains any genes. Fortunately, the Annotation System created by Henning Christiansen can solve this problem. This system contains

prismAnnot() predicate that makes Viterbi predications in the most efficient way. The user can arbitrarily add more annotation arguments anywhere in the model in order to get a concise Viterbi explanation that is what the user expected.

Therefore, in the general PRISM model with annotation (Appendix 3), we add two arguments, N and Ann. N denotes the position of each letter in a given sequence. Ann records the position of a state and emission letters by that state. In order to distinguish the annotation arguments and normal arguments, we add '--' in front of the annotation arguments. The purpose of running this program on a given sequence is able to know whether the given sequence contains any gene or 'signals ' and where they are. Therefore, in the program, we use argument Ann to record the position and emission letters of the 'direct_start' and the'direct_stop' states by the following codes:

```
hmmGF2(Seq,Q,--N,--Ann):-
       msw(emit(Q,1),L1),
       msw(emit(Q,2),L2),
       msw(emit(Q,3),L3),
       Seq = [L1,L2,L3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       -- Ann = [s(N,Q),e(N,L1,L2,L3)|Rest],
       hmmGF(Ls,Q,--N3,--Rest).
```

The above probabilistic predicate `hmmGF2(Seq,Q,--N,--Ann)`is called by the 'direct_start' and the 'direct_stop' states. --N and --Ann are two annotation arguments. N corresponds to a position of the 'direct_start' or the 'direct_stop' state. When `msw(emit(Q,1),L1)` this call is executed, the variable L1 will randomly choose one of the possible values as letter emission at the first position of the state. The other variable L2 and L3 will do the same as L1 to randomly choose one of possible letters. When these three variables are assign to their own value for letter emission. The argument Ann will record position and emission letters of that state by the following codes:

```
-- Ann = [s(N,Q),e(N,L1,L2,L3)|Rest]
```

Furthermore, we use the same way to record the position and emission letters for some 'signal' states such as 'PB', 'GB', 'SD' and so on. This process is carried out by the probabilistic predicate `hmmGF1(Seq,Q,--N,--Ann).`

Finally, let's run this program on the annotation system. Here, we do not explain how to use annotation system, so you may consult the article called 'on working with annotations to PRISM model' (Christiansen, 2008). After we load the program in the system, we can use the efficient Viterbi to find out whether a given sequence contains

any genes. In Appendix 7.3.1, the results of using the efficient Viterbi on three annotated sequences are showed up.
For the first annotated sequence (Data 1), we run the efficient Viterbi in the following way:

```
?- viterbiAnnot(hmmGF([Data 1],A),P).
```

```
A = [s(14,sd),e(14,t,g,g,a,g,g),s(24,direct_start),e(24,a,t,g),s(276,direct_stop
),e(276,t,a,c),s(290,stemA),e(290,t,g,g,g,g,g),s(302,stemB),e(302,c,c,c,c,c,a),s
(316,end)]
P = 0.0 ?
```

From the above result, we obtain the following information:
The 'signal' called SD starts at position 14, which is 'tggagg'. A gene starts at position 24 and stops at position 276. The start codon of the gene is 'atg' and the stop codon is 'tac'. Two parts of stem of hairpin terminator are stemA that starts at position 290 and stemB that starts at position 302.

As mentioned in §3.1, three types of stop codon are 'tag', 'taa' and 'tga'. However, the result indicates that the program detects 'tac' as a stop codon, 'tac' is not one of three possible stop codons. Furthermore, this sequence contains a gene that starts at position 31 and stops at position 283, which has already been experimentally annotated. From the above result, we can see the program finds a different position of the gene. So we need to modify this program to precisely detect the start codon and stop codon of a gene.

7.4 Modified general PRISM model

In the previous section §7.3, we observed that the general PRISM program was unable to more precisely detect a real gene. So we need to modify this program by following three aspects：
(1). Start codon of a gene is constrained on 'atg','ttg' and 'gtg'.
(2). Stop codon of a gene is constrained on 'taa','tag' and 'tga'.
(3). Coding region of a gene is constrained not to emit any stop codons.

First, we define 'direct_start' as start codon, 'direct_stop' as stop codon and 'direct_coding' as coding region of a gene. In this PRISM program (Appendix 5), we declare a random switch called `whichstart`, which value may be chosen from three possible types of start codon. In the set_sw declaration, we assign different probabilities to three possible start codons. The following is these two declarations:

```
values(whichstart,[['a','t','g'],['g','t','g'],['t','t','g']]).
set_sw(whichstart,[0.85,0.11,0.04]),
```

And then, a probabilistic predicate `hmmGF2(Seq,Q,--N,--Ann)` that depends on `msw`, which is called by the 'direct_start' state. When `msw(whichstart,S)` is executed, the variable `S` will randomly achieve one of the possible values as a start codon to emit. S contains three letters, which will be assigned to S1, S2 and S3 by `S = [S1,S2,S3]`. The annotation argument Ann will record position and three emission letters of the 'direct start' state by `-- Ann = [s(N,Q),e(N,S1,S2,S3)|Rest]`.

```
hmmGF2(Seq, Q, --N, --Ann) :-
       msw(whichstart,S),
       S = [S1,S2,S3],
       Seq = [S1,S2,S3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       -- Ann = [s(N,Q),e(N,S1,S2,S3)|Rest],
       hmmGF(Ls,Q,--N3,--Rest).
```

For the stop codon of a gene, we use the same method as the start codon to make probabilistic chose. The random switch is called `whichstop`.

As we know, each codon consists of three letters. The combination of triplet of 'a','t','c','g' is 64 possiblities, which includls three stop codons. For the coding region of a gene, we declare a random switch called `whichcodon`, which value can be chosen from 61 possible types of codons. These 61 possible codons do not contain the stop codons 'taa','tag' and 'tga'. Thus, we can make sure that the 'direct_coding' state does not emit any stop codons. And then, we make a call `msw(whichcodon,S)` to randomly chose one of 61 possible values as a codon to emit.

Compare with previous general PRISM program, the modified program can more precisely detect genes. The results of using the efficient Viterbi by modified program on three annotated sequences are showed in §Appendix 7.3.2. The following is the results of testing the modified program with the first annotated sequence.

```
A = [s(14,sd),e(14,t,g,g,a,g,g),s(31,direct_start),e(31,a,t,g),s(283,direct_stop
),e(283,t,a,a),s(290,stemA),e(290,t,g,g,g,g,g),s(302,stemB),e(302,c,c,c,c,c,a),s
(316,end)]
P = 0.0
```

At this time, the result indicates that the input sequence contains a gene that starts at position 31 and stops at position 283. Start codon of the gene is 'atg' and stop codon is 'taa'. This result is identical with experimental annotation. We use the same way to test the modified program with the other two sequences, and then we observed that the results are also identical with experimental annotation (Appendix 7.3.2).

7.5 Co-ordinated PRISM models

In the section §3.3, we described six patterns of two genes overlap each other. And then, we built two_genes_overlap HMMs based on these patterns in the section §6.2. In this approach, we only consider two genes overlap each other in a DNA segment. In addition, we ignore two genes' respective 'signals' such as 'pb','gb','sd' overlap each other. Actually, in a DNA segment, it could be more than two genes, but at most six genes in the different reading frames that overlap each other. Furthermore, these genes' 'signals' could also be overlap each other. So we can see that there are so many cases need to be considered in order to detect all possible overlapping genes. Obviously, it is unwise to use the previous approach.

7.5.1 Coordinating HMMs approach

An approach called coordinating HMMs (CHMMs) that was first defined by Henning Christiansen in an unpublished note called 'A first description of coordinating HMMs and other coordinating models' (Christiansen, 2008). CHMMs are suggested to run separate models in parallel, which is capable of describing arbitrary overlapping patterns. Overlapping genes may exist in six reading frames, three reading frames in each direction. We are interested in detecting all possible genes in six different reading frames. So we intend to build three HMMs to detect three reading frames in the DNA direct strand repectively (Fig.22). After reversing the sequence of the direct strand, we can detect the other three reading frames by running the same three HMMs on the reversed sequence.

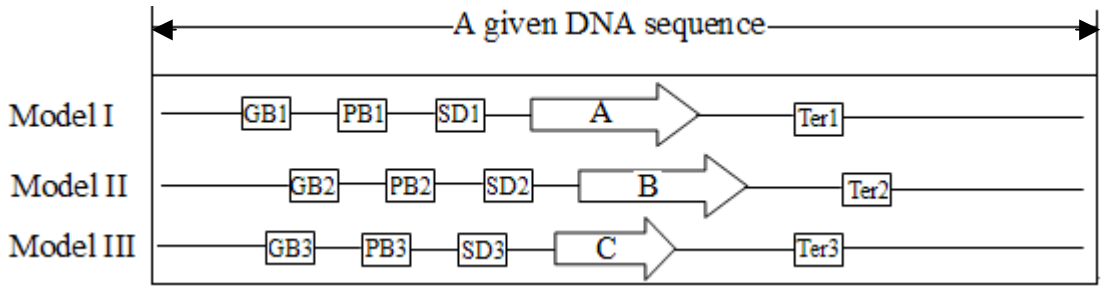


Fig. 22 Three models are used to detect three overlapping genes respectively. These three overlapping genes- gene A, B and C are located in a given DNA sequence. Model I is used to detect gene A of reading frame 1, Model II is used to detect gene B of reading frame 2 and Model III is used to detect gene C of reading frame 3.

These three HMMs are built with extending the modified PRISM model. The first model is used for detecting all possible genes of reading frame 1. The second and third models are used to detect all possible genes of reading frame 2 and reading frame 3 respectively. We want these three HMMs to run separately and independently, but prouce identical sequence at the same time. To be more precisely, these three

models can be combined into one model, which is putting these three models together by using different states' name and predicates. And then we can use the following codes to combine these three HMMs:

```
hmmGF(S):- equal_list(S1,S2,S3,S), hmmA(S1), hmmB(S2), hmmC(S3).
```

It will map a given sequence S into unique S1, S2 and S3 and find the best Viterbi explanation for it, combining the three sub-models `hmmA/1`, `hmmB/1` and `hmmC/1`. Here, `equal_list` means S=S1=S2=S3. So the probability of the combined explanation is the product of probabilities for the explanations provided by the individual explanations (Christiansen, 2008). E_{AB} denotes the explantion in the combined models, and E_A, E_B explanations in the sub-models.:

$$\arg\max_{E_{AB}}(P(E_{AB})) = \arg\max_{E_A}(P(E_A)) \times \arg\max_{E_B}(P(E_B))$$

Notice that, comparing Viterbi computations in the combined model, we will get the same result by just running the individual models separately. In the following section of testing models, we will show the results of running three individual models on a given sequence.

In the program (Appendix 6), we defined a new argument 'R'. R is a list of number 0, 1, 2, which represents different reading frames. A given sequence will be assigned to number 0, 1, and 2 in order. For example, the following is a given sequence and a list of number 0, 1, 2:

```
    a, c, t, c, g, g, a, c, c, t, t, g, c, a, t, g, c, a, c, t, g, c
R : 0, 1, 2, 0,  1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0
```

We use the value of R to know which reading frame a gene is in. For the above example, we assume the underlined 'a,t,g' is start codon of a gene. Since the first letter 'a' of the start codon corresponds to number 1 of R, we ensure that this gene is in the reading frame 2. The gene is in the reading frame 1 when the first letter of start codon corresponds to number 0 of R.

In this program, we use a call `R1 is (R+1) mod 3` to shift the number of R with letters emission. In some states such as 'non_coding1', 'inter_GP', 'inter_PS', etc, we use this call because these states emit a letter at each time. So the number of R will be changed from 0 to 1 because R is 0, (0+1) mod 3 will be 1, or changed from 1 to 2. In some states such as 'direct_start', 'direct_coding', 'direct_stop', etc, we don't need to use this call because these states emit 3 or multiples of 3 letters at each time, the number of R doesn't change until next state. For example, if the first emission letter of 'direct_start' corresponds to number 1 of R, after emitting three letters of this state, the first letter of next state will have the same number 1.

7.5.2 Coordinated PRISM model I

The coordinated PRISM model I is used to detect all possible genes of reading frame 1, so we only need to find all possible start codons whose first letter corresponds to number 0 of R. We find out that we may achieve it by making probabilistic chose for the emission length of 'non_coding2' state, which is previous state of 'direct_start'. In values_x declaration of this program, we declare three new random switches called period_frame1, period_frame2 and period_frame3, whose value can be chosen from a list of number 3, 6, 9, 12 …, a list of number 2, 5, 8, 11… and a list of number 1, 4, 7, 10…. The three groups of numbers are computed by three predicate `list_mod_3(L1,L)`, `list_frame_3(L1,L)` and `list_frame_2(L1,L)` respectively. The declarations are showed as follows:

```
values_x(period_frame1(non_coding2),L):-    % Choose length from L
      non_coding2_length(L1),
      list_mod_3(L1,L).                    % L is a list of [3,6,9,12,…]

values_x(period_frame2(non_coding2),L):-    % Choose length from L
      non_coding2_length(L1),
      list_frame_3(L1,L).                  % L is a list of [2,5,8,11,…]

values_x(period_frame3(non_coding2),L):-    % Choose length from L
      non_coding2_length(L1),
      list_frame_2(L1,L).                  % L is a list of [1,4,7,10,…]
```

In the modeling part of the program, we use a call `msw(trans(Q),Q1)` to make probabilistic state transition. When next state is 'non_coding2', the emission length of this state and corresponding R value both need to be considered at the same time. When the argument R is 0 at the first position of 'non_coding2', we should make a call `msw(period_frame1(Q1),Length)` to randomly chose emission length of the state from a list of number 3, 6, 9, 12 …. Thus, the argument R will be 0 at the first position of 'direct_start' state. In other words, the detected genes start at the position of R=0, which will make sure the detected genes are in the reading frame 1. The codes are showed as follows:

```
Q1 = 'non_coding2' ->
( R = 0 ->
  msw(period_frame1(Q1),Length),
  hmmGF3(Seq,Q1,R,Length,--N,--Ann)
```

When the argument R is 1 at the first position of 'non_coding2', we should make a

call `msw(period_frame2(Q1),Length)` to randomly chose emission length of the state from a list of number 2, 5, 8, 11 …. For example, if variable 'Length' is randomly assigned to number 2, this means the emission length of 'non_coding2' is 2 letters. The first letter corresponds to R=1, the second letter will be R=2. So the next R=0 will be at the first position of next state 'direct_start'. Thus, we use this way to make sure that only the genes of reading frame 1 can be detected. The codes are showed as follows:

```
R = 1 ->
msw(period_frame2(Q1),Length),
hmmGF3(Seq,Q1,R,Length,--N,--Ann)
```

When the argument R is 2 at the first position of 'non_coding2', we should make a call `msw(period_frame3(Q1),Length)` to randomly chose emission length of the state from a list of number 1, 4, 7, 10 …. Thus, we use the same way as above to make sure that only the genes of reading frame 1 can be detected. The codes are showed as follows:

```
R = 2 ->
msw(period_frame3(Q1),Length),
hmmGF3(Seq,Q1,R,Length,--N,--Ann)
```

7.5.3 Coordinated PRISM model II

The coordinated PRISM model II is used to detect all possible genes of reading frame 2, so we only need to find all possible start codons whose first letter corresponds to R=1. The following is the different part of model II comparing with model I:

```
Q1 = 'non_coding2' ->
( R = 0 ->
  msw(period_frame3(Q1),Length),
  hmmGF3(Seq,Q1,R,Length,--N,--Ann)
;
  R = 1 ->
  msw(period_frame1(Q1),Length),
  hmmGF3(Seq,Q1,R,Length,--N,--Ann)
;
  msw(period_frame2(Q1),Length),
  hmmGF3(Seq,Q1,R,Length,--N,--Ann)
)
```

The above codes describe that R=1 will always be at the first position of next state 'direct_start'. More precisely, the detected genes start at the position of R=1, which means the detected genes are in the reading frame 2. When R=0, emission length of

'non_coding2' will be randomly chosen from number 1, 4, 7, 10…. When R=1, emission length of 'non_coding2' will be randomly chosen from number 3, 6, 9, 12…. When R=2, emission length of 'non_coding2' will be randomly chosen from number 2, 5, 8, 11….

7.5.4 Coordinated PRISM model III

The coordinated PRISM model III is used to detect all possible genes of reading frame 3, so we only need to find all possible start codons whose first letter corresponds to R=2. The following is the different part of model III comparing with model I:

```
Q1 = 'non_coding2' ->
( R = 0 ->
  msw(period_frame2(Q1),Length),
  hmmGF3(Seq,Q1,R,Length,--N,--Ann)
;
  R = 1 ->
  msw(period_frame3(Q1),Length),
  hmmGF3(Seq,Q1,R,Length,--N,--Ann)
;
  msw(period_frame1(Q1),Length),
  hmmGF3(Seq,Q1,R,Length,--N,--Ann)
)
```

The above codes describe that R=2 will always be at the first position of next state 'direct_start'. More precisely, the detected genes start at the position of R=2, which means the detected genes are in the reading frame 3. When R=0, emission length of 'non_coding2' will be randomly chosen from number 2, 5, 8, 11…. When R=1, emission length of 'non_coding2' will be randomly chosen from number 1, 4, 7, 10…. When R=2, emission length of 'non_coding2' will be randomly chosen from number 3, 6, 9, 12….

7.5.5 Testing Coordinated PRISM models

In the previous sections, we described three coordinated PRISM models. Now, we want to analyze the same sequence by these models separately and independently. So these models can be run separately one by one in parallel.

The artificial sequence is showed as follows:

```
a,t,g,c,a,t,g,c,g,g,c,c,t,a,a,g,c,c,c,g,a,t,g,c,c,a,g,c,c,t,a,a,c,g,t,a,a
startC  startD          stopC          startE            stopE     stopD
```

The above sequence contains gene C, D and E that overlap each other. Gene C is in the reading frame 1, gene D is in the reading frame 2 and gene E is in the reading frame 3.

Let's run three programs named Coordinated PRISM model I, model II and model III on the above artificial sequence respectively:

```
viterbiAnnot(hmmGF([a,t,g,c,a,t,g,c,g,g,c,c,t,a,a,g,c,c,c,g,a,t,g,c,c
,a,g,c,c,t,a,a,c,g,t,a,a],A),P).
```

The result of running the above Viterbi by model I is showed as follows:

```
A = [s(1,direct_start),e(1,a,t,g),s(13,direct_stop),e(13,t,a,a),s(38,end)]
P = 0.0 ?
```

The result of running the above Viterbi by model II is showed as follows:

```
A = [s(5,direct_start),e(5,a,t,g),s(35,direct_stop),e(35,t,a,a),s(38,end)]
P = 0.0 ?
```

The result of running the above Viterbi by model III is showed as follows:

```
A = [s(5,sd),e(5,a,t,g,c,g,g),s(21,direct_start),e(21,a,t,g),s(30,direct_stop),e
(30,t,a,a),s(38,end)]
P = 0.0 ?
```

From above results, we can see that model I detects gene C of reading frame 1 that starts at position 1 and R=0, whereas model II detects gene D of reading frame 2 that starts at position 5 and R=1 and model III detects gene E of reading frame 3 that starts at position 21 and R=2.

In order to evaluate the accuracy of the three coordinated PRISM models, we prepared a data set that consists of five DNA sequences, Data 1, 2, 3, 4 and 5 (Appendix 7). The results of running on different models are showed in the following table (Table 2).

Table 2 Evaluated three coordinated PRISM models

Programs	Data 1 yqfT	Data 2 yaaB	Data 3 yaaA	Data 4 A B	Data 5 C D E
Model I	Yes	Yes	No	No Yes	Yes No No
Model II	No	No	Yes	No No	No Yes No
Model III	No	No	No	Yes No	No No Yes

Data 1, Data 2 and Data 3 are annotated sequences (Appendix 7.1), and each of them only contains one gene. Data 4 and Data 5 are artificial sequences (Appendix 7.2), which contains overlapping genes A, B, C, D and E.

From the evaluation results, we can see that yqfT gene of Data 1 and yaaB gene of Data 2 can be found by Model I, which is because they are located in the reading frame 1 and Model I is only used to detect genes of reading frame 1. The yaaA gene of reading frame 2 can be found by Model II. The artificial sequences of Data 4 contains two overlapping genes that are gene A of reading frame 3 and gene B of reading frame 1. The results show that gene A is detected by Model III and gene B is detected by Model I. It confirms that Model I is used to detect genes of reading frame 1 and Model III is used to detect genes of reading frame 3. The artificial sequences of Data 5 contains three overlapping genes that are gene C of reading frame 1, gene D of reading frame 2 and gene E of reading frame 3. These results further confirms that Model I precisely detect genes of reading frame 1, Model II precisely detect genes of reading frame 2 and Model III precisely detect genes of reading frame 3. Due to Viterbi computations in the combined model are equivalent to the results by just running the individual models separately, so we can get a conclusion about whether a give DNA sequence contains overlapping genes or not from the table 2. Our conclusion is the given DNA sequences of Data 4 and Data 5 contain overlapping genes, gene A and B overlap each other in Data 4 and gene C, D and E overlap each other in Data 5.

LOST project description

The project is funded research work which is a combination of computer science and molecular biology. The main goal of the project is to generate a new tool to detect genes by using PRISM. Traditional tools are based on HMMs and SCFGs, and all of the traditional tools have some limitations to detect genes. By using PRISM, the expressive power is improved a lot. So apparently, the project has a good opportunity to develop a better tool and even revolutionary work for molecular biology field
The project is built up by Professor Henning Christiansen and Professor John Gallagher from computer science at Roskilde University and associate Professor Ole Skovgaard from nature science at Roskilde University. The project includes also several Phd students, postdocs, external partners Krogh and Sato and some industry co-operators. It lasts for four years which is from 2007 to 2011.

Discussion

PRISM appears as a powerful and flexible modeling environment, which includes HMMs, stochastic context free grammars (SCFGs), discrete Bayesian networks, etc.,

all embedded in a declarative language. We built a general HMM in PRISM, which is used to detect all possible genes in the DNA direct strand. And then, we modified this model in order to more precisely detect a real gene. PRISM provides us its flexibility to easily build and modify HMMs. By utilizing PRISM's flexibility, we implemented co-ordinated three models that are used to detect all possible overlapping genes that might be in six reading frames of a DNA fragment.

PRISM has its limitation, that is, the Viterbi explanation in PRISM is unefficent. It is required to combine with other technique tool called annotation system created by Henning Christiansen to get efficient Viterbi explanation. Based on the results of efficient Viterbi explanation, we can know if the DNA sequence contains any overlapping genes by testing our co-ordinated models on a given DNA seuquence. Another problem is PRISM uses a lot of memories and takes so much space, because it is a very general system, which subsumes a variety of statistically based machine learning approaches such as HMMs, discrete Bayesian networks, stochastic context free grammars, etc., all embedded in a declarative language. So PRISM system is slow than specialized HMM tools.

Finally, we made three co-ordinated PRISM models that can be run separately one by one in parallel. So we could predict all possible overlapping genes in the different reading frames by these co-ordinated PRISM models. We tested these models on an artificial sequence that contains three overlapping genes in different reading frames. The test results are showed in §7.5.5, each model for a specific reading frame, three overlapping genes were detected by these three models respectively. So these co-ordinated PRISM models were successfully tested by the artificial sequence. We tried to test these models on a real data, which is a data set consists of four fragments of DNA sequences as mentioned in § 5 (Fig.13). These DNA fragments contain three overlapping genes pfl_0939, pfl_0940 and iiv14. Unfortunately, we could not get any results of efficient Viterbi computations by running these three models on these sequences. It seems complexity of Viterbi computation increases a lot when an input sequence is more than 2000 bases. If we have more time, we would also try to modify our models to satisfy the real data with three overlapping genes. Until now, we only used PRISM for HMMs; we could also use the full power of PRISM, such as stochastic context free grammars (SCFGs). We could use SCFGs to describe the structure of hairpin loop. As we know, the stem part of hairpin structure can not be described by HMMs, which is because HMMs are linear models. Thus, we may have chance for precisely detecting all possible genes by HMMs with combining SCFGs.

Conclusion

As the genetic information increasing rapidly, the biological analysis and researches

more and more rely on computational methods. Hidden Markov models (HMMs) have been proven as one of the most widely used methods in computational biology. Due to the limitation of the traditional HMMs, the complicated problems such as overlapping genes could not be successfully solved by HMM-based current gene finders.

We presented computational methods for modeling and analysis that combine statistic and machine learning with logic programming technology. We suggested using logic-statistical modeling system called PRISM (Sato and Kameya, 2001) for building and training HMMs. PRISM is a probabilistic version of Prolog, extended with discrete random variables called msw's. Known models such as HMM and SCFG can be expressed in PRISM in straightforward ways. PRISM is embedded with statistical and machine learning techniques such as Viterbi and EM learning. So PRISM can run Viterbi computations to make predictions for a given sequence and the system can learn probabilities for the random variables of a model from given sequences.

We made several experiments, in the first experiment (§7.1), we developed a general PRISM model to detect non_overlap genes in the DNA direct strand. In the second experiment (§7.2), we developed a two_genes_overlap PRISM model to detect two overlapping genes in a DNA fragment. In the third experiment (§7.3), we modified the general PRISM model by adding two annotation arguments in order to get efficient Viterbi computations to make predictions for a given sequence. From the results of running the efficient Viterbi by the model in the third experiment, we observed that this model was not able to precisely detect genes. So we modified this general PRISM model in the forth experiment (§7.4) by considering three conditions about start odon, stop codon and coding region in order to more precisely detect genes. As we know, in a DNA fragment, it could be more than two genes, but at most six genes in different reading frames that overlap each other. Furthermore, these genes' signals such as promoter, shine dalgarno sequence could also overlap each other. This means the two_genes_overlap PRISM model could only be used to make gene predictions for a specific case, which is two genes overlap in their coding regions without considering their signals.

We presented an approach called coordinating HMMs (CHMMs) that attempts to run separate models in parallel, which is capable of describing arbitrary overlapping structures. More precisely, several models run separately and independently, but at the same time produce identical squences. In this experiment (§7.5), we developed three coordinated PRISM models that are extensions of the modified general PRISM model in §7.4. The first coordinated model (§7.5.2) is used to detect all possible genes in the reading frame 1. The second model (§7.5.3) and the third model (§7.5.4) are used to detect all possible genes in the reading frame 2 and reading frame 3 respectively. Finally, we did test experiment based on an artificial sequence containing three overlapping genes by these three coordinated PRISM models. Based on separate analyses at the same time to the same sequence by these coordinated models, all

possible overlapping genes could be detected with considering their signals. Our experiments indicate that logic – statistical modeling method provides more expressive power and flexibility of modeling.

Acknowledgements

This work is supported by the project "Logic-statistic modeling and analysis of biological sequence data" (LOST) founded by the NABIIT program under the Danish Strategic Research Council. We would like to thank our supervisor, Henning Christiansen for some good suggestions, and Matthieu Petit for providing us a simple gene finder program in PRISM and also thank all LOST project members for helpful discussions.

References

Blattner F.R., *et al*. (1997) The Complete Genome Sequence of *Escherichia coli K-12*, Science 277, 1453-1474

Boyle R.D., (1984) HiddenMarkovModels, School of Computing Studies, University of Leeds, Leeds, LS2 9JT., website,
http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html

Christiansen H. (2008) Logical-Statistical models and parameter learning in the PRISM system. Roskilde University, Computer Science Dept.Course Note Version 29, Sep, 2008.

Christiansen H. (2008) On working with annotations to PRISM models. Roskilde University, Department of Communication, Business and Information Technologies. Preliminary Version, .

Christiansen H. and Dahmcke C.M., (2007) A Machine Learning Approach to Test Data Generation: A Case Study in Evaluation of Gene Finders. Leipzig, Germany, July 18-20, 2007

Christiansen H. (2008) A first description of coordinating HMMs and other coordinating models. Roskilde University, Department of Communication, Business and Information Technologies.November 14, 2008.

DeHaseth P.L., Zupancic M.L., Record M.T., Jr.(1998) RNA polymerase-promoter interactions: the comings and goings of RNA polymerase. J Bacteriol. 180 (12):3019-25.

De Hoon M.J., Makita Y., Nakai K., Miyano S. (2005) Prediction of transcriptional terminators in *Bacillus subtilis* and related species. PLoS Comput Biol. Aug; 1(3):e25. Epub.

Delcher A., Harmon D., Kasif S., White O. and Salzberg S. (1999) Improved microbial gene identification with GLIMMER. Nucl. Acids Res., 27(23):4636–4641.

Durbin R., Eddy S., Krogh A. and Mitchison G. (1998) Biological Sequence Analysis – Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press.

Fukuda Y., Washio T. and Tomita M. (1999) Comparative study of overlapping genes in the genomes of *Mycoplasma genitalium* and *Mycoplasma pneumoniae*. Oxford University Press, Nucleic Acids Resarc, vol. 27, No. 8

Ghahramani Z. and Jordan M. (1997) Factorial Hidden Markov Models. Kluwer Academic Publishers, Boston, Manufactured in the Netherlands. Machine Learning, 1-31.

Griffiths A.J.F., Lewontin R.C., Gelbart W.M. , Wessler S.R. , (2004) Introduction To Genetic Analysis 8th edition

Hannenhalli S.S., Hayes W.S., Hatzigeorgiou A.G., Fickett J.W. (1999) Bacterial start site prediction. Nucleic Acids Res. 1; 27(17):3577-82.

Harley C. B. and Reynolds R. P. (1987) Analysis of *E. coli* promoter sequences. Nucleic Acids Res. 15:2343–2361.

Hawley D. K. and McClure W. R. (1983) Compilation and analysis of *Escherichia coli* promoter sequences. Nucleic Acids Res. 11:2237–2255.

Kingsford C.L., Ayanbule K. and Salzberg S.L. (2007) Rapid, accurate, computational discovery of Rho-independent transcription terminators illuminates their relationship to DNA uptake. Genome Biology, 8:R22.

Krogh A., Mian I.S., Haussler D. (1994) A hidden Markov model that finds genes in *E. coli* DNA. Nucleic Acids Res, 22: 4768-4778

Larsen T.S. and Krogh A. (2003) EasyGene-a proaryoic gene finderthat ranks ORFs by statistical significance. BMC Bioinformatics 4: 21.

Lukashin A.V. and Borodovsky M. (1998) GeneMark.hmm: new solutions for gene finding. Nucleic Acids Res. 26: 1107-1115.

Nelson D. L. and Cox M. M., (2005) Lehninger Principles of Biochemistry 4th edition

Nielsen P. and Krogh A. (2005) Large-scale prokaryotic gene prediction and comparison to genome annotation. Bioinformatics 21: 4322-4329.

Rabiner L.R. (1989) "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". Proceedings of the IEEE , Vol. 77, No. 2, pp. 257-286

Rabiner L.R. and Juang B.H. (1986) An introduction to hidden markov models. IEEE ASSP Magazine. (3)1:4-15.

Shine J. and Dalgarno L. (1975) Determinant of cistron specificity in bacterial ribosomes. Nature 254, 34 - 38; doi:10.1038/254034

Salzberg S.L., Delcher A.L., Kasif S. and White O. (1998) Microbial gene identification using interpolated Markov models. Nucl. Acids Res., 26(2):544–548.

Sato T. (2007) PRISM, PRogramming In Statistical Modeling; web site, http://sato-www.cs.titech.ac.jp/prism/.

Sato T. and Kameya Y. (2001) Parameter learning of logic programs for symbolic-statistical modeling. J. Artif. Intell. Res. (JAIR), 15:391-454.

Siebenlist U. and Gilbert W. (1980) Contacts between *Escherichia coli* RNA polymerase and an early promoter of phage T7. Proc Natl Acad Sci U S A. 77(1): 122–126.

Silby M.W., Levy S.B. (2008) Overlapping Protein-Encoding Genes in *Pseudomonas fluorescens Pf0-1*. PLoS Genet 4(6): e1000094. doi:10.1371/ journal.pgen.1000094

Tan S. (2008) More about the Genetic Code - start codons, stop codons and degeneracy, website, http://cluelessaboutdna.blogspot.com/2008/01/more-about-genetic-code-start-codons.html

Wilson K.S. and von Hippel P.H. (1995) Transcription termination at intrinsic terminators: the role of the RNA hairpin. Proc Natl Acad Sci U S A. 12; 92(19):8793-7.

Appendix

Appendix 1 Evaluation of HMM based gene finders

Running GeneMark:
http://exon.gatech.edu/GeneMark/gmhmm2_prok.cgi

Fragment 1

Predicted genes

Gene #	Strand	LeftEnd	RightEnd	Gene Length	Class
1	+	<2	361	360	1
2	+	501	2693	2193	1
3	+	2683	2883	201	1
4	+	2894	3757	864	1
5	-	3802	>4056	255	1

Fragment 2

Predicted genes

Gene #	Strand	LeftEnd	RightEnd	Gene Length	Class
1	+	<2	256	255	1
2	-	301	1164	864	1
3	-	1175	1375	201	1
4	-	1365	3557	2193	1
5	-	3697	>4056	360	1

Fragment 3

Predicted genes

Gene #	Strand	LeftEnd	RightEnd	Gene Length	Class
1	+	<2	1393	1392	1
2	+	1383	1583	201	1
3	+	1594	>1683	90	1

Fragment 4

Predicted genes

Gene #	Strand	LeftEnd	RightEnd	Gene Length	Class
1	-	<1	90	90	1
2	-	101	301	201	1
3	-	291	>1682	1392	1

Running EasyGene:
http://www.cbs.dtu.dk/services/EasyGene/

Fragment 1

# seqname	model	feature	start	end	score	+/-	?	startc	odds
Fragment	EC03	CDS	525	1670	9.13781e-08	+	0	#ATG	37.6438
Fragment	EC03	CDS	2211	3005	0.0487023	+	0	#ATG	-11.0948

Fragment 2

# seqname	model	feature	start	end	score	+/-	?	startc	odds
Fragment	EC03	CDS	256	1086	0.0409014	+	0	#TTG	-17.8857
Fragment	EC03	CDS	3377	4126	0.19847	+	0	#TTG	-19.1132
Fragment	EC03	CDS	316	1083	6.85574e-12	-	0	#ATG	78.7408
Fragment	EC03	CDS	1129	1224	3.00165e-33	-	0	#ATG	69.755

Fragment 3

# seqname	model	feature	start	end	score	+/-	?	startc	odds
Fragment	EC03	CDS	89	1309	4.87121e-14	+	0	#GTG	88.1863
Fragment	EC03	CDS	425	1177	0.226943	-	0	#ATG	-22.5618

Fragment 4:

# seqname	model	feature	start	end	score	+/-	?	startc	odds
Fragment	EC03	CDS	270	1559	6.88083e-06	+	0	#ATG	-17.1972
Fragment	EC03	CDS	303	995	2.86011e-07	-	0	#GTG	52.5438

Running Glimmer:

http://www.ncbi.nlm.nih.gov/genomes/MICROBES/glimmer_3.cgi

GLIMMER (ver. 3.02; iterated) predictions:

orfID	start	end	frame	score
>Fragment 4				
orf00001	313	104	-2	8.43
orf00002	935	303	-3	7.52
orf00003	1408	1019	-2	2.48
orf00005	55	1478	-2	2.66

GLIMMER (ver. 3.02; iterated) predictions:

orfID	start	end	frame	score
>Fragment 4				
orf00001	280	104	-2	8.08
orf00002	935	303	-3	6.87
orf00003	1153	1019	-2	5.14
orf00004	1143	1559	+3	2.34

GLIMMER (ver. 3.02; iterated) predictions:

orfID	start	end	frame	score
>Fragment 3				
orf00002	311	102	-3	5.21
orf00004	1103	963	-3	5.12

Appendix 2 General PRISM model

```
%-------------
% General PRISM program
%-------------
% Written by Yuan Zhang and HongBo Liu, (c) 2008
%-------------
% Some ideas of this PRISM program design are obtained from
%  the MERGED model program written by Henning Christiansen
%  and the simple gene finder program written by Matthieu Petit
%-------------
% Model Description:
% - This model is used to detect non-overlap genes in DNA direct strand.
% - The probabilities of the sequences of Gilbert Box(gb) and Pribnow Box(pb)
%  are used to predict non_overlap genes in direct strand.
%  The space between two boxes is about 15-20 bp.
%  The above information is obtained from the article 'Analysis of E. coli
%  promoter sequences' written by Harley and Reynolds(1987)
% - The sequence of Shine Dalgarno(sd)-AGGAGG is used to predict the staring
%  position of gene translation
%  The space between sd and start codon is about 5-10 bp (Shine and Dalgarno,1975).
% -Given a short sequence(Seq), viterbif(Seq) can compute the most probable path.

 target(hmmGF,1).

% Maximal bounds of the emission length for some states
non_coding_length(10).
inter_GP_length(21).
inter_PS_length(30).
direct_coding_length(250).
rho_length(60).
tailA_length(15).
tailT_length(15).
loop_length(10).

% Transition of the general model
% State Description :
%  - 'begin' represents the start state without emitting any letters.
%  - 'non_coding1' represents the non_coding region that are the trasition
%    region before encounter a promoter.
%  - 'gb' represents Gilbert Box, it contains six letters 'TTGACA'.
%  - 'inter_GP' represents the region between Gilber Box and Pribnow Box.
%  - 'pb' represents Pribnow Box, it contains six letters 'TATAAT'.
```

```
%  - 'inter_PS' represents the region between Pribnow Box and Shine-Dalgarno.
%  - 'sd'represents Shine-Dalgarno sequence, it contains six letters 'AGGAGG'.
%  - 'non_coding2' represents the non_coding region in front of a start codon.
%  - 'direct_start' represents start codon of a gene in DNA direct strand, it
%    contains three letters.
%  - 'direct_coding' represents the coding region of a gene.
%  - 'direct_stop' represents the stop codon of a gene, it contains three letters.
%  - 'non_coding3' represents the non_coding region that are the trasition region
%    before encounter a terminator.
%  - 'rho_term' represents rho-dependent terminator, it contains 30-60 letters.
%  - 'tailA' represents a specific sequence of hairpin loop terminator, it contains
%    6-15 'A' letters.
%  - 'stemA' and 'stemB' represent G-C pairing sequence of hairpin loop terminator.
%  - 'tailT' represents a specific sequence of hairpin loop terminator, it contains
%   6-15 'T' letters.
%  - 'loop' represents a specific sequence of hairpin loop terminator, it contains
%   1-10 letters with more 'A' and 'T'.
%  - 'end' represents the ending state without emitting any letter.

values(trans(begin),[non_coding1]).
values(trans(non_coding1),[gb,pb,sd]).
values(trans(gb),[inter_GP]).
values(trans(inter_GP),[pb]).
values(trans(pb),[inter_PS]).
values(trans(inter_PS),[sd]).
values(trans(sd),[non_coding2]).
values(trans(non_coding2),[direct_start]).
values(trans(direct_start),[direct_coding]).
values(trans(direct_coding),[direct_stop]).
values(trans(direct_stop),[non_coding2,non_coding1,non_coding3]).
values(trans(non_coding3),[rho_term,tailA]).
values(trans(rho_term),[non_coding1,end]).
values(trans(tailA),[stemA]).
values(trans(stemA),[loop]).
values(trans(loop),[stemB]).
values(trans(stemB),[tailT]).
values(trans(tailT),[non_coding1,end]).

% Emission length of some states
values_x(period(non_coding1),[0-L]) :-
        non_coding_length(L).

values_x(period(non_coding2),[0-L]) :-
        non_coding_length(L).
```

```
values_x(period(non_coding3),[0-L]) :-
        non_coding_length(L).

values_x(period(inter_GP),[15-L]):-
      inter_GP_length(L).

values_x(period(inter_PS),[15-L]):-
      inter_PS_length(L).

values_x(period(direct_coding),L):-
      direct_coding_length(L1),
      list_mod_3(L1,L).

values_x(period(rho_term),[0-L]):-
      rho_length(L).

values_x(period(tailA),[0-L]):-
      tailA_length(L).

values_x(period(tailT),[0-L]):-
      tailT_length(L).

values_x(period(loop),[3-L]):-
      loop_length(L).

values(emit(_Q),['a','c','t','g']).
values(emit(_Q,_P),['a','c','t','g']).

set_parameters :-
       set_sw(trans(begin),[1.0]),
       set_sw(trans(non_coding1),[0.11,0.26,0.63]),
       set_sw(trans(gb),[1.0]),
       set_sw(trans(inter_GP),[1.0]),
       set_sw(trans(pb),[1.0]),
       set_sw(trans(inter_PS),[1.0]),
       set_sw(trans(sd),[1.0]),
       set_sw(trans(non_coding2),[1.0]),
       set_sw(trans(direct_start),[1.0]),
       set_sw(trans(direct_coding),[1.0]),
       set_sw(trans(direct_stop),[0.63,0.26,0.11]),
       set_sw(trans(non_coding3),[0.60,0.40]),
       set_sw(trans(rho_term),[0.85,0.15]),
       set_sw(trans(tailA),[1.0]),
```

```
set_sw(trans(stemA),[1.0]),
set_sw(trans(loop),[1.0]),
set_sw(trans(stemB),[1.0]),
set_sw(trans(tailT),[0.85,0.15]),
set_sw(emit('non_coding1'),[0.01,0.80,0.18,0.01]),
set_sw(emit('non_coding2'),[0.01,0.80,0.18,0.01]),
set_sw(emit('non_coding3'),[0.01,0.80,0.18,0.01]),
set_sw(emit('inter_GP'),[0.01,0.80,0.18,0.01]),
set_sw(emit('inter_PS'),[0.01,0.80,0.18,0.01]),
set_sw(emit('direct_coding'),[0.54,0.11,0.23,0.12]),
set_sw(emit('rho_term'),[0.07,0.80,0.12,0.01]),
set_sw(emit('tailA'),[0.97,0.01,0.01,0.01]),
set_sw(emit('tailT'),[0.01,0.01,0.97,0.01]),
set_sw(emit('loop'),[0.63,0.01,0.35,0.01]),
set_sw(emit('direct_start',1),[0.8,0.01,0.1,0.09]),
set_sw(emit('direct_start',2),[0.01,0.01,0.97,0.01]),
set_sw(emit('direct_start',3),[0.01,0.01,0.01,0.97]),
set_sw(emit('direct_stop',1),[0.01,0.01,0.97,0.01]),
set_sw(emit('direct_stop',2),[0.86,0.01,0.01,0.12]),
set_sw(emit('direct_stop',3),[0.36,0.01,0.01,0.62]),
set_sw(emit('gb',1),[0.03,0.09,0.78,0.10]),
set_sw(emit('gb',2),[0.10,0.03,0.82,0.05]),
set_sw(emit('gb',3),[0.03,0.14,0.15,0.68]),
set_sw(emit('gb',4),[0.58,0.12,0.20,0.10]),
set_sw(emit('gb',5),[0.31,0.52,0.10,0.07]),
set_sw(emit('gb',6),[0.54,0.05,0.24,0.17]),
set_sw(emit('pb',1),[0.03,0.08,0.82,0.07]),
set_sw(emit('pb',2),[0.89,0.03,0.07,0.01]),
set_sw(emit('pb',3),[0.26,0.10,0.52,0.12]),
set_sw(emit('pb',4),[0.59,0.12,0.14,0.15]),
set_sw(emit('pb',5),[0.49,0.21,0.19,0.11]),
set_sw(emit('pb',6),[0.03,0.05,0.89,0.03]),
set_sw(emit('sd',1),[0.94,0.02,0.02,0.02]),
set_sw(emit('sd',2),[0.02,0.02,0.02,0.94]),
set_sw(emit('sd',3),[0.02,0.02,0.02,0.94]),
set_sw(emit('sd',4),[0.94,0.02,0.02,0.02]),
set_sw(emit('sd',5),[0.02,0.02,0.02,0.94]),
set_sw(emit('sd',6),[0.02,0.02,0.02,0.94]),
set_sw(emit('stemA',1),[0.01,0.01,0.97,0.01]),
set_sw(emit('stemA',2),[0.01,0.01,0.01,0.97]),
set_sw(emit('stemA',3),[0.01,0.01,0.01,0.97]),
set_sw(emit('stemA',4),[0.01,0.01,0.01,0.97]),
set_sw(emit('stemA',5),[0.01,0.01,0.01,0.97]),
set_sw(emit('stemA',6),[0.01,0.01,0.01,0.97]),
```

```
       set_sw(emit('stemB',6),[0.01,0.01,0.97,0.01]),
       set_sw(emit('stemB',5),[0.01,0.01,0.01,0.97]),
       set_sw(emit('stemB',4),[0.01,0.01,0.01,0.97]),
       set_sw(emit('stemB',3),[0.01,0.01,0.01,0.97]),
       set_sw(emit('stemB',2),[0.01,0.01,0.01,0.97]),
       set_sw(emit('stemB',1),[0.01,0.01,0.01,0.97]).

hmmGF(Seq) :- hmmGF(begin,Seq).

hmmGF(Q,Seq):-
       msw(trans(Q),Q1),
       (    Q1 = 'end' ->
            Seq = []
       ;
          ( Q1 = 'gb';
            Q1 = 'pb';
            Q1 = 'sd';
            Q1 = 'stemA';
            Q1 = 'stemB') ->
            msw(emit(Q1,1),L1),
            msw(emit(Q1,2),L2),
            msw(emit(Q1,3),L3),
            msw(emit(Q1,4),L4),
            msw(emit(Q1,5),L5),
            msw(emit(Q1,6),L6),
            Seq = [L1,L2,L3,L4,L5,L6|Ls],
            hmmGF(Q1,Ls)
       ;
          ( Q1 = 'direct_start';
            Q1 = 'direct_stop') ->
            msw(emit(Q1,1),L1),
            msw(emit(Q1,2),L2),
            msw(emit(Q1,3),L3),
            Seq = [L1,L2,L3|Ls],
            hmmGF(Q1,Ls)
       ;
       msw(period(Q1),Length),
       sequence_out(Q1,Length,Seq-Seq1),
       hmmGF(Q1,Seq1)).

sequence_out(_Q, 0,Seq-Seq) :-
       !.

sequence_out(Q, Length,[L|Seq]-Seq1) :-
```

```
        !,
        msw(emit(Q),L),
        Length1 is Length-1,
        sequence_out(Q, Length1, Seq-Seq1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computation of all the numbers [3,6,9,12...] < Number divided by 3 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_mod_3(Number,[]) :-
        Number < 3.

list_mod_3(Number,[3|Rest]) :-
        Number1 is Number-6,
        list_mod_3_rec(Number1,6,Rest).

% Recursiv call
list_mod_3_rec(Number,Elt,[Elt]) :-
        Number < 3.

list_mod_3_rec(Number,Elt,[Elt|Rest]) :-
        Number1 is Number-3,
        Elt1 is Elt+3,
        list_mod_3_rec(Number1,Elt1,Rest).
```

Appendix 3 General PRISM model with annotations

```
%-------------
% General PRISM program with annotations
%-------------
% Written by Yuan Zhang and HongBo Liu, (c) 2008
%-------------

target(hmmGF,3).

non_coding_length(50).
non_coding2_length(20).
inter_GP_length(21).
inter_PS_length(30).
direct_coding_length(250).
rho_length(60).
tailA_length(15).
```

```
tailT_length(15).
loop_length(10).

values(trans(begin),[non_coding1]).
values(trans(non_coding1),[gb,pb,sd,end]).
values(trans(gb),[inter_GP]).
values(trans(inter_GP),[pb]).
values(trans(pb),[inter_PS]).
values(trans(inter_PS),[sd]).
values(trans(sd),[non_coding2]).
values(trans(non_coding2),[direct_start]).
values(trans(direct_start),[direct_coding]).
values(trans(direct_coding),[direct_stop]).
values(trans(direct_stop),[non_coding2,non_coding1,non_coding3]).
values(trans(non_coding3),[rho_term,tailA]).
values(trans(rho_term),[non_coding1,end]).
values(trans(tailA),[stemA]).
values(trans(stemA),[loop]).
values(trans(loop),[stemB]).
values(trans(stemB),[tailT]).
values(trans(tailT),[non_coding1,end]).

values_x(period(non_coding1),[0-L]) :-
       non_coding_length(L).

values_x(period(non_coding2),[0-L]) :-
       non_coding2_length(L).

values_x(period(non_coding3),[0-L]) :-
       non_coding_length(L).

values_x(period(inter_GP),[15-L]):-
      inter_GP_length(L).

values_x(period(inter_PS),[15-L]):-
      inter_PS_length(L).

values_x(period(direct_coding),L):-
      direct_coding_length(L1),
      list_mod_3(L1,L).

values_x(period(rho_term),[10-L]):-
      rho_length(L).
```

```
values_x(period(tailA),[0-L]):-
      tailA_length(L).

values_x(period(tailT),[0-L]):-
      tailT_length(L).

values_x(period(loop),[3-L]):-
      loop_length(L).

values(emit(_Q),['a','c','t','g']).
values(emit(_Q,_P),['a','c','t','g']).

      :-
       set_sw(trans(begin),[1.0]),
       set_sw(trans(non_coding1),[0.10,0.26,0.63,0.01]),
       set_sw(trans(gb),[1.0]),
       set_sw(trans(inter_GP),[1.0]),
       set_sw(trans(pb),[1.0]),
       set_sw(trans(inter_PS),[1.0]),
       set_sw(trans(sd),[1.0]),
       set_sw(trans(non_coding2),[1.0]),
       set_sw(trans(direct_start),[1.0]),
       set_sw(trans(direct_coding),[1.0]),
       set_sw(trans(direct_stop),[0.63,0.26,0.11]),
       set_sw(trans(non_coding3),[0.60,0.40]),
       set_sw(trans(rho_term),[0.85,0.15]),
       set_sw(trans(tailA),[1.0]),
       set_sw(trans(stemA),[1.0]),
       set_sw(trans(loop),[1.0]),
       set_sw(trans(stemB),[1.0]),
       set_sw(trans(tailT),[0.85,0.15]),
       set_sw(emit('non_coding1'),[0.01,0.80,0.18,0.01]),
       set_sw(emit('non_coding2'),[0.01,0.80,0.18,0.01]),
       set_sw(emit('non_coding3'),[0.01,0.80,0.18,0.01]),
       set_sw(emit('inter_GP'),[0.01,0.80,0.18,0.01]),
       set_sw(emit('inter_PS'),[0.01,0.80,0.18,0.01]),
       set_sw(emit('direct_coding'),[0.54,0.11,0.23,0.12]),
       set_sw(emit('rho_term'),[0.07,0.80,0.12,0.01]),
       set_sw(emit('tailA'),[0.97,0.01,0.01,0.01]),
       set_sw(emit('tailT'),[0.01,0.01,0.97,0.01]),
       set_sw(emit('loop'),[0.63,0.01,0.35,0.01]),
       set_sw(emit('direct_start',1),[0.8,0.01,0.1,0.09]),
       set_sw(emit('direct_start',2),[0.01,0.01,0.97,0.01]),
       set_sw(emit('direct_start',3),[0.01,0.01,0.01,0.97]),
```

```
        set_sw(emit('direct_stop',1),[0.01,0.01,0.97,0.01]),
        set_sw(emit('direct_stop',2),[0.86,0.01,0.01,0.12]),
        set_sw(emit('direct_stop',3),[0.36,0.01,0.01,0.62]),
        set_sw(emit('gb',1),[0.03,0.09,0.78,0.10]),
        set_sw(emit('gb',2),[0.10,0.03,0.82,0.05]),
        set_sw(emit('gb',3),[0.03,0.14,0.15,0.68]),
        set_sw(emit('gb',4),[0.58,0.12,0.20,0.10]),
        set_sw(emit('gb',5),[0.31,0.52,0.10,0.07]),
        set_sw(emit('gb',6),[0.54,0.05,0.24,0.17]),
        set_sw(emit('pb',1),[0.03,0.08,0.82,0.07]),
        set_sw(emit('pb',2),[0.89,0.03,0.07,0.01]),
        set_sw(emit('pb',3),[0.26,0.10,0.52,0.12]),
        set_sw(emit('pb',4),[0.59,0.12,0.14,0.15]),
        set_sw(emit('pb',5),[0.49,0.21,0.19,0.11]),
        set_sw(emit('pb',6),[0.03,0.05,0.89,0.03]),
        set_sw(emit('sd',1),[0.97,0.01,0.01,0.01]),
        set_sw(emit('sd',2),[0.01,0.01,0.01,0.97]),
        set_sw(emit('sd',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('sd',4),[0.97,0.01,0.01,0.01]),
        set_sw(emit('sd',5),[0.01,0.01,0.01,0.97]),
        set_sw(emit('sd',6),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',1),[0.01,0.01,0.97,0.01]),
        set_sw(emit('stemA',2),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',4),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',5),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',6),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',6),[0.01,0.01,0.97,0.01]),
        set_sw(emit('stemB',5),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',4),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',2),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',1),[0.01,0.01,0.01,0.97]).

hmmGF(Seq,--Ann) :- hmmGF(Seq,begin,--1,--Ann).

hmm([], end, --N, --[s(N,end)]).

hmmGF(Seq,Q,--N,--Ann):-
        msw(trans(Q),Q1),
        (    Q1 = 'end' ->
             Seq=[],
             hmm([], end, --N, --Ann)
        ;
```

```
          (Q1 = 'gb';
           Q1 = 'pb';
           Q1 = 'sd';
           Q1 = 'stemA';
           Q1 = 'stemB')  ->
           hmmGF1(Seq,Q1,--N,--Ann)
      ;
          (Q1 = 'direct_start';
           Q1 = 'direct_stop')  ->
           hmmGF2(Seq,Q1,--N,--Ann)
      ;
           msw(period(Q1),Length),
           hmmGF3(Seq, Q1,Length, --N, --Ann)).

hmmGF1(Seq,Q,--N,--Ann):-
      msw(emit(Q,1),L1),
      msw(emit(Q,2),L2),
      msw(emit(Q,3),L3),
      msw(emit(Q,4),L4),
      msw(emit(Q,5),L5),
      msw(emit(Q,6),L6),
      Seq = [L1,L2,L3,L4,L5,L6|Ls],
      -- N1 is N+1,
      -- N2 is N+2,
      -- N3 is N+3,
      -- N4 is N+4,
      -- N5 is N+5,
      -- N6 is N+6,
      -- Ann = [s(N,Q),e(N,L1,L2,L3,L4,L5,L6)|Rest],
      hmmGF(Ls,Q,--N6,--Rest).

hmmGF2(Seq,Q,--N,--Ann):-
      msw(emit(Q,1),L1),
      msw(emit(Q,2),L2),
      msw(emit(Q,3),L3),
      Seq = [L1,L2,L3|Ls],
      -- N1 is N+1,
      -- N2 is N+2,
      -- N3 is N+3,
      -- Ann = [s(N,Q),e(N,L1,L2,L3)|Rest],
      hmmGF(Ls,Q,--N3,--Rest).

hmmGF3([L|Ls], Q, Length,--N, --Ann):-
      !,
```

```
       msw(emit(Q), L),
       -- N1 is N+1,
       Length1 is Length-1,
       (    Length1 = 0  ->
            hmmGF(Ls,Q,--N1,--Ann)
       ;
       hmmGF3_rec(Ls, Q, Length1,--N1, --Ann)).

hmmGF3_rec([L|Ls], Q, Length,--N, --Ann):-
       !,
       msw(emit(Q), L),
       -- N1 is N+1,
       Length1 is Length-1,
       (    Length1 = 0  ->
            hmmGF(Ls,Q,--N1,--Ann)
       ;
       hmmGF3_rec(Ls, Q, Length1,--N1, --Ann)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computation of all the numbers [3,6,9,12...] < Number divided by 3 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_mod_3(Number,[]) :-
       Number < 3.

list_mod_3(Number,[210|Rest]) :-
       Number1 is Number-213,
       list_mod_3_rec(Number1,213,Rest).

% Recursiv call
list_mod_3_rec(Number,Elt,[Elt]) :-
       Number < 3.

list_mod_3_rec(Number,Elt,[Elt|Rest]) :-
       Number1 is Number-3,
       Elt1 is Elt+3,
       list_mod_3_rec(Number1,Elt1,Rest).
```

Appendix 4 Two genes overlap PRISM model

```
%-------------
% Two_Genes_Overlap PRISM program with annotations (Type I)
```

```
%-------------
% Written by Yuan Zhang and HongBo Liu, (c) 2008
%-------------
% Model Description: ------|------gene A------>------------
%                                |----gene B------->
%   -The two genes geneA and geneB are localized in the same DNA direct strand.
%   -They are in the different reading frames, gene A is in frame 1 and gene B
%    is in frame 2 or frame 3
%   -The length between two start codons can be chosen from a list [1,4,7,10....],
%    which will ensure gene A and B are in the  reading frame 1 and frame 2 or
%    3 respectively.
%   -Given a short sequence(Seq), viterbif(Seq) can compute the most probable path.

target(hmmGF,1).

% Maximal bounds of the emission length for some states
non_coding_length(10).
coding_A_length(20).
coding_B_length(30).
inter_length(20).

% Transition of this model
% State Description :
%     - 'begin' represents the start state without emitting any letters.
%     - 'non_coding' represents the non_coding region that are the region before
%       gene A starts and the region after gene B ends.
%     - 'start_codon_A' represents the start codon of gene A, it contains three
%       letters.
%     - 'inter_region' represents the region between start codon of gene A and
%       start codon of gene B.
%     - 'start_codon_B' represents the start codon of gene B, it contains three
%        letters.
%     - 'coding_A' represents the coding region of gene A including several letters.
%     - 'stop_codon_A' represents the stop codon of gene A, it contains three
%       letters.
%     - 'coding_B' represents the coding region of gene B including several letters
%     - 'stop_codon_B' represents the stop codon of gene B, it contains three
%       letters.
%     - 'end' represents the ending state without emitting any letter

% State transition declarations
values(trans(begin),[non_coding]).
values(trans(non_coding),[start_codon_A,end]).
values(trans(start_codon_A),[inter_region]).
```

```
values(trans(inter_region),[start_codon_B]).
values(trans(start_codon_B),[coding_A]).
values(trans(coding_A),[stop_codon_A,end]).
values(trans(stop_codon_A),[coding_B]).
values(trans(coding_B),[stop_codon_B,end]).
values(trans(stop_codon_B),[begin,end]).

% Emission length of some states
values_x(period(non_coding),[0-L]) :-
        non_coding_length(L).

values_x(period_frame2(inter_region),L):-
      inter_length(L1),
      list_frame_2(L1,L).

values_x(period_frame2(coding_A),L):-
      coding_A_length(L1),
      list_frame_3(L1,L).

values_x(period_frame2(coding_B),L):-
      coding_A_length(L1),
      list_frame_2(L1,L).

values_x(period_frame3(inter_region),L):-
      inter_length(L1),
      list_frame_3(L1,L).

values_x(period_frame3(coding_A),L):-
      coding_A_length(L1),
      list_frame_2(L1,L).

values_x(period_frame3(coding_B),L):-
      coding_A_length(L1),
      list_frame_3(L1,L).

% State emisstion declarations
values(emit(_Q),['a','c','t','g']).
values(emit(_Q,_P),['a','c','t','g']).

% Reading frame chosen declarations
values(whichFrame(begin),['frame2','frame3']).

% Set parameters
       :-
```

```
        set_sw(trans('begin'),[1.0]),
        set_sw(trans('non_coding'),[0.99,0.01]),
        set_sw(trans('start_codon_A'),[1.0]),
        set_sw(trans('inter_region'),[1.0]),
        set_sw(trans('start_codon_B'),[1.0]),
        set_sw(trans('coding_A'),[0.90,0.10]),
        set_sw(trans('stop_codon_A'),[1.0]),
        set_sw(trans('coding_B'),[0.80,0.20]),
        set_sw(trans('stop_codon_B'),[0.85,0.15]),
        set_sw(whichFrame('begin'),[0.5,0.5]),
        set_sw(emit('non_coding'),[0.01,0.80,0.18,0.01]),
        set_sw(emit('inter_region'),[0.01,0.80,0.18,0.01]),
        set_sw(emit('start_codon_A',1),[0.8,0.01,0.1,0.09]),
        set_sw(emit('start_codon_A',2),[0.01,0.01,0.97,0.01]),
        set_sw(emit('start_codon_A',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stop_codon_A',1),[0.01,0.01,0.97,0.01]),
        set_sw(emit('stop_codon_A',2),[0.86,0.01,0.01,0.12]),
        set_sw(emit('stop_codon_A',3),[0.36,0.01,0.01,0.62]),
        set_sw(emit('start_codon_B',1),[0.8,0.01,0.1,0.09]),
        set_sw(emit('start_codon_B',2),[0.01,0.01,0.97,0.01]),
        set_sw(emit('start_codon_B',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stop_codon_B',1),[0.01,0.01,0.97,0.01]),
        set_sw(emit('stop_codon_B',2),[0.86,0.01,0.01,0.12]),
        set_sw(emit('stop_codon_B',3),[0.36,0.01,0.01,0.62]),
        set_sw(emit('coding_A'),[0.01,0.80,0.18,0.01]),
        set_sw(emit('coding_B'),[0.01,0.80,0.18,0.01]).

hmmGF(Seq,--Ann) :- hmmGF(Seq,begin,--1,--Ann).

hmm([], end, --N, --[s(N,end)]).

hmmGF(Seq,Q,--N,--Ann):-
        msw(whichFrame(Q),F),
        (    F = 'frame2' ->
             hmmGF_frame2(Seq,Q,--N,--Ann)
        ;
        hmmGF_frame3(Seq,Q,--N,--Ann)
        ).

hmmGF_frame2(Seq,Q,--N,--Ann) :-
        msw(trans(Q),Q1),
        (    Q1 = 'end' ->
             Seq =[],
             hmm([], end, --N, --Ann)
```

```
       ;
           Q1 = 'begin' ->
           hmmGF(Seq,Q1,--N,--Ann)
       ;
           Q1 = 'non_coding' ->
           msw(period(Q1),Length),
           hmmGF2(Seq, Q1,Length, --N, --Ann)
       ;
          (Q1 = 'inter_region';
           Q1 = 'coding_A';
           Q1 = 'coding_B' )  ->
           msw(period_frame2(Q1),Length),
           hmmGF2(Seq, Q1,Length, --N, --Ann)
       ;
       hmmGF1(Seq,Q1,--N,--Ann)).

hmmGF_frame3(Seq,Q,--N,--Ann) :-
       msw(trans(Q),Q1),
       (    Q1 = 'end'  ->
           Seq = [],
           hmm([], end, --N, --Ann)
       ;
           Q1 = 'begin' ->
           hmmGF(Seq,Q1,--N,--Ann)
       ;
           Q1 = 'non_coding' ->
           msw(period(Q1),Length),
           hmmGF3(Seq, Q1,Length, --N, --Ann)
       ;
          (Q1 = 'inter_region';
           Q1 = 'coding_A';
           Q1 = 'coding_B' )  ->
           msw(period_frame3(Q1),Length),
           hmmGF3(Seq, Q1,Length, --N, --Ann)
       ;
       hmmGF4(Seq,Q1,--N,--Ann)).

hmmGF1(Seq,Q,--N,--Ann):-
       msw(emit(Q,1),L1),
       msw(emit(Q,2),L2),
       msw(emit(Q,3),L3),
       Seq = [L1,L2,L3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
```

```
        -- N3 is N+3,
        -- Ann = [s(N,Q),e(N,L1,L2,L3)|Rest],
        hmmGF_frame2(Ls,Q,--N3,--Rest).

hmmGF2([L|Ls], Q, Length,--N, --Ann):-
        !,
        msw(emit(Q), L),
        -- N1 is N+1,
        Length1 is Length-1,
        (    Length1 = 0 ->
             hmmGF_frame2(Ls,Q,--N1,--Ann)
        ;
        hmmGF2_rec(Ls, Q, Length1,--N1, --Ann)
        ).

hmmGF2_rec([L|Ls], Q, Length,--N, --Ann):-
        !,
        msw(emit(Q), L),
        -- N1 is N+1,
        Length1 is Length-1,
        (    Length1 = 0  ->
             hmmGF_frame2(Ls,Q,--N1,--Ann)
        ;
        hmmGF2_rec(Ls, Q, Length1,--N1, --Ann)
        ).

hmmGF3([L|Ls], Q, Length,--N, --Ann):-
        !,
        msw(emit(Q), L),
        -- N1 is N+1,
        Length1 is Length-1,
        (    Length1 = 0  ->
             hmmGF_frame3(Ls,Q,--N1,--Ann)
        ;
        hmmGF3_rec(Ls, Q, Length1,--N1, --Ann)
        ).

hmmGF3_rec([L|Ls], Q, Length,--N, --Ann):-
        !,
        msw(emit(Q), L),
        -- N1 is N+1,
        Length1 is Length-1,
        (    Length1 = 0  ->
             hmmGF_frame3(Ls,Q,--N1,--Ann)
```

```
       ;
       hmmGF3_rec(Ls, Q, Length1,--N1, --Ann)
       ).

hmmGF4(Seq,Q,--N,--Ann):-
       msw(emit(Q,1),L1),
       msw(emit(Q,2),L2),
       msw(emit(Q,3),L3),
       Seq = [L1,L2,L3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       -- Ann = [s(N,Q),e(N,L1,L2,L3)|Rest],
       hmmGF_frame3(Ls,Q,--N3,--Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Computation of all the numbers [1,4,7,10...] that makes two genes %
%%  in different reading frames.                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_frame_2(Number,[]) :-
       Number < 3.

list_frame_2(Number,[1|Rest]) :-
       Number1 is Number-6,
       list_frame_2_rec(Number1,4,Rest).

% Recursiv call
list_frame_2_rec(Number,Elt,[Elt]) :-
       Number < 3.

list_frame_2_rec(Number,Elt,[Elt|Rest]) :-
       Number1 is Number-3,
       Elt1 is Elt+3,
       list_frame_2_rec(Number1,Elt1,Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Computation of all the numbers [2,5,8,11...] that makes two genes %
%%  in different reading frames.                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_frame_3(Number,[]) :-
       Number < 3.
```

```
list_frame_3(Number,[2|Rest]) :-
       Number1 is Number-6,
       list_frame_3_rec(Number1,5,Rest).

% Recursiv call
list_frame_3_rec(Number,Elt,[Elt]) :-
       Number < 3.

list_frame_3_rec(Number,Elt,[Elt|Rest]) :-
       Number1 is Number-3,
       Elt1 is Elt+3,
       list_frame_3_rec(Number1,Elt1,Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computation of all the numbers [3,6,9,12...] < Number divided by 3 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_mod_3(Number,[]) :-
       Number < 3.

list_mod_3(Number,[3|Rest]) :-
       Number1 is Number-6,
       list_mod_3_rec(Number1,6,Rest).

% Recursiv call
list_mod_3_rec(Number,Elt,[Elt]) :-
       Number < 3.

list_mod_3_rec(Number,Elt,[Elt|Rest]) :-
       Number1 is Number-3,
       Elt1 is Elt+3,
       list_mod_3_rec(Number1,Elt1,Rest).
```

Appendix 5 Modified general PRISM model

```
%-------------
% Modified General PRISM model with annotations
%-------------
% Written by Yuan Zhang and HongBo Liu, (c) 2008
%-------------
% Model Description:
%   - Start codon of a gene is constrained on 'atg','ttg' and 'gtg'.
```

```
%    Emitted letters in 'direct_start' state have to satisfy the above constraint.
%  - Stop  codon of a gene is constrained on 'taa','tag' and 'tga'.
%    Emitted letters in 'direct_stop' state have to satisfy the above constraint.
%  - Coding region of a gene is constrained not to emit stop codons.

target(hmmGF,3).

non_coding_length(50).
non_coding2_length(20).
inter_GP_length(21).
inter_PS_length(30).
direct_coding_length(500).
rho_length(60).
tailA_length(15).
tailT_length(15).
loop_length(10).

values(trans(begin),[non_coding1]).
values(trans(non_coding1),[gb,pb,sd,non_coding2,end]).
values(trans(gb),[inter_GP]).
values(trans(inter_GP),[pb]).
values(trans(pb),[inter_PS]).
values(trans(inter_PS),[sd]).
values(trans(sd),[non_coding2]).
values(trans(non_coding2),[direct_start,end]).
values(trans(direct_start),[direct_coding]).
values(trans(direct_coding),[direct_stop]).
values(trans(direct_stop),[non_coding2,non_coding1,non_coding3,end]).
values(trans(non_coding3),[rho_term,tailA,non_coding1,end]).
values(trans(rho_term),[non_coding1,end]).
values(trans(tailA),[stemA,end]).
values(trans(stemA),[loop,end]).
values(trans(loop),[stemB]).
values(trans(stemB),[tailT]).
values(trans(tailT),[non_coding1,end]).

values_x(period(non_coding1),[0-L]) :-
        non_coding_length(L).

values_x(period(non_coding2),[0-L]) :-
        non_coding2_length(L).

values_x(period(non_coding3),[0-L]) :-
        non_coding_length(L).
```

```
values_x(period(inter_GP),[15-L]):-
      inter_GP_length(L).

values_x(period(inter_PS),[15-L]):-
      inter_PS_length(L).

values_x(period(direct_coding),L):-
      direct_coding_length(L1),
      list_mod_3(L1,L).

values_x(period(rho_term),[10-L]):-
      rho_length(L).

values_x(period(tailA),[0-L]):-
      tailA_length(L).

values_x(period(tailT),[0-L]):-
      tailT_length(L).

values_x(period(loop),[3-L]):-
      loop_length(L).

values(emit(_Q),['a','c','t','g']).
values(emit(_Q,_P),['a','c','t','g']).
values(whichstart,[['a','t','g'],['g','t','g'],['t','t','g']]).
values(whichstop, [['t','a','a'],['t','g','a'],['t','a','g']]).
values(whichcodon,[['a','a','a'],['a','c','a'],['t','t','t'],['t','c','a'],
                   ['a','a','t'],['a','c','t'],['t','t','a'],['t','c','t'],
                   ['a','a','c'],['a','c','c'],['t','t','g'],['t','c','c'],
                   ['a','a','g'],['a','c','g'],['t','t','c'],['t','c','g'],
                   ['a','t','a'],['a','g','a'],
                   ['a','t','t'],['a','g','t'],['t','a','t'],['t','g','t'],
                   ['a','t','c'],['a','g','c'],['t','a','c'],['t','g','c'],
                   ['a','t','g'],['a','g','g'],           ['t','g','g'],
                   ['c','c','c'],['c','t','a'],['g','g','g'],['g','t','a'],
                   ['c','c','a'],['c','t','c'],['g','g','a'],['g','t','c'],
                   ['c','c','t'],['c','t','g'],['g','g','t'],['g','t','t'],
                   ['c','c','g'],['c','t','t'],['g','g','c'],['g','t','g'],
                   ['c','a','c'],['c','g','a'],['g','a','g'],['g','c','a'],
                   ['c','a','t'],['c','g','t'],['g','a','a'],['g','c','t'],
                   ['c','a','a'],['c','g','c'],['g','a','t'],['g','c','g'],
                  ['c','a','g'],['c','g','g'],['g','a','c'],['g','c','c']]).
      :-
```

```
set_sw(trans(begin),[1.0]),
set_sw(trans(non_coding1),[0.11,0.15,0.63,0.10,0.01]),
set_sw(trans(gb),[1.0]),
set_sw(trans(inter_GP),[1.0]),
set_sw(trans(pb),[1.0]),
set_sw(trans(inter_PS),[1.0]),
set_sw(trans(sd),[1.0]),
set_sw(trans(non_coding2),[0.85,0.15]),
set_sw(trans(direct_start),[1.0]),
set_sw(trans(direct_coding),[1.0]),
set_sw(trans(direct_stop),[0.63,0.25,0.11,0.01]),
set_sw(trans(non_coding3),[0.24,0.26,0.35,0.15]),
set_sw(trans(rho_term),[0.85,0.15]),
set_sw(trans(tailA),[0.89,0.11]),
set_sw(trans(stemA),[0.89,0.11]),
set_sw(trans(loop),[1.0]),
set_sw(trans(stemB),[1.0]),
set_sw(trans(tailT),[0.85,0.15]),
set_sw(whichstart,[0.85,0.11,0.04]),
set_sw(whichstop, [0.85,0.11,0.04]),
set_sw(whichcodon,[ 0.016,0.0164,0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,      0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                    0.0164,0.0164,0.0164,0.0164,
                   0.0164,0.0164,0.0164,0.0164]),
set_sw(emit('non_coding1'),[0.01,0.80,0.18,0.01]),
set_sw(emit('non_coding2'),[0.01,0.80,0.18,0.01]),
set_sw(emit('non_coding3'),[0.01,0.80,0.18,0.01]),
set_sw(emit('inter_GP'),[0.01,0.80,0.18,0.01]),
set_sw(emit('inter_PS'),[0.01,0.80,0.18,0.01]),
set_sw(emit('direct_coding'),[0.54,0.11,0.23,0.12]),
set_sw(emit('rho_term'),[0.07,0.80,0.12,0.01]),
set_sw(emit('tailA'),[0.97,0.01,0.01,0.01]),
```

```
        set_sw(emit('tailT'),[0.01,0.01,0.97,0.01]),
        set_sw(emit('loop'),[0.63,0.01,0.35,0.01]),
        set_sw(emit('direct_start',1),[0.8,0.01,0.1,0.09]),
        set_sw(emit('direct_start',2),[0.01,0.01,0.97,0.01]),
        set_sw(emit('direct_start',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('direct_stop',1),[0.01,0.01,0.97,0.01]),
        set_sw(emit('direct_stop',2),[0.86,0.01,0.01,0.12]),
        set_sw(emit('direct_stop',3),[0.36,0.01,0.01,0.62]),
        set_sw(emit('gb',1),[0.03,0.09,0.78,0.10]),
        set_sw(emit('gb',2),[0.10,0.03,0.82,0.05]),
        set_sw(emit('gb',3),[0.03,0.14,0.15,0.68]),
        set_sw(emit('gb',4),[0.58,0.12,0.20,0.10]),
        set_sw(emit('gb',5),[0.31,0.52,0.10,0.07]),
        set_sw(emit('gb',6),[0.54,0.05,0.24,0.17]),
        set_sw(emit('pb',1),[0.03,0.08,0.82,0.07]),
        set_sw(emit('pb',2),[0.89,0.03,0.07,0.01]),
        set_sw(emit('pb',3),[0.26,0.10,0.52,0.12]),
        set_sw(emit('pb',4),[0.59,0.12,0.14,0.15]),
        set_sw(emit('pb',5),[0.49,0.21,0.19,0.11]),
        set_sw(emit('pb',6),[0.03,0.05,0.89,0.03]),
        set_sw(emit('sd',1),[0.97,0.01,0.01,0.01]),
        set_sw(emit('sd',2),[0.01,0.01,0.01,0.97]),
        set_sw(emit('sd',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('sd',4),[0.97,0.01,0.01,0.01]),
        set_sw(emit('sd',5),[0.01,0.01,0.01,0.97]),
        set_sw(emit('sd',6),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',1),[0.01,0.01,0.97,0.01]),
        set_sw(emit('stemA',2),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',4),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',5),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',6),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',6),[0.01,0.01,0.97,0.01]),
        set_sw(emit('stemB',5),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',4),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',2),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',1),[0.01,0.01,0.01,0.97]).

hmmGF(Seq,--Ann) :- hmmGF(Seq,begin,--1,--Ann).

hmm([], end, --N, --[s(N,end)]).

hmmGF(Seq,Q,--N,--Ann):-
```

```
       msw(trans(Q),Q1),
       (    Q1 = 'end' ->
            Seq=[],
            hmm([], end, --N, --Ann)
       ;
           (Q1 = 'gb';
            Q1 = 'pb';
            Q1 = 'sd';
            Q1 = 'stemA';
            Q1 = 'stemB')  ->
            hmmGF1(Seq,Q1,--N,--Ann)
       ;
            Q1 = 'direct_start' ->
            hmmGF2(Seq,Q1,--N,--Ann)
       ;
            Q1 = 'direct_stop'  ->
            hmmGF3(Seq,Q1,--N,--Ann)
       ;
            Q1 = 'direct_coding' ->
            msw(period(Q1),Length),
            hmmGF4(Seq,Q1,Length,--N,--Ann)
       ;
       msw(period(Q1),Length),
       hmmGF5(Seq, Q1,Length, --N, --Ann)
       ).

hmmGF1(Seq,Q,--N,--Ann):-
       msw(emit(Q,1),L1),
       msw(emit(Q,2),L2),
       msw(emit(Q,3),L3),
       msw(emit(Q,4),L4),
       msw(emit(Q,5),L5),
       msw(emit(Q,6),L6),
       Seq = [L1,L2,L3,L4,L5,L6|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       -- N4 is N+4,
       -- N5 is N+5,
       -- N6 is N+6,
       -- Ann = [s(N,Q),e(N,L1,L2,L3,L4,L5,L6)|Rest],
       hmmGF(Ls,Q,--N6,--Rest).

% Constraint on start codon 'atg','gtg' and 'ttg'
```

```
hmmGF2(Seq, Q, --N, --Ann) :-
       msw(whichstart,S),
       S = [S1,S2,S3],
       Seq = [S1,S2,S3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       -- Ann = [s(N,Q),e(N,S1,S2,S3)|Rest],
       hmmGF(Ls,Q,--N3,--Rest).

% Constraint on stop codon 'taa','tag' and 'tga'
hmmGF3(Seq, Q, --N, --Ann) :-
       msw(whichstop,S),
       S = [S1,S2,S3],
       Seq = [S1,S2,S3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       -- Ann = [s(N,Q),e(N,S1,S2,S3)|Rest],
       hmmGF(Ls,Q,--N3,--Rest).

hmmGF4(Seq, Q, Length, --N, --Ann) :-
       !,
       msw(whichcodon,S),
       S = [S1,S2,S3],
       Seq = [S1,S2,S3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       Length1 is Length-3,
       (    Length1 = 0 ->
            hmmGF(Ls,Q,--N3,--Ann)
       ;
       hmmGF4_rec(Ls, Q, Length1,--N3, --Ann)
       ).

hmmGF4_rec(Seq, Q, Length, --N, --Ann) :-
       !,
       msw(whichcodon,S),
       S = [S1,S2,S3],
       Seq = [S1,S2,S3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
```

```
       Length1 is Length-3,
       (    Length1 = 0 ->
            hmmGF(Ls,Q,--N3,--Ann)
       ;
       hmmGF4_rec(Ls, Q, Length1,--N3, --Ann)
       ).

hmmGF5([L|Ls], Q, Length,--N, --Ann):-
       !,
       msw(emit(Q), L),
       -- N1 is N+1,
       Length1 is Length-1,
       (    Length1 = 0 ->
            hmmGF(Ls,Q,--N1,--Ann)
       ;
       hmmGF5_rec(Ls, Q, Length1,--N1, --Ann)
       ).

hmmGF5_rec([L|Ls], Q, Length,--N, --Ann):-
       !,
       msw(emit(Q), L),
       -- N1 is N+1,
       Length1 is Length-1,
       (    Length1 = 0 ->
            hmmGF(Ls,Q,--N1,--Ann)
       ;
       hmmGF5_rec(Ls, Q, Length1,--N1, --Ann)
       ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computation of all the numbers [3,6,9,12...] < Number divided by 3 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_mod_3(Number,[]) :-
      Number < 3.

list_mod_3(Number,[150|Rest]) :-
      Number1 is Number-153,
      list_mod_3_rec(Number1,153,Rest).

% Recursiv call
list_mod_3_rec(Number,Elt,[Elt]) :-
      Number < 3.
```

```
list_mod_3_rec(Number,Elt,[Elt|Rest]) :-
      Number1 is Number-3,
      Elt1 is Elt+3,
      list_mod_3_rec(Number1,Elt1,Rest).
```

Appendix 6 Coordinated PRISM models

```
%-------------
% Coordinated PRISM Model I
%-------------
% Written by Yuan Zhang and HongBo Liu, (c) 2008
%-------------
% Model Description:
%   - This model is used for detecting all possible genes
%    are in the reading frame 1 of DNA direct strand.

target(hmmGF,3).

non_coding_length(50).
non_coding2_length(20).
inter_GP_length(21).
inter_PS_length(30).
direct_coding_length(500).
rho_length(60).
tailA_length(15).
tailT_length(15).
loop_length(10).

values(trans(begin),[non_coding1,non_coding2,direct_start]).
values(trans(non_coding1),[gb,pb,sd,end]).
values(trans(gb),[inter_GP]).
values(trans(inter_GP),[pb]).
values(trans(pb),[inter_PS]).
values(trans(inter_PS),[sd]).
values(trans(sd),[non_coding2]).
values(trans(non_coding2),[direct_start,end]).
values(trans(direct_start),[direct_coding]).
values(trans(direct_coding),[direct_stop]).
values(trans(direct_stop),[non_coding2,non_coding1,non_coding3,end]).
values(trans(non_coding3),[rho_term,tailA,non_coding1,end]).
values(trans(rho_term),[non_coding1,end]).
values(trans(tailA),[stemA,end]).
```

```
values(trans(stemA),[loop,end]).
values(trans(loop),[stemB]).
values(trans(stemB),[tailT]).
values(trans(tailT),[non_coding1,end]).

values_x(period(non_coding1),[0-L]) :-
       non_coding_length(L).

values_x(period(non_coding3),[0-L]) :-
       non_coding_length(L).

values_x(period(inter_GP),[15-L]):-
      inter_GP_length(L).

values_x(period(inter_PS),[15-L]):-
      inter_PS_length(L).

values_x(period(direct_coding),L):-
      direct_coding_length(L1),
      list_mod_3(L1,L).

values_x(period(rho_term),[10-L]):-
      rho_length(L).

values_x(period(tailA),[0-L]):-
      tailA_length(L).

values_x(period(tailT),[0-L]):-
      tailT_length(L).

values_x(period(loop),[3-L]):-
      loop_length(L).

values_x(period_frame1(non_coding2),L):-
      non_coding2_length(L1),
      list_mod_3(L1,L).

values_x(period_frame2(non_coding2),L):-
      non_coding2_length(L1),
      list_frame_3(L1,L).

values_x(period_frame3(non_coding2),L):-
      non_coding2_length(L1),
      list_frame_2(L1,L).
```

```
values(emit(_Q),['a','c','t','g']).
values(emit(_Q,_P),['a','c','t','g']).
values(whichstart,[['a','t','g'],['g','t','g'],['t','t','g']]).
values(whichstop, [['t','a','a'],['t','g','a'],['t','a','g']]).
values(whichcodon,[['a','a','a'],['a','c','a'],['t','t','t'],['t','c','a'],
                   ['a','a','t'],['a','c','t'],['t','t','a'],['t','c','t'],
                   ['a','a','c'],['a','c','c'],['t','t','g'],['t','c','c'],
                   ['a','a','g'],['a','c','g'],['t','t','c'],['t','c','g'],
                   ['a','t','a'],['a','g','a'],
                   ['a','t','t'],['a','g','t'],['t','a','t'],['t','g','t'],
                   ['a','t','c'],['a','g','c'],['t','a','c'],['t','g','c'],
                   ['a','t','g'],['a','g','g'],           ['t','g','g'],
                   ['c','c','c'],['c','t','a'],['g','g','g'],['g','t','a'],
                   ['c','c','a'],['c','t','c'],['g','g','a'],['g','t','c'],
                   ['c','c','t'],['c','t','g'],['g','g','t'],['g','t','t'],
                   ['c','c','g'],['c','t','t'],['g','g','c'],['g','t','g'],
                   ['c','a','c'],['c','g','a'],['g','a','g'],['g','c','a'],
                   ['c','a','t'],['c','g','t'],['g','a','a'],['g','c','t'],
                   ['c','a','a'],['c','g','c'],['g','a','t'],['g','c','g'],
                  ['c','a','g'],['c','g','g'],['g','a','c'],['g','c','c']]).

      :-
       set_sw(trans(begin),[0.85,0.10,0.05]),
       set_sw(trans(non_coding1),[0.11,0.15,0.63,0.11]),
       set_sw(trans(gb),[1.0]),
       set_sw(trans(inter_GP),[1.0]),
       set_sw(trans(pb),[1.0]),
       set_sw(trans(inter_PS),[1.0]),
       set_sw(trans(sd),[1.0]),
       set_sw(trans(non_coding2),[0.85,0.15]),
       set_sw(trans(direct_start),[1.0]),
       set_sw(trans(direct_coding),[1.0]),
       set_sw(trans(direct_stop),[0.63,0.25,0.11,0.01]),
       set_sw(trans(non_coding3),[0.24,0.26,0.35,0.15]),
       set_sw(trans(rho_term),[0.85,0.15]),
       set_sw(trans(tailA),[0.89,0.11]),
       set_sw(trans(stemA),[0.89,0.11]),
       set_sw(trans(loop),[1.0]),
       set_sw(trans(stemB),[1.0]),
       set_sw(trans(tailT),[0.85,0.15]),
       set_sw(whichstart,[0.85,0.11,0.04]),
       set_sw(whichstop, [0.85,0.11,0.04]),
       set_sw(whichcodon,[ 0.016,0.0164,0.0164,0.0164,
```

```
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,      0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164,
                        0.0164,0.0164,0.0164,0.0164]),
    set_sw(emit('non_coding1'),[0.01,0.80,0.18,0.01]),
    set_sw(emit('non_coding2'),[0.01,0.80,0.18,0.01]),
    set_sw(emit('non_coding3'),[0.01,0.80,0.18,0.01]),
    set_sw(emit('inter_GP'),[0.01,0.80,0.18,0.01]),
    set_sw(emit('inter_PS'),[0.01,0.80,0.18,0.01]),
    set_sw(emit('direct_coding'),[0.54,0.11,0.23,0.12]),
    set_sw(emit('rho_term'),[0.07,0.80,0.12,0.01]),
    set_sw(emit('tailA'),[0.97,0.01,0.01,0.01]),
    set_sw(emit('tailT'),[0.01,0.01,0.97,0.01]),
    set_sw(emit('loop'),[0.63,0.01,0.35,0.01]),
    set_sw(emit('direct_start',1),[0.8,0.01,0.1,0.09]),
    set_sw(emit('direct_start',2),[0.01,0.01,0.97,0.01]),
    set_sw(emit('direct_start',3),[0.01,0.01,0.01,0.97]),
    set_sw(emit('direct_stop',1),[0.01,0.01,0.97,0.01]),
    set_sw(emit('direct_stop',2),[0.86,0.01,0.01,0.12]),
    set_sw(emit('direct_stop',3),[0.36,0.01,0.01,0.62]),
    set_sw(emit('gb',1),[0.03,0.09,0.78,0.10]),
    set_sw(emit('gb',2),[0.10,0.03,0.82,0.05]),
    set_sw(emit('gb',3),[0.03,0.14,0.15,0.68]),
    set_sw(emit('gb',4),[0.58,0.12,0.20,0.10]),
    set_sw(emit('gb',5),[0.31,0.52,0.10,0.07]),
    set_sw(emit('gb',6),[0.54,0.05,0.24,0.17]),
    set_sw(emit('pb',1),[0.03,0.08,0.82,0.07]),
    set_sw(emit('pb',2),[0.89,0.03,0.07,0.01]),
    set_sw(emit('pb',3),[0.26,0.10,0.52,0.12]),
    set_sw(emit('pb',4),[0.59,0.12,0.14,0.15]),
    set_sw(emit('pb',5),[0.49,0.21,0.19,0.11]),
    set_sw(emit('pb',6),[0.03,0.05,0.89,0.03]),
    set_sw(emit('sd',1),[0.97,0.01,0.01,0.01]),
```

```
        set_sw(emit('sd',2),[0.01,0.01,0.01,0.97]),
        set_sw(emit('sd',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('sd',4),[0.97,0.01,0.01,0.01]),
        set_sw(emit('sd',5),[0.01,0.01,0.01,0.97]),
        set_sw(emit('sd',6),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',1),[0.01,0.01,0.97,0.01]),
        set_sw(emit('stemA',2),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',4),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',5),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemA',6),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',6),[0.01,0.01,0.97,0.01]),
        set_sw(emit('stemB',5),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',4),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',3),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',2),[0.01,0.01,0.01,0.97]),
        set_sw(emit('stemB',1),[0.01,0.01,0.01,0.97]).

hmmGF(Seq,--Ann) :- hmmGF(Seq,begin,0,--1,--Ann).

hmm([], end, R,--N, --[s(N,end)]).

hmmGF(Seq,Q,R,--N,--Ann):-
        msw(trans(Q),Q1),
        (    Q1 = 'end' ->
             Seq=[],
             hmm([], end,R, --N, --Ann)
        ;
             Q1 = 'non_coding2' ->
             ( R = 0 ->
               msw(period_frame1(Q1),Length),
               hmmGF3(Seq,Q1,R,Length,--N,--Ann)
             ;
               R = 1 ->
               msw(period_frame2(Q1),Length),
               hmmGF3(Seq,Q1,R,Length,--N,--Ann)
             ;
               msw(period_frame3(Q1),Length),
               hmmGF3(Seq,Q1,R,Length,--N,--Ann)
             )
        ;
             Q1 = 'direct_start' ->
             hmmGF4(Seq,Q1,R,--N,--Ann)
        ;
```

```
           Q1 = 'direct_stop'  ->
           hmmGF5(Seq,Q1,R,--N,--Ann)
       ;
          (Q1 = 'gb';
           Q1 = 'pb';
           Q1 = 'sd';
           Q1 = 'stemA';
           Q1 = 'stemB')  ->
           hmmGF2(Seq,Q1,R,--N,--Ann)
       ;
           Q1 = 'direct_coding' ->
           msw(period(Q1),Length),
           hmmGF1(Seq,Q1,R,Length,--N,--Ann)
       ;
           msw(period(Q1),Length),
           hmmGF3(Seq, Q1,R,Length,--N, --Ann)
       ).

hmmGF1(Seq, Q, R,Length,--N, --Ann):-
       !,
       msw(whichcodon,S),
       S = [S1,S2,S3],
       Seq = [S1,S2,S3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       Length1 is Length-3,
       (    Length1 = 0 ->
           hmmGF(Ls,Q,R,--N3,--Ann)
       ;
           hmmGF1_rec(Ls, Q, R,Length1,--N3, --Ann)
       ).

hmmGF1_rec(Seq, Q, R,Length,--N, --Ann):-
       !,
       msw(whichcodon,S),
       S = [S1,S2,S3],
       Seq = [S1,S2,S3|Ls],
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       Length1 is Length-3,
      (    Length1 = 0 ->
           hmmGF(Ls,Q,R,--N3,--Ann)
```

```
      ;
          hmmGF1_rec(Ls, Q, R,Length1,--N3, --Ann)
      ).

hmmGF2([L1,L2,L3,L4,L5,L6|Ls],Q,R,--N,--[s(N,Q),e(N,L1,L2,L3,L4,L5,L6)|Ann])
:-
       msw(emit(Q,1),L1),
       msw(emit(Q,2),L2),
       msw(emit(Q,3),L3),
       msw(emit(Q,4),L4),
       msw(emit(Q,5),L5),
       msw(emit(Q,6),L6),
       -- N1 is N+1,
       -- N2 is N+2,
       -- N3 is N+3,
       -- N4 is N+4,
       -- N5 is N+5,
       -- N6 is N+6,
       hmmGF(Ls,Q,R,--N6,--Ann).

hmmGF3([L|Ls], Q, R,Length,--N, --Ann):-
       !,
       msw(emit(Q), L),
       -- N1 is N+1,
       Length1 is Length-1,
       R1 is (R+1) mod 3,
       (    Length1 = 0 ->
            hmmGF(Ls,Q,R1,--N1,--Ann)
       ;
            hmmGF3_rec(Ls, Q,R1,Length1,--N1, --Ann)
       ).

hmmGF3_rec([L|Ls], Q, R, Length,--N, --Ann):-
       !,
       msw(emit(Q), L),
       -- N1 is N+1,
       Length1 is Length-1,
       R1 is (R+1) mod 3,
      (    Length1 = 0 ->
           hmmGF(Ls,Q,R1,--N1,--Ann)
      ;
           hmmGF3_rec(Ls, Q, R1,Length1,--N1, --Ann)
      ).
```

```
% Constraint on start codon 'atg','gtg' and 'ttg'
hmmGF4(Seq, Q, R, --N, --Ann) :-
      msw(whichstart,S),
      S = [S1,S2,S3],
      Seq = [S1,S2,S3|Ls],
      -- N1 is N+1,
      -- N2 is N+2,
      -- N3 is N+3,
      Ann = [s(N,Q),e(N,S1,S2,S3)|Rest],
      hmmGF(Ls,Q,R,--N3,--Rest).

% Constraint on stop codon 'taa','tag' and 'tga'
hmmGF5(Seq, Q, R, --N, --Ann) :-
      msw(whichstop,S),
      S = [S1,S2,S3],
      Seq = [S1,S2,S3|Ls],
      -- N1 is N+1,
      -- N2 is N+2,
      -- N3 is N+3,
      Ann = [s(N,Q),e(N,S1,S2,S3)|Rest],
      hmmGF(Ls,Q,R,--N3,--Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computation of all the numbers [3,6,9,12...] < Number divided by 3 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_mod_3(Number,[]) :-
      Number < 3.

list_mod_3(Number,[3|Rest]) :-
      Number1 is Number-6,
      list_mod_3_rec(Number1,6,Rest).

% Recursive call
list_mod_3_rec(Number,Elt,[Elt]) :-
      Number < 3.

list_mod_3_rec(Number,Elt,[Elt|Rest]) :-
      Number1 is Number-3,
      Elt1 is Elt+3,
      list_mod_3_rec(Number1,Elt1,Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Computation of all the numbers [1,4,7,10...] that makes two genes  %%
```

```
%%   in different reading frames.                                      %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_frame_2(Number,[]) :-
       Number < 3.

list_frame_2(Number,[1|Rest]) :-
       Number1 is Number-6,
       list_frame_2_rec(Number1,4,Rest).

% Recursive call
list_frame_2_rec(Number,Elt,[Elt]) :-
       Number < 3.

list_frame_2_rec(Number,Elt,[Elt|Rest]) :-
       Number1 is Number-3,
       Elt1 is Elt+3,
       list_frame_2_rec(Number1,Elt1,Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Computation of all the numbers [2,5,8,11...] that makes two genes  %%
%%   in different reading frames.                                     %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_frame_3(Number,[]) :-
       Number < 3.

list_frame_3(Number,[2|Rest]) :-
       Number1 is Number-6,
       list_frame_3_rec(Number1,5,Rest).

% Recursive call
list_frame_3_rec(Number,Elt,[Elt]) :-
       Number < 3.

list_frame_3_rec(Number,Elt,[Elt|Rest]) :-
       Number1 is Number-3,
       Elt1 is Elt+3,
       list_frame_3_rec(Number1,Elt1,Rest).
```

```
%-------------
% Coordinated PRISM Model II
%-------------
% Written by Yuan Zhang and HongBo Liu, (c) 2008
%-------------
% Model Description:
%   - This model is used for detecting all possible genes
%     are in the reading frame 2 of DNA direct strand.
% The different part of the codes to Model I is showed as follows

            Q1 = 'non_coding2' ->
            ( R = 0 ->
              msw(period_frame3(Q1),Length),
              hmmGF3(Seq,Q1,R,Length,--N,--Ann)
            ;
              R = 1 ->
              msw(period_frame1(Q1),Length),
              hmmGF3(Seq,Q1,R,Length,--N,--Ann)
            ;
              msw(period_frame2(Q1),Length),
              hmmGF3(Seq,Q1,R,Length,--N,--Ann)
            )
%-------------
% Coordinated PRISM Model III
%-------------
% Written by Yuan Zhang and HongBo Liu, (c) 2008
%-------------
% Model Description:
%   - This model is used for detecting all possible genes
%     are in the reading frame 3 of DNA direct strand.
% The different part of the codes to Model I is showed as follows

            Q1 = 'non_coding2' ->
            ( R = 0 ->
              msw(period_frame2(Q1),Length),
              hmmGF3(Seq,Q1,R,Length,--N,--Ann)
            ;
              R = 1 ->
              msw(period_frame3(Q1),Length),
              hmmGF3(Seq,Q1,R,Length,--N,--Ann)
            ;
              msw(period_frame1(Q1),Length),
              hmmGF3(Seq,Q1,R,Length,--N,--Ann)
            )
```

Appendix 7 Testing data

Appendix 7.1 Annotated sequence

Bacillus subtilis subsp. subtilis str. 168, complete genome [NC_000964].

Data 1: Region 2592203...2592517 (315bp, contains yqfT gene**)**

```
       1 catacattaa gtctggaggt ggaatgaccg atgattttat ttcaaagaat tattttgcag
      61 cggcttaatc aggctactgc tgatgatttg ctgaagtact caaaacaata cggaatcagc
     121 ttaacaagat cgcaggccgt tgaagtagcg aacctgttat acgggaaaaa tgtaaacatc
     181 tttaatgaga gtgaacgaat gcggctgctg aaacaagtag agacgattac atcgaaagaa
     241 acagctcaaa cagttaatga actattcaaa caatttacaa gctaaaaaat gggggtgtta
     301 tcccccattt tttat
```

Data 2: REGION: 4537...4775 (239bp, contains yaaB gene**)**

```
       1 gttagtgaag tgaagaaatg aggtgagcaa ttgtatattc atttaggtga tgactttgtg
      61 gtttcaacac gagatattgt cggcattttt gactttaaag ccaacatgtc gcctattgtt
     121 gaagaatttc tgaaaaaaca gaaacacaag gtggtgcctt ccgtaaacgc acgcccaaat
     181 ctatcgtagt cacggttcag aatatatatt actctccctt atcttccagc acattaaaa
```

Data 3: REGION: 3176...3471 (282bp, contains yaaA gene**)**

```
       1 gaaagaggtc gatataatgg caaatccgat ttcaattgat acagagatga ttacactcgg
      61 acaattctta aaattagccg atgtgattca gtctggcggt atggcgaagt ggtttttaag
     121 cgagcatgaa gtgcttgtga acgatgagcc ggacaaccgc cggggcagaa agctgtatgt
     181 tggagatgtg gtagagattg aaggatttgg ttcatttcaa gtcgtcaatt aa
```

Appendix 7.2 Artificial sequence

Data 4: gene A and gene B overlap each other

```
 g,c,a,t,g,c,g,a,c,a,t,g,c,g,g,c,c,t,a,a,g,c,g,a,c,t,c,t,a,a,c
    startA        startB          stopA              stopB
```

Data 5: gene C, D and E overlap each other, gene C is in reading frame 1, gene D is in reading frame 2, gene E is in reading frame 3

```
a,t,g,c,a,t,g,c,g,g,c,c,t,a,a,g,c,c,c,g,a,t,g,c,c,a,g,c,c,t,a,a,c,g,t,a,a
startC  startD          stopC          startE            stopE     stopD
```

Appendix 7.3 Testing results

Appendix 7.3.1 Testing general PRISM model with annotation

The codes of this program are showed in Appendix 3.

1. Run the efficient Viterbi on the sequence of Data 1 as follows:

```
?- viterbiAnnot(hmmGF([Data 1],A),P).
viterbiAnnot( hmmGF([c,a,t,a,c,a,t,t,a,a,g,t,c,t,g,g,a,g,g,t,g,g,a,a,t,g,a,c
,c,g,a,t,g,a,t,t,t,t,a,t,t,t,c,a,a,a,g,a,a,t,t,a,t,t,t,t,g,c,a,g,c,g,g,c,t,t
,a,a,t,c,a,g,g,c,t,a,c,t,g,c,t,g,a,t,g,a,t,t,t,g,c,t,g,a,a,g,t,a,c,t,c,a,a,a
,a,c,a,a,t,a,c,g,g,a,a,t,c,a,g,c,t,t,a,a,c,a,a,g,a,t,c,g,c,a,g,g,c,c,g,t,t,g
,a,a,g,t,a,g,c,g,a,a,c,c,t,g,t,t,a,t,a,c,g,g,g,a,a,a,a,a,t,g,t,a,a,a,c,a,t,c
,t,t,t,a,a,t,g,a,g,a,g,t,g,a,a,c,g,a,a,t,g,c,g,g,c,t,g,c,t,g,a,a,a,c,a,a,g,t
,a,g,a,g,a,c,g,a,t,t,a,c,a,t,c,g,a,a,a,g,a,a,a,c,a,g,c,t,c,a,a,a,c,a,g,t,t,a
,a,t,g,a,a,c,t,a,t,t,c,a,a,a,c,a,a,t,t,t,a,c,a,a,g,c,t,a,a,a,a,a,a,t,g,g,g,g
,g,t,g,t,t,a,t,c,c,c,c,c,a,t,t,t,t,t,t,a,t],A),P).
```

The following is the result of running the above Viterbi:

```
A = [s(14,sd),e(14,t,g,g,a,g,g),s(24,direct_start),e(24,a,t,g),s(276,direct_stop
),e(276,t,a,c),s(290,stemA),e(290,t,g,g,g,g,g),s(302,stemB),e(302,c,c,c,c,c,a),s
(316,end)]
P = 0.0 ?
```

2. Run the efficient Viterbi on the sequence of Data 2 as follows:

```
?- viterbiAnnot(hmmGF([Data 2],A),P).
viterbiAnnot(hmmGF([g,t,t,a,g,t,g,a,a,g,t,g,a,a,g,a,a,a,t,g,a,g,g,t,g,a,g,c,
a,a,t,t,g,t,a,t,a,t,t,c,a,t,t,t,a,g,g,t,g,a,t,g,a,c,t,t,t,g,t,g,g,t,t,t,c,a,
a,c,a,c,g,a,g,a,t,a,t,t,g,t,c,g,g,c,a,t,t,t,t,t,g,a,c,t,t,t,a,a,a,g,c,c,a,a,
c,a,t,g,t,c,g,c,c,t,a,t,t,g,t,t,g,a,a,g,a,a,t,t,t,c,t,g,a,a,a,a,a,a,c,a,g,a,
a,a,c,a,c,a,a,g,g,t,g,g,t,g,c,c,t,t,c,c,g,t,a,a,a,c,g,c,a,c,g,c,c,c,a,a,a,t,
c,t,a,t,c,g,t,a,g,t,c,a,c,g,g,t,t,c,a,g,a,a,t,a,t,a,t,a,t,t,a,c,t,c,t,c,c,c,
t,t,a,t,c,t,t,c,c,a,g,c,a,c,a,t,t,a,a,a,a],A),P).
```

The following is the result of running the above Viterbi:

```
A = [s(3,sd),e(3,t,a,g,t,g,a),s(10,direct_start),e(10,g,t,g),s(235,direct_stop),
e(235,t,a,a),s(240,end)]
P = 0.0 ?
```

3. Run the efficient Viterbi on the sequence of Data 3 as follows:

```
?- viterbiAnnot(hmmGF([Data 3],A),P).

viterbiAnnot(hmmGF([g,a,a,a,g,a,g,g,t,c,g,a,t,a,t,a,a,t,g,g,c,a,a,a,t,c,c,g,
a,t,t,t,c,a,a,t,t,g,a,t,a,c,a,g,a,g,a,t,g,a,t,t,a,c,a,c,t,c,g,g,a,c,a,a,t,t,
c,t,t,a,a,a,a,t,t,a,g,c,c,g,a,t,g,t,g,a,t,t,c,a,g,t,c,t,g,g,c,g,g,t,a,t,g,g,
c,g,a,a,g,t,g,g,t,t,t,t,t,a,a,g,c,g,a,g,c,a,t,g,a,a,g,t,g,c,t,t,g,t,g,a,a,c,
```

```
g,a,t,g,a,g,c,c,g,g,a,c,a,a,c,c,g,c,c,g,g,g,g,c,a,g,a,a,a,g,c,t,g,t,a,t,g,t,
t,g,g,a,g,a,t,g,t,g,g,t,a,g,a,g,a,t,t,g,a,a,g,g,a,t,t,t,g,g,t,t,c,a,t,t,t,c,
a,a,g,t,c,g,t,c,a,a,t,t,a,a,a,g,c,g,g,g,t,g,a,c,a,c,t,g,a,t,t,g,t,a,t,a,t,c,
c,a,g,a,a,c,t,t,a,g,a,a,c,t,g,a,c,a,t,c,t,t,a,c,c,g],A),P).
```

The following is the result of running the above Viterbi:

```
A = [s(3,sd),e(3,a,a,g,a,g,g),s(12,direct_start),e(12,a,t,a),s(264,direct_stop),
e(264,t,a,g),s(283,end)]
P = 0.0 ?
```

Appendix 7.3.2 Testing modified general PRISM model

The codes of this program are showed in Appendix 5.

```
1. Run the efficient Viterbi on the sequence of Data 1 as follows:
?- viterbiAnnot(hmmGF([Data 1],A),P).
```

The following is the result of running the above Viterbi:

```
A = [s(14,sd),e(14,t,g,g,a,g,g),s(31,direct_start),e(31,a,t,g),s(283,direct_stop
),e(283,t,a,a),s(290,stemA),e(290,t,g,g,g,g,g),s(302,stemB),e(302,c,c,c,c,c,a),s
(316,end)]
P = 0.0
```

```
2. Run the efficient Viterbi on the sequence of Data 2 as follows:
?- viterbiAnnot(hmmGF([Data 2],A),P).
```

The following is the result of running the above Viterbi:

```
A = [s(18,sd),e(18,a,t,g,a,g,g),s(31,direct_start),e(31,t,t,g),s(187,direct_stop
),e(187,t,a,g),s(240,end)]
P = 0.0 ?
```

```
3. Run the efficient Viterbi on the sequence of Data 3 as follows:
?- viterbiAnnot(hmmGF([Data 3],A),P).
```

The following is the result of running the above Viterbi:

```
A = [s(3,sd),e(3,a,a,g,a,g,g),s(17,direct_start),e(17,a,t,g),s(230,direct_stop),
e(230,t,a,a),s(283,end)]
P = 0.0
```

Appendix 7.3.3 Testing two genes overlap PRISM model

The codes of this program are showed in Appendix 4.

```
Run the efficient Viterbi on the sequence of Data 4 as follows:
viterbiAnnot(hmmGF([g,c,a,t,g,c,g,a,c,a,t,g,c,g,g,c,c,t,a,a,g,c,g,a,c,t,c,t,
a,a,c],A),P).
```

The following is the result:

```
A = [s(3,start_codon_A),e(3,a,t,g),s(10,start_codon_B),e(10,a,t,g),s(18,stop_cod
on_A),e(18,t,a,a),s(28,stop_codon_B),e(28,t,a,a),s(32,end)]
P = 0.0 ?
```

Appendix 7.3.4 Testing coordinated PRISM models

The codes of this program are showed in Appendix 6.

Run the program named Coordinated PRISM model I, model II and model III on the sequence of Data 5 respectively:

`viterbiAnnot(hmmGF([a,t,g,c,a,t,g,c,g,g,c,c,t,a,a,g,c,c,c,g,a,t,g,c,c,a,g,c,c,t,a,a,c,g,t,a,a],A),P).`

The result of running the above Viterbi by model I is showed as follows:

```
A = [s(1,direct_start),e(1,a,t,g),s(13,direct_stop),e(13,t,a,a),s(38,end)]
P = 0.0 ?
```

The result of running the above Viterbi by model II is showed as follows:

```
A = [s(5,direct_start),e(5,a,t,g),s(35,direct_stop),e(35,t,a,a),s(38,end)]
P = 0.0 ?
```

The result of running the above Viterbi by model III is showed as follows:

```
A = [s(5,sd),e(5,a,t,g,c,g,g),s(21,direct_start),e(21,a,t,g),s(30,direct_stop),e
(30,t,a,a),s(38,end)]
P = 0.0 ?
```