UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science & Mathematics
School of Electronics and Computer Science

Evolving the Structure of Hidden Markov Models for Biological Sequence Analysis

by

Kyoung-Jae Won

A thesis submitted in partial fulfilment for the
degree of Doctor of Philosophy

December 2005

Supervisor: Dr. Adam Prügel-Bennett

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE
DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

A thesis submitted in partial fulfilment for the
degree of Doctor of Philosophy

Evolving the Structure of Hidden Markov Models for Biological Sequence Analysis

by Kyoung-Jae Won

Hidden Markov Models (HMMs) are widely used for biological sequence analysis because of their ability to incorporate biological information in their structure. An automatic method of optimising the structure of HMMs for biological sequence analysis is highly desirable. However, this raises two important issues: first, the new HMMs should be able to grow enough to represent biological phenomenon, and we need to reduce overfitting of the HMM so that it has good generalization performance on unseen sequences.

In this thesis, we explore the possibility of using a genetic algorithm (GA) for optimising the HMM structure. The Baum-Welch algorithm is hybridised within its evolutionary cycle. To prevent overfitting, a separate dataset is used for comparing the performance of the HMMs to that used for the Baum-Welch training.

The proposed GA for hidden Markov models (GA-HMM) allows HMMs with different number of states to evolve. The GA-HMM was capable of finding an HMM comparable to a hand-coded HMM designed for the same task, which has been published previously.

We also propose Block-HMMs where the topology of HMMs was assembled from biologically meaningful building blocks. New genetic operators are designed to evolve the HMM structure while preserving the blocks.

We applied the evolving HMM structure methods to modelling the promoter and coding region of a prokaryote and predicting the secondary structure of proteins. The Block-HMM method could generate HMM structures and find conserved promoter region and triplet codon model without any prior information on the sequences. When the Block-HMM is tested for the protein secondary structure prediction problem, it showed superior performance to other prediction methods using HMMs and was comparable to the best known techniques for this problem.

Contents

List of Figures

List of Tables

Nomenclature

Θ	Set of parameters of an HMM
$\mathbb{P}(x\|y)$	Probability of event x given y
π	path through states of an HMM
$\mathcal{Q}$	Set of states of an HMM
$\boldsymbol{q}$	$(q_1, q_2, \ldots, q_T)$, state sequence
$\boldsymbol{x}$	$(x_1, x_2, \ldots, x_T)$, observation sequence
a_{ij}	Transition probability from state i to state j
$e_i(a)$	Probability of emitting a symbol a in a state i
L_{tot}	likelihood of observed sequence given the HMM variables
$\mathcal{L}$	log likelihood

Acknowledgements

I would like to thank my supervisor, Adam Prügel-Bennett, for guiding my course over the years. He has encouraged me to explore new areas of bioinformatics and allowed me to develop broad range of technical and scientific experiences. Thanks to him I could raise my academic ability through the years at Southampton. I would like to express my thanks to Anders Krogh at Bioinformatics Centre in University of Copenhagen for being my external advisor. The valuable discussions with him enabled me to take steps forward to profound area of research on bioinformatics.

I would like to thank Moonsang Seo for reviewing my thesis and sharing his knowledge of biology. I wish to express my gratitude to Lise Petersen and Thomas Hamelryck at Bioinformatics Centre in University of Copenhagen. They filled part of my thesis by kindly sharing their knowledge and valuable data with me. Also, I would like to thank all my friends and colleagues at Southampton and Copenhagen.

I would like to thank professor Hong-Tae Jeon, and my old colleagues in Korea and Vietnam for their continuous support. Finally, I would like to thank my parents for their endless love and support and for giving me the motivation to finish this thesis.

To my parents for letting me pursue my dream and being my support for so long

Chapter 1

Introduction

During the last two decades, owing to the development of sequencing technology, the amount of biological sequence data has increased explosively. To deal with this enormous amount of data as well as the complex nature of biological phenomena, computer-based research has become essential in biological sequence analysis. This computer aided research on biology or bioinformatics has become a crucial field of science that enables researchers to speed up their research and discover unknown facts that cannot be revealed without it. In sequence analysis the computer aided approaches are used in various areas such as interpreting the function of the biological sequences and tracing evolutionary information between different organisms.

Machine learning is a field of research where computational algorithms learn from a set of data. It uses artificial intelligence techniques to learn complex and real-world data. Because of their ability to learn complex data which cannot be defined well, machine learning techniques have been replacing the classical statistical approaches in modelling the biological sequences. Numerous attempts have been made to develop more accurate analysis tools for biological sequences. Among those attempts neural networks are most widely used. Neural networks are machine learning algorithms used for pattern recognition and signal processing. With their ability to train sequences in the networks, they were applied for the prediction of intron splice sites of human pre-mRNA [7]. GRAIL [8] and GRAIL II [9] were developed using neural networks to recognise coding region of the DNA sequence. For the modelling of the promoters and splice junctions in Human DNA, Time-delay neural network (TDNN) architecture was suggested [10]. Neural networks for the TATA-box and the initiators are trained separately and combined to a TDNN. In GeneParser [11] intron-exon and splice site indicators are weighted by neural networks. Most of the protein secondary structure prediction methods are built using neural networks. PHD [12], PSIPRED [13], SSpro [14], SSpro8 [15] and YASPIN [16] are neural networks based protein secondary structure predictors.

Hidden Markov Models (HMMs) have been widely used in biological sequence analysis. HMMs are stochastic models capable of statistical learning and classification. Although these models were originally applied to speech recognition [17], they have proved highly successful for recognizing biological sequences [18]. They have been used efficiently in a number of tasks ranging from amino acid profile searching [19] to multiple sequence alignments [20], to transmembrane helix architecture [21]. Their success owes much to their ability to encode biological information in their structure while allowing many unknown quantities to be learned through the optimisation of their transition and emission probabilities. Also, an HMM can be effectively constructed by combining several small HMM modules that are modelled and trained separately. In this way, an HMM can grow by concatenating biologically modelled modules.

Even though hand designed HMMs were successful in modelling biological sequences, the number of applications is limited. One of the reasons lies in the difficulties in modelling complex nature of biology with an HMM structure. Also, there are some inherent weaknesses in the HMMs. Firstly, it can suffer from the overfitting problem. The performance of an HMM becomes worse once it is trained too much for the given sequences. Secondly, the structure model has to be selected carefully. Because HMMs have a large number of unstructured parameters, there can exist several architectures which can represent the same biological sequences. Complex HMM architectures are apt to suffer overfitting. On the other hand, too simplified model usually deteriorate its performance. Even though there are elegant parameter learning algorithms in HMMs such as Baum-Welch algorithm, structure learning methods still remains unexploited. Automatic optimisation of the structure of HMMs that determines the size of an HMM would potentially be highly beneficial. One of the candidates that may enable automatic optimisation is to use Genetic Algorithms (GAs).

In this thesis the use of Genetic Algorithms for optimising the HMM structure is investigated. A Genetic Algorithm is a robust general purpose optimisation technique which evolves a population of solutions [22]. GAs have been widely used to optimise architectures for Neural Networks [23]. One of the advantages of using GAs for the HMM structure problem is to utilise the flexibility provided by Genetic Algorithms (GAs) to gain the advantage of automatic structure discovery while retaining some of the benefits of a hand designed architecture. That is, by choosing the representation and genetic operators, we attempt to bias the search towards biologically plausible HMM architectures. In addition, we can incorporate the Baum-Welch algorithm which is traditionally used to optimise the emission and transition probabilities as part of the GA. The optimisation of HMM architectures is a discrete optimisation problem which is easy to implement in a GA. We can simultaneously optimise the continuous probabilities by hybridising the GA with Baum-Welch. Furthermore, GAs allow us to tailor the search operators so as to bias the search towards biologically plausible structures and enable us to combine small HMM modules with crossover operators.

In the previous literature, GAs have been used to train the structure of an HMM. Yada *et al.* [24] used a GA to find a TATA box model. Thomsen [5] designed similar genetic strategy to evolve an HMM for the secondary structure prediction problem. They included a term in their fitness function to penalise over-complex models. However, their results depended critically on the penalisation parameter. To evolve HMM structures while penalising over complex model, we split the training set into two. One part of the training set is used for training the HMMs using Baum-Welch, the other part of the data set is used to evaluate the HMM's fitness. We also propose a selective Baum-Welch scheme, where only part of the individuals of the population are Baum-Welch trained. Those two methods are used in GA-HMM to prevent overfitting phenomena.

The Block-HMM was designed to restrict the HMM structure evolution to biologically meaningful blocks. The topology of Block-HMM was assembled from the blocks motivated by applications of HMMs in biological sequence analysis. New GA operations were designed not to break the properties of the block. The Block-HMM can be thought of a genetic model which uses the property of modularity of HMMs by crossing over blocks instead of states. We used the Block-HMM to find a structural model for the promoter region and coding region of *Campylobacter jejuni*. On the discriminative test the HMM structure found by the Block-HMM showed better sensitivity than the hand-designed HMM.

To prove the usefulness of the Block-HMM we applied it to the protein secondary structure problem. The proposed Block-HMM method produced a better HMM structure than that any other automatic way of HMM structure learning algorithm produced so far on protein secondary structure prediction problem. It was also superior to elaborately hand designed HMM architecture.

The work carried out in this thesis is published in four journal papers and presented in two conferences:

- Won, K.-J., Hamelryck, T, Prügel-Bennett, A. and Krogh, A. (2005) HMM Structure Learning using Genetic Algorithms: Prediction of Protein Secondary Structure, *Bioinformatics*, submitted.
- Won, K.-J., Hamelryck, T, Prügel-Bennett, A. and Krogh, A. (2005) A Protein Secondary Structure Prediction using evolved HMM, *Young Bioinformaticians' forum*, Selected to present.
- Won, K.-J., Prügel-Bennett, A. and Krogh, A. (2004) Evolving the Structure of Hidden Markov Models. *IEEE Transactions on Evolutionary Computation*, Accepted.
- Won, K.-J.,Hamelryck, T, Prügel-Bennett, A. and Krogh, A. (2005) Evolving Hidden Markov Models for Protein Secondary Structure Prediction. *Proceedings of*

the 2005 IEEE Congress on Evolutionary Computation, pp. 33-40.

- Won, K.-J., Prügel-Bennett, A. and Krogh, A. (2004) The Block Hidden Markov Model for Biological Sequence Analysis. *Lecture Notes in Computer Science*, vol. 3213, pp. 64-70.

- Won, K.-J., Prügel-Bennett, A. and Krogh, A. (2004) Training HMM Structure with Genetic Algorithm for Biological Sequence Analysis. *Bioinformatics*, vol. 20, no. 18, pp. 3613-3627.

I am responsible for all implementation on GA-HMMs and Block-HMMs. Advice on biology was supplied by Professor Anders Krogh, Dr. Lise Petersen, and Dr. Thomas Hamelryck at Bioinformatics Centre in University of Copenhagen. The ideas described in this thesis arose in discussion with Anders Krogh, my supervisor and myself. The original contribution to the field is described in chapter 5-8.

This work is expected to be useful for bioinformaticians who would like to design an HMM for biological sequences analysis. The result found by using Block-HMM can be used as a preliminary HMM for further improvement. The algorithm using blocks may be able to give some hints on designing other automatic way of HMM structure learning for other computer science areas such as pattern or speech recognition.

The remainder of this thesis begins with basic biological backgrounds in chapter 2. Genetic Algorithms are discussed in chapter 3. Chapter 4 provides overviews on HMMs and describes the training and decoding methods. Chapter 5 describes how we hybridized the HMM and the GA to design GA-HMM. In chapter 6 the Block-HMM is introduced. New genetic operations are devised to deal with blocks. In this chapter the Block-HMM used to find an HMM structure for the DNA sequences is discussed. In chapter 7 protein secondary structure prediction method using the Block-HMM is described. In addition, the results of the Block-HMM are compared with other prediction methods. Chapter 8 investigates how the parameters work inside the Block-HMM.

Chapter 2

Computational Approaches to Biological Sequence Analysis

2.1 Background on Molecular Biology

2.1.1 DNA

DNA consists of two long strands that wrap around each other to form the *double helix*. Each strand is built from a small set of molecules called *nucleotides* or *bases*. The four nucleotides are adenine(A), guanine(G), thymine(T) and cytosine(C). The order of the nucleotides contains the information that builds an organism. The information is read in three processes called replication, transcription, and translation.

By a chemical bond adenine always pairs with thymine (an A-T pair) and cytosine with guanine (a C-G pair). Therefore, each strand in a DNA double helix becomes a chemical mirror image of the other. When a cell divides to form two new daughter cells *replication* process takes place. Each strand of the double helix are untwisted and used as a template to form a *complementary strand* (figure 2.1).

DNA also acts as a template for the synthesis of RNA in a process called *transcription*. During this process only specific parts of the genome are transcribed to produce RNA molecules. The RNA polymerase recognises the start point (*Transcription Start Site*:TSS) of a gene on the DNA and transcribes the mRNA until it reaches a termination signal. A variety of different termination signals are used by the genome. When it copies the DNA sequence,thymine(T) is replaced with uracil(U).

A ribosome reads the mRNA and performs translation into protein. The ribosome attaches to the mRNA and recognises the first AUG triplet codon (start codon). It reads the sequence, 3 bases at a time, and synthesises amino acids. This translation

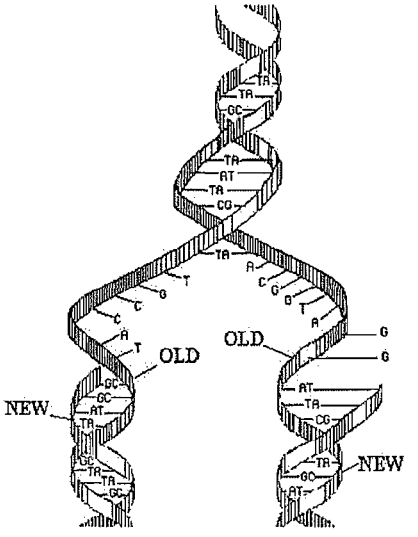


FIGURE 2.1: A simplified representation of a DNA molecule separating to form two new molecules. Two strands of DNA are obtained from one. Adopted from National Health Museum web page [1].

terminates when the ribosome reaches the stop codon (UGA, UAA, or UAG). Figure 2.2 illustrates the gene structure and the steps involved in synthesis of a protein.

Promoters are located upstream of the TSS. They have a TATAAT sequence, called the TATA box, as well as one or more promoter elements further upstream. The TATA box is found in eukaryote about 30 base pairs upstream from the site where transcription begins and about 10 base pairs upstream in prokaryote. In *E. coli*, for example, there are particular conserved sequences of TATAAT located 10 bases upstream (-10 region) of the TSS and TTGACA located 35 upstream (-35 region) of the TSS. However, the conserved region does not always contain the identical sequence and a variety of types of promoters are possible. In some cases the TATA box is not even explicitly shown, which makes it difficult to find the exact site where the transcription starts.

A long DNA sequence that is not interrupted by a stop codon and encodes part or all of a protein is called an *Open Reading Frame* (ORF). In prokaryotic genes most of the ORF is composed of coding sequences, whereas, in eukaryotic genes the ORF is composed of exon-intron structures. The exons have information used in producing a protein. In general, introns are much longer than exons. The functions of introns are still obscure. To produce mRNA, exons in DNA sequence are cut and spliced. The splicing of mRNA must be done with great precision. A small missplicing causes a frame shift and produces new codons specifying a totally different sequence of amino acids. The splice sites are largely defined by sequences within the intron. The intron begins with GU and ends

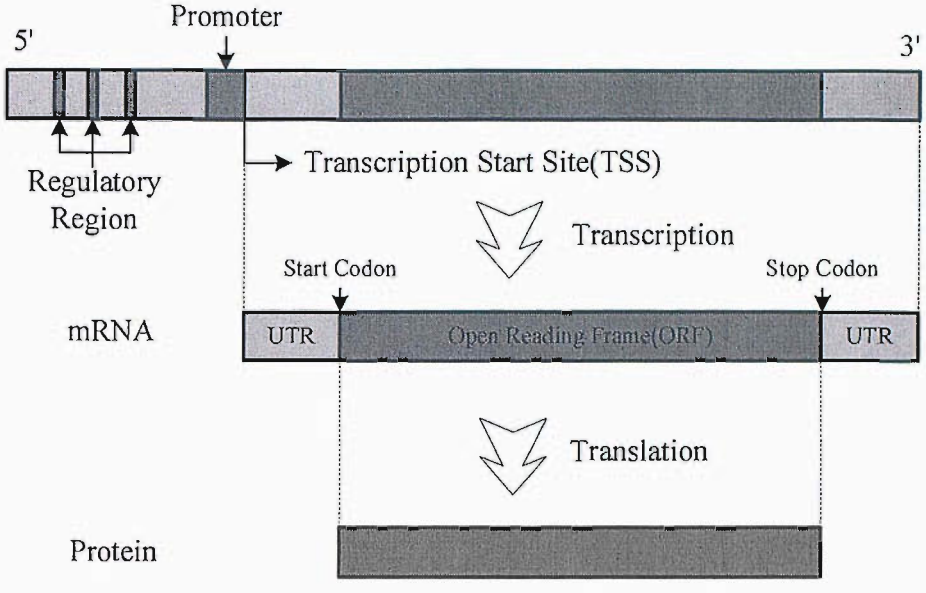


FIGURE 2.2: Gene structure and the steps involved in synthesis of a protein. Through transcription the DNA information is duplicated in mRNA. A protein is produced through translation. During translation ribosome reads the sequence, 3 bases at a time, and synthesises the amino acid. Adopted from [2] and modified.

with AG. Figure 2.3 shows the gene structure of eukaryotic genes and figure 2.4 shows the splicing procedure to produce mRNA. Genes of higher eukaryotes may span up to millions of base pairs. The human dystrophin gene, for example, is 2.2 million base pairs long. The relationships between a gene DNA sequence, its primary transcript, the various forms of mRNA, and the final protein sequence is very complex.

2.1.2 Protein Structure

During translation 20 amino acids are created from codons. Because there are $64(4^3)$ possible codons, some codons are redundant. Figure 2.5 shows how RNA is translated into protein.

Each amino acid has a similar, yet unique structure. Amino acids are classified by the chemical nature of their side chains. The chemical nature of the side chains plays the key roles in forming the protein structure. The amino acids are linked by dehydration synthesis forming peptide bonds in the protein structure. Some amino acids are conserved through evolution at specific locations in a protein sequence because they are essential for stability, formation of specific binding sites, or catalyst reaction.

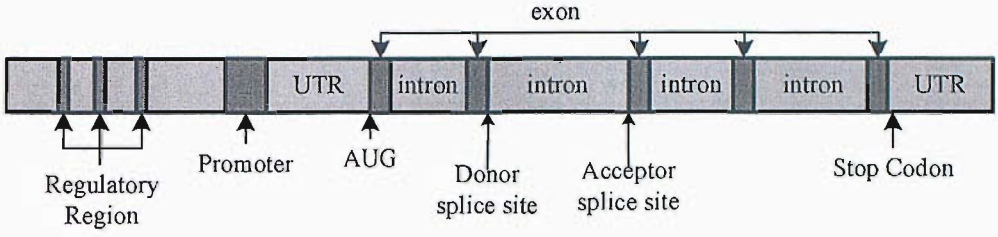


FIGURE 2.3: Gene structure of eukaryotic genes. The ORF of eukaryotic genes is composed of exon-intron structure. In general, exons which are known to contain the genetic information are shorter than introns. Adopted from [2] and modified.

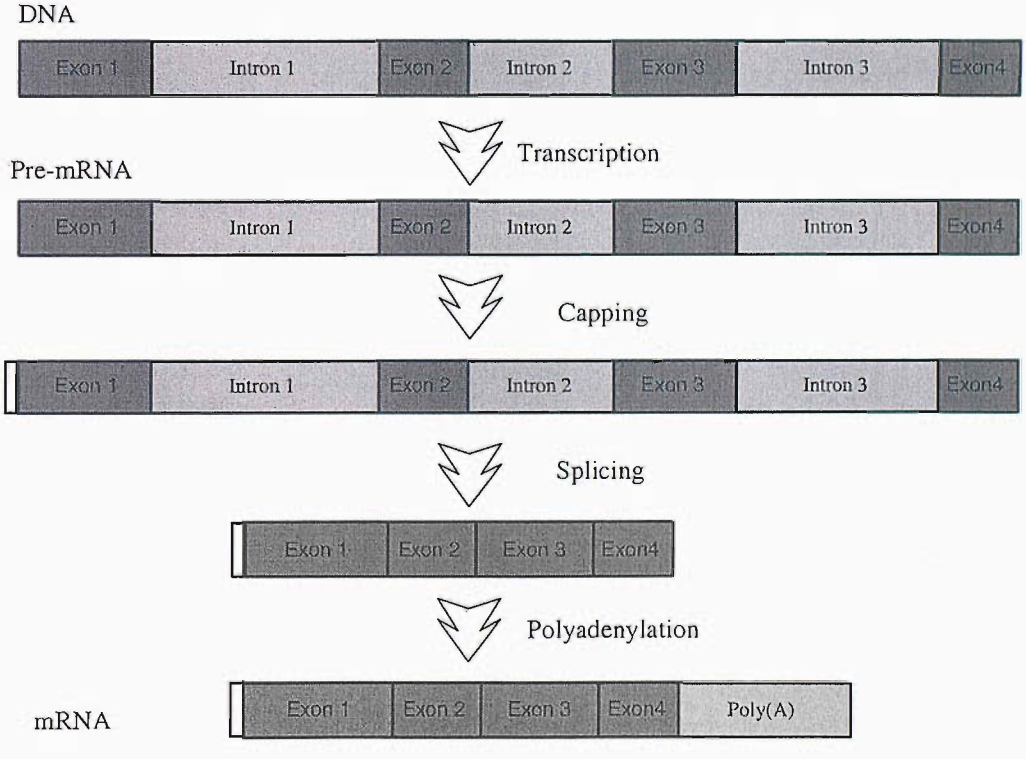


FIGURE 2.4: Transcription and splicing of eukaryotic genes. The introns in DNA are removed and the remaining exons compose the mRNA. The produced mRNA has a cap on the head and the poly(A) on the tail. Adopted from [2] and modified.

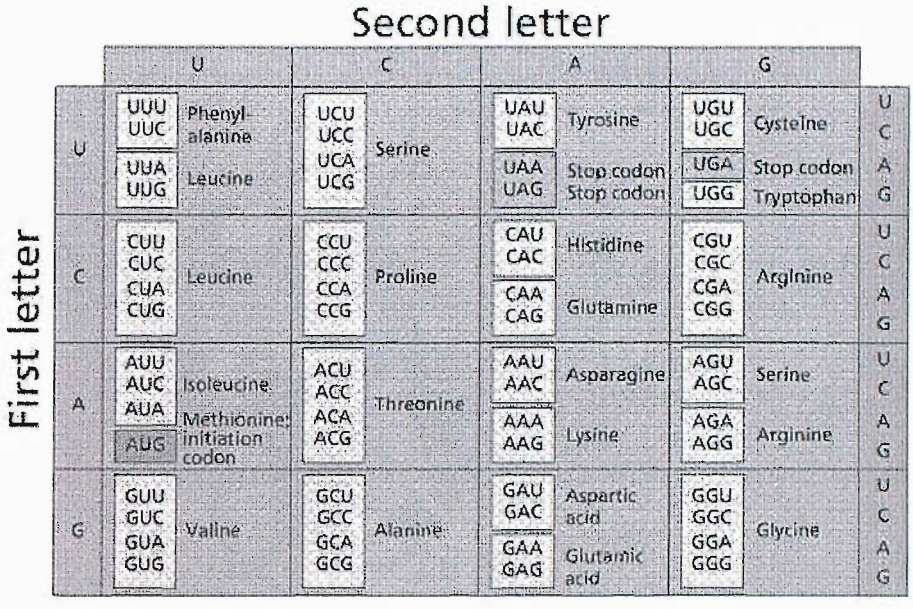


FIGURE 2.5: Genetic Codes. From the possible 64 (4^3) codons only 20 amino acids are created. Adapted from [3].

Proteins are polymers of amino acids. The primary structure of protein is the sequence of amino acids, which forms a chain connected by peptide bonds. Nearby amino acids associate with one another to form regions of secondary structure consisting of α-helices, β-strands and coils.

- α-Helices: These are rod shaped. The peptide is coiled around an imaginary cylinder and stabilised by hydrogen bonds formed between components of the peptide bonds.
- β-strands: The amino acids adopt the conformation of a narrow strips which form sheets like paper and the structure is stabilised by hydrogen bonds between amino acids in different polypeptide strands. They are usually found in the form of parallel or antiparallel strands.
- coil : Other parts of the structure that are not highly stable.

The elements of secondary structure pack together in a defined manner to generate a polypeptide's tertiary structure. Amino acids which are very distant in the primary structure might be close in the tertiary structure because of the folding of the chain. The quaternary structure is the arrangement of polypeptide subunits within complex proteins made up of two or more subunits.

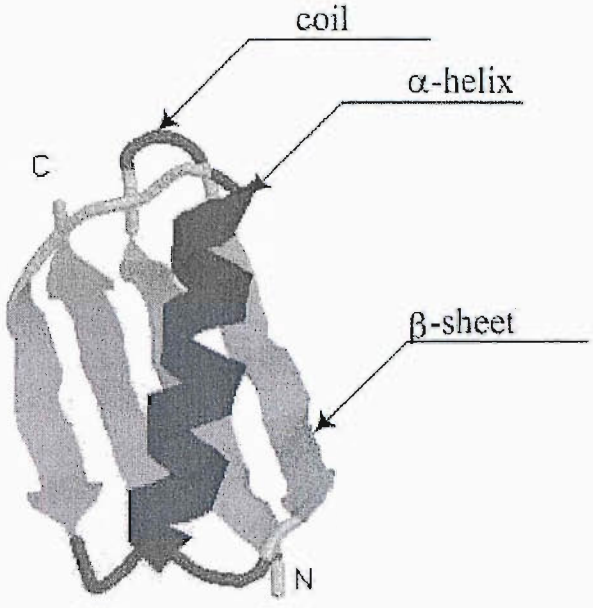


FIGURE 2.6: A protein is shown as Cartoons with Structure coloring. The α-helix is a spiral; the β-strands are depicted with arrows; coils are mostly connecting segments. Generated by using Chime [4].

2.2 Machine Learning Methods for Biological Sequence Analysis

Machine learning is a field of research where computational methods learn to answer complicated problems based on sets of provided data. It has been widely used for biological sequence analysis and supplies us with effective tools for many areas of bioinformatics such as gene finding and protein structure prediction.

2.2.1 Gene Finding

Gene finding is the area of computational biology that is involved in algorithmically identifying stretches of sequences that are actually functional (code for proteins or have regulatory functions) from non-coding or junk sequences. The gene finding strategies can be categorised as 1)site based methods, 2)contents based method and 3)comparative methods. 1) Site based methods use the position property of the sequences such as conserved sequences, donor and acceptor splice sites, transcription factor binding sites, poly(A) signals and start or stop codons. PROSITE expression was built on conserved sequences and allowable variations information [25]. Weight matrix was used in the place of consensuses [26, 25, 27, 28]. Later neural networks were applied to DNA signal recognition problems [7, 29, 30]. 2) Content based methods use the bulk properties of

sequences, such as CpG islands of the gene [31,32] and repetitive DNA sequence [33,34]. 3) Comparative methods compare the sequences from related organisms and use the similarities between them [35]. Currently, integrating all of the three methods together is popular for higher prediction accuracy. Those integrated gene-finders usually use dynamic programming. GRAIL II [9], GeneParser [11], FGENEH [36], GeneID [37] and GeneWise [38] were developed based on dynamic programming methods.

Today the most accurate gene-finding methods use machine learning techniques such as neural networks, decision trees and hidden Markov models to evaluate the information from all of the traditional approaches in order to make a prediction. In particular, hidden Markov models have been successfully applied in biological sequence analysis as they can be used to accommodate biological knowledge of their structure and enable statistical modelling. Genie [39], GENSCAN [40], FGENEH [41] and HMMgene [42] were developed on the basis of hidden Markov models.

2.2.2 Protein Structure Prediction

Predicting protein secondary structure is an important step towards understanding protein structure and its function. This task has attracted considerable attention and consequently represents one of the most studied problems in bioinformatics. The protein secondary structure prediction methods predict secondary region from primary sequences.

2.2.2.1 Definition of Protein Secondary Structure

According to the DSSP [43] definition there are 8 types of structure in protein secondary structure: H (α-helix), G (3_{10}-helix), I (π-helix), E (extended strand), B (residue in isolated β-bridge), S (bend), T (hydrogen bonded turn) and C (others). Most of prediction methods used a reduction scheme whereby H and G are converted to H, E and B are converted to E, and all the other to C. Figure 2.7 shows a protein sequence and its secondary structure labels.

```
> group1hk9:A
Sequence  : SLQDPFLNALRRERVPVSIYLVNGIKLQGQ
Observed  : CCHHHHHHHHHHCCCCEEEEECCCCEEEEE
Predicted : CCCCHHHHHHHHCCCCCCCCCCCCCEECCE
```

FIGURE 2.7: A protein sequence with label and its prediction.

There are several ways to check the performance of the predictors. The widely used Q_{index} is the percentage of residues predicted correctly as helix (Q_H), strand (Q_E), coil

(Q_C) or for all three labels(Q_3). For a single conformational state Q_i is defined as

$$Q_i = \frac{number\ of\ residues\ correctly\ predicted\ in\ state\ i}{number\ of\ residues\ observed\ in\ state\ i} \times 100, \tag{2.1}$$

where i is either helix, strand or coil. For all three state

$$Q_3 = \frac{number\ of\ residues\ correctly\ predicted}{number\ of\ all\ residues} \times 100. \tag{2.2}$$

In this example Q_H is 80 ((8/10) × 100)% and Q_3 is 70 ((21/30) × 100)%.

Segment overlap measure (SOV) considers type and position of the secondary structure segments rather than a per-residue assignment of conformational state [44]. Let s_1 and s_2 denote an observed and predicted secondary structure segment. A segment is the set of adjacent residues with the same label. In figure 2.7 the first two observed segments are 'CC' and 'HHHHHHHHHH'. When (s_1, s_2) is a pair of overlapping segments, we can define $S(i)$ as the set of all the overlapping pairs of segments (s_1, s_2) where s_1 and s_2 have at least one residue in state i in common, and $S'(i)$ as the set of segments s_1 that do not produce any segment pair. The segment overlap quantity measure for a single conformation state i is

$$SOV(i) = 100 \times \frac{1}{N(i)} \sum_{S(i)} \left[\frac{minov(s_1, s_2) + \delta(s_1, s_2)}{maxov(s_1, s_2)} \times len(s_1) \right] \tag{2.3}$$

with the normalisation value $N(i)$ defined as

$$N(i) = \sum_{S(i)} len(s_1) + \sum_{S'(i)} len(s_1) \tag{2.4}$$

where $len(s_1)$ is the number of residues in the segments s_1. $minov(s_1, s_2)$ is the length of actual overlap of s_1 and s_2, $maxov(s_1, s_2)$ is the length of the total extent for which either of the segments s_1 or s_2 has a residue in state i, and $\delta(s_1, s_2)$ is defined as

$$\begin{aligned} \delta(s_1, s_2) = \min \{ & (maxov(s_1, s_2) - minov(s_1, s_2)), \\ & minov(s_1, s_2), int(len(s_1)/2), int(len(s_2)/2) \} . \end{aligned} \tag{2.5}$$

For the three state case of helix (H), strand (E), and coil (C),

$$SOV = 100 \times \frac{1}{N} \sum_{i \in H,E,C} \sum_{S(i)} \left[\frac{minov(s_1, s_2) + \delta(s_1, s_2)}{maxov(s_1, s_2)} \times len(s_1) \right] \tag{2.6}$$

with the normalisation value $N(i)$ defined as

$$N = \sum_{i \in H,E,C} N(i). \tag{2.7}$$

In the example shown in figure 2.7,

$$\begin{aligned} SOV(H) &= 100 \times \frac{1}{10+0} \times \left[\frac{8+2}{10} \times 10 \right] = 100\% \\ SOV(E) &= 100 \times \frac{1}{5+5+5} \times \left[\left(\frac{2+1}{5} + \frac{1+0}{5} \right) \times 5 \right] = 26.7\% \\ SOV(C) &= 100 \times \frac{1}{2+4+4} \times \left[\frac{2+1}{4} \times 2 + \frac{4+2}{13} \times 2 \times 4 \right] = 51.9\%. \end{aligned}$$

Then, the overall segment overlap is

$$SOV = 100 \times \frac{1}{10+15+10} \times \left[10 + 4 + \frac{3}{4} \times 2 + \frac{12}{13} \times 4 \right] = 54.8\%.$$

For more detail on SOV see *e.g.* [44].

2.2.2.2 Secondary Structure Predictors

Early prediction methods were developed based on stereochemical principles [45] and statistics [46,47]. Later, a whole family of related sequences was used instead of a single sequence for analysis [48]. By clustering the sequences in an aligned family the secondary structure prediction rate for the cAMP-dependent kinases could be boosted .

Since then the prediction rate has steadily risen due to both algorithmic development and proliferation of the available data. The first machine learning method to predict protein secondary structure was built using neural networks [49,50]. Qian and Sejnowski constructed a cascaded neural networks. A second network was cascaded to improve the performance. Rost and Sander used a number of machine learning techniques including early stopping, ensemble averages of different network, and weighting scheme to construct a predictor, called PHD [12]. In particular, they started feeding the multiple sequence in the form of *profile* instead of just feeding the base sequence to the neural networks. These profiles contain information of an amino acid frequency vector. By using profiles PHD can boost the prediction rate up to 71% in the cross-validation test. Psipred [13] has a similar structure with PHD. It receives the feeding input from the profile obtained by running PSI-BLAST [51]. A method to use multiple sequence information as an output level was also suggested [52]. SSpro [14], SSpro8 [15] was constructed using recurrent neural networks. YASPIN [16] cascaded hidden Markov model with neural networks to filter the output of the neural networks.

Support vector machines have also been used and show promising results [53]. Recently, the prediction accuracy has been improved by cascading a second layer of support vector machines [54, 55]. Currently machine learning methods typically improve their performance by combining several predictors and using evolutionary information.

Most of these methods use *sliding window* technique. The sliding window technique slides a fixed-sized window along a sequence. Figure 2.8 illustrates a sliding window with a window size of 11. When the window is centered on an amino acid, the secondary structure of the amino acid is predicted using evolutionary information and multiple sequence alignment. The evolutionary information of a sequence is retrieved by running programs such as PSI-BLAST [51]. This process continues until the end of the sequence is reached.

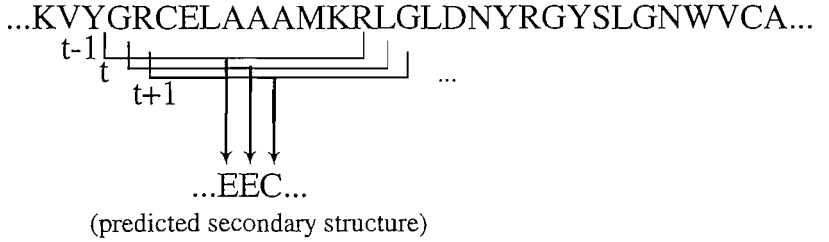


FIGURE 2.8: A sliding window with a window size of 11. A secondary structure of a residue located at the center of a window is predicted.

Hidden Markov models (HMMs) were also used to predict protein structures. HMMSTR [6] was the first successful protein secondary structure predictor based on HMMs. It was constructed by identifying recurring protein backbone motifs (called invariant/initiation sites or I-sites) and representing them as a Markov chain. Consequently, the topology of HMMSTR can be interpreted as a description of the protein backbone in terms of consecutive I-sites. Thomsen used genetic algorithms to evolve the HMM structure to find the protein secondary structure [5]. However, the performance was much worse than the hand-designed HMMSTR.

2.2.3 Similarity Searches on Sequence Database

When proteins or gene sequences are very similar, they are called *homologous sequences*. Homologous sequences are often derived from the same ancestral sequence, share the same structure, and have similar biological function even when they come from very different organisms. Top ranking secondary structure predictors use the homologous sequence information by running BLAST or PSI-BLAST to increase the prediction rate.

BLAST (Basic Local Alignment and Search Tool) provides a method for rapid searching nucleotide and protein databases [56]. The BLAST programs consist of a set of sequence

comparison algorithms to search sequence databases for optimal local alignments to a query. It computes scores by reference to an amino acid substitution matrix. Position specific iterative BLAST (PSI-BLAST) is an enhancement of the BLAST program [51]. It searches a database for local alignments using gapped BLAST and builds a multiple alignment and a profile. The profile is then used to search the protein database again.

Figure 2.9 shows the result of PSI-BLAST. We ran PSI-BLAST against the UniProt 90 protein sequence database [57]. The result comprises of two parts. The first part of the result lists all the names of homologous sequences with *score* and *E-value*. The score is a measure of the similarity of the query to the sequence in the list. It is calculated using a substitution matrix. In the substitution matrix the rate of possible change to each other residue is written. We used BLOSUM62 [58] matrix for the substitution matrix. The E-value is a measure of the reliability of the score. The lower the E value, the more significant the score. The other part of the PSI-BLAST result is composed of pairwise alignments of each homologous sequence. BLAST and PSI-BLAST are found at http://www.ncbi.nlm.nih.gov/BLAST.

The homologous sequences from the PSI-BLAST are aligned by a multiple sequence alignment method and a weight is assigned to every amino acid in protein sequences. If an alignment contains a majority of very similar proteins and a small number of slightly different sequences from the majority, the minority will have almost no influence on the prediction. Among several weighting schemes PSI-BLAST uses the position-based sequence weighting method[59]. The weight is given as

$$w_\kappa = \frac{\sum_{t=1}^{T} \frac{1}{r(\kappa,t)\cdot s(\kappa,t)}}{\sum_{\iota=1}^{\Gamma} \sum_{t=1}^{T} \frac{1}{r(\iota,t)\cdot s(\iota,t)}} \tag{2.8}$$

where Γ is the number of homologous sequences, $r(h,t)$ is the number of different residues at position t for homologous sequence h and $s(h,t)$ is the number of times the particular residue appears at a specific position.

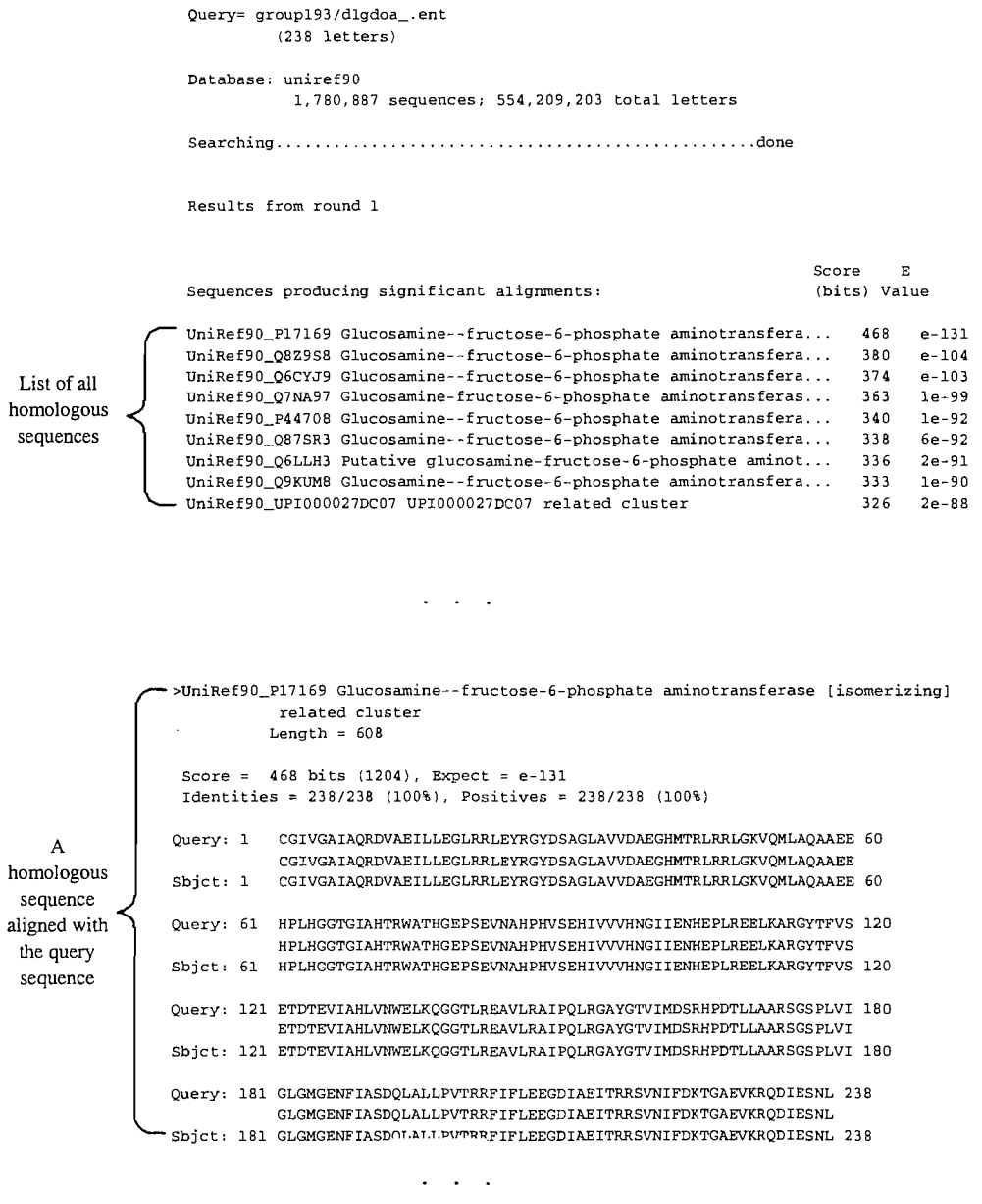


FIGURE 2.9: A result of PSI-BLAST with a query sequence. It contains the homologous sequences aligned with the query sequence.

Chapter 3

Genetic Algorithms(GAs)

3.1 Genetic Algorithms

Genetic algorithms (GAs) are stochastic algorithms inspired by biological genetic phenomena. They are frequently used to solve optimisation problems and have been applied to game playing and adaptive control problem. Unlike other stochastic searching algorithms such as hill climbing and simulated annealing [60], GAs maintain several solutions in parallel. They exploit searching spaces by combining two existing solutions to create new solutions or progressively changing a solution. The combining method enable GAs to make a large jump in the search space. Maintaining a collection of solutions in parallel, GAs are able to exploit the search space in several locations at the same time. This increases the possibility for GAs to find global solution.

In a GA a problem is expressed using a genetic representation, usually a population of binary strings. The initial population is usually composed of random strings. Associated with each string is a fitness value that numerically shows how well the string is suitable for the given problem. In each generation a proportion of the population undergo genetic operations and produce offspring to form the next generation. In the classical GA (figure 3.1) a genetic cycle comprises the fitness calculation, selection and genetic operations. The search space of a GA depends on those components.

Because of their stochastic nature, GAs often cause disruption when they explore the solution domain. However, the non-local movement operator, crossover, is still a fascinating way to expand the scope of the exploration in the solution domain.

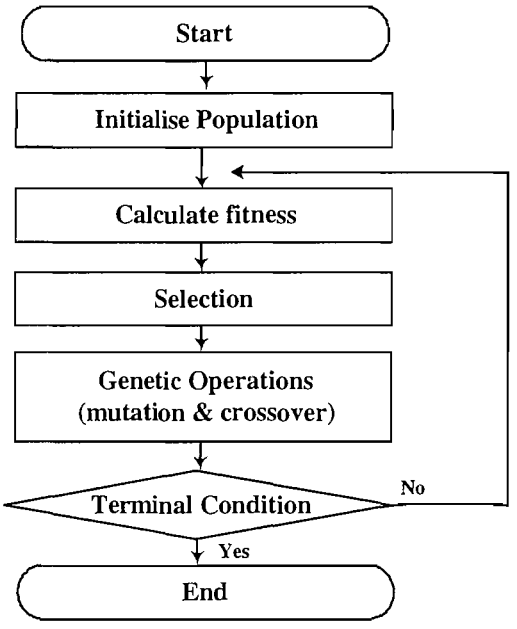


FIGURE 3.1: A procedure of a classic GA.

3.2 Genetic Operations

3.2.1 Crossover and Mutation

Mutation takes place on randomly selected individuals. It changes the value at one or more sites of the strings. In *graphical models* it can be designed to increase the number of nodes. If the mutation rate is too high then mutation will almost always be deleterious. Nevertheless, mutation is an important GA operator to explore the search space and to maintain diversity.

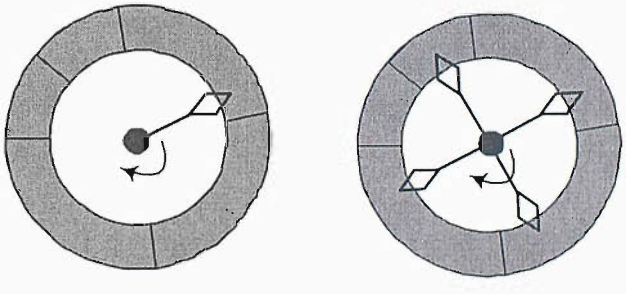
Under crossover a pair of individuals are mixed to produce two offspring. It enables a GA to reduce the problem of local minima. Even though it can cause disruption, well organized crossover methods can sometimes produce faster and better results than other learning algorithms. Classically, single-point crossover and multi-point crossover are used. A section or sections of the string cut by crossover are swapped with the section of another string. For the effective shuffling of two strings or alleles, uniform crossover [61] and bit-simulated crossover [62] have been developed. Under uniform crossover a child is created from two adults by randomly choosing alleles from one or the other parents. Under bit-simulated crossover a child is created by taking a variable from any of the members of the population.

GAs have succeeded in providing good solutions in some problems like the Travelling Salesman Problems [63] and graph-coloring problems [64]. This is partly because the GAs use the special crossover operator. Crossover can be very disruptive to an extent

that a child is unlikely to have a similar fitness to its parents. In graphical models it can create a child which has illegal paths for the given problem. Nevertheless, crossovers have a significant potential to find a solution with great efficiency. It remains a task to explore how to perform crossover in a useful way.

3.2.2 Selection

Through selection fitter members of the population are chosen. Usually a weight is assigned to an individual depending on its fitness. A new population is then created by sampling from the old population according to the weight. Different sampling strategies are possible. The two most commonly used sampling methods are roulette wheel sampling and stochastic universal sampling. In roulette wheel sampling, probabilities are assigned in a roulette wheel in proportion to the fitness values of the members. Then it draws a population of P (number of population) members like playing P games of roulette. Roulette wheel selection allows fit members of the population to be lost through bad luck. The loss of diversity may lead to a degrade of the performance of a GA. To overcome this drawback stochastic universal sampling [65] was introduced. In the stochastic universal sampling P equally spaced points are used once instead of drawing 1 point P times as the roulette wheel sampling methods does. This diminishes the fluctuations.



(a) Roulette Wheel Sampling (b) Stochastic Universal Sampling

FIGURE 3.2: Roulette wheel sampling and stochastic universal sampling.

Boltzmann selection has been studied [66] to perform a scaled selection. In Boltzmann selection each member of the population is chosen with a probability

$$P_\alpha = \frac{\omega_\alpha}{\sum_\beta \omega_\beta}, \tag{3.1}$$

with each member of the population is assigned a weight

$$\omega_\alpha = e^{\frac{s(E_\alpha - E_1)}{\sigma}}. \tag{3.2}$$

Here, s controls the strength of the selection and σ is the standard deviation in the fitness of the members of population. If s is zero all members of the population have equal probability to be chosen, and as s increases fitter members receive more chance to be chosen. One of the advantages of the Boltzmann selection is that the selective pressure is invariant under addition or multiplication of a constant to the fitness. That is, selection is unaffected by variations in the range of the distribution of fitness values. Typically, the range of the distribution decreases during the evolution of a GA. Without scaling the selection strength would be reduced as the fitness of members of the population becomes more similar and the population could suffer random genetic drift away from the best fitness.

3.3 Types of GA

In a generational GA, at each generation the whole population is replaced. Whereas, in a steady state GA a proportion of the members in the population are replaced. The selection strength and generic drift are stronger in a steady state GA than in a generational GA. Therefore the steady state GA can lose more chances to explore the landscape.

While in the classic GA selection and reproduction take place across the whole population, multiple island GAs divide the population into several islands. On each island the evolution process takes place independently except for occasional migration between the islands. It gives each island time to explore the search space and increases the chance to find global solution.

3.4 Parallel Genetic Algorithms

The evolutionary process is usually severely time consuming when they are implemented on a single processor because the fitness values of every individual is calculated in a single processor. The solution to this problem can be handled by using the parallel and distributed model. The parallel computing saves time by distributing the computational effort to a number of processors. And it also emulates the natural parallel evolution from the algorithmic point of view.

The parallel genetic algorithm uses a number of processors to run independent tasks.

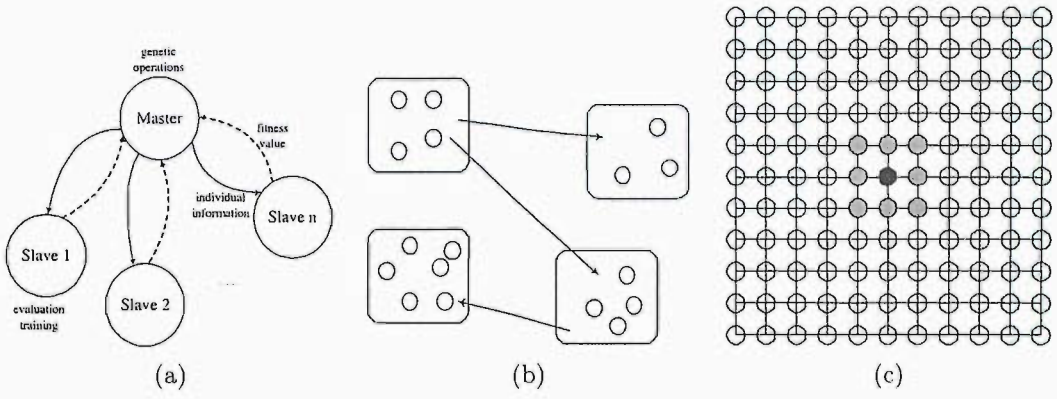


FIGURE 3.3: Distributed model for parallel genetic algorithms. (a) Master-slave model (b) island model (c) grid model

Ideally, P (size of the population) processors can be used to evolve a population of P members. These strategy can be implemented both in a global model and a population-based model.

In a global parallel evolutionary algorithm a master process manages the population and each slave locally trains an individual. The master collects the trained results and applies genetic operators to generate the next generation. In a population-based model subpopulations are generated and genetic operations are executed between neighboring individuals. The two most popular categories of the population-based models are *island* models and *grid* models. The *island* model [67] has separated subpopulations. Subpopulations exchange information by allowing migration of some individuals from other subpopulations. In the *grid* model [68] individuals are placed on a large one, two or three-dimensional grid. Reproductions take place locally within a small neighborhood.

Chapter 4

Hidden Markov Models(HMMs)

4.1 Hidden Markov Models

An HMM is a learning machine that assigns a probabilistic relation between an observed sequence of symbols and a sequence of hidden states $\boldsymbol{q} = (q_1, q_2, \ldots, q_T)$. A Markov transition structure links the hidden states. In HMMs, a sequence, $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$, consist of symbols x_t belonging to some alphabet, $\mathcal{S}$. In biological sequence analysis the alphabet might be, for example, the set of four possible nucleotides in DNA, 'A', 'C', 'G' and 'T' or the set of 20 amino acids that are the building blocks of proteins.

HMMs are based on two sets of conditional relations. In HMMs, x_t is independent of all other observations and states given q_t depends on the previous n states (the nth order Markov process)

$$\mathbb{P}\left(x_t | x_{t-1}, x_{t-2}, \ldots, x_1\right) = \mathbb{P}\left(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-n}\right). \tag{4.1}$$

We denote the set of parameters that define an HMM by Θ. Given a sequence $\boldsymbol{x}$, an HMM returns a 'probability' $\mathbb{P}\left(\boldsymbol{x}|\Theta\right)$, where

$$\sum_{\boldsymbol{x} \in \mathcal{S}^T} \mathbb{P}\left(\boldsymbol{x}|\Theta\right) = 1,$$

so it is a probability distribution over sequences of length T (we use $\mathcal{S}^T$ to denote the set of all sequences of length T). To understand the meaning of this probability, we can imagine some process of interest, $\mathcal{P}$, (e.g. molecular evolution) that generates a set of sequences of length T with probability $\mathbb{P}\left(x|\mathcal{P}\right)$. Our aim is to find an HMM, Θ, such that $\mathbb{P}\left(x|\Theta\right)$ is as close as possible to $\mathbb{P}\left(x|\mathcal{P}\right)$. Of course, we usually do not know $\mathcal{P}$. Rather we have some training examples consisting of a set of sequences. We can then use the maximum likelihood principle to estimate the HMM, which corresponds to

maximising $\mathbb{P}(\boldsymbol{x}|\Theta)$ with respect to Θ ($\mathbb{P}(\boldsymbol{x}|\Theta)$ is known as the *likelihood* of the event $\boldsymbol{x}$ given a model Θ). Given an observation sequence $\boldsymbol{x}$, we can treat this as a function of Θ.

The HMM is a probabilistic finite state machine which can be represented as a directed graph in which the nodes correspond to states and the edges correspond to possible transitions between states. A transition probability is associated with each edge with the constraint that the sum of the transition probabilities for edges exiting a node must sum to one. In addition, there is an emission probability table associated with each state which encodes the probability of each symbol $a \in \mathcal{S}$ being 'emitted' given that the machine is in that state. We define one state as the start state, which does not emit a symbol, but has transitions to the other states. To compute probabilities from our HMM we consider an 'event' to be a $path(\pi)$ through the graph where we emit a symbol every time we enter a state. The probability of the event is equal to the probability of the path times the probability of emitting that sequence of observed symbols given that we have taken that path through the finite state machine. The probability of a sequence is then found by summing over all paths that emit that sequence.

To formalise the HMM, we denote the set of states by $\mathcal{Q}$, the transition probabilities from state i to j by a_{ij}, and the probability of emitting a symbol a given that we are in a state i by $e_i(a)$. To calculate the probabilities of all the paths that render a sequence $\boldsymbol{x}$ are calculated. Let $\boldsymbol{q} = (q_1, q_2, \ldots, q_T)$ be a sequence of states, then the likelihood of a sequence $\boldsymbol{x}$ is given by

$$\mathbb{P}(\boldsymbol{x}|\Theta) = \sum_{\boldsymbol{q} \in \mathcal{Q}^T} \mathbb{P}(\boldsymbol{x}, \boldsymbol{q}|\Theta). \tag{4.2}$$

The probability of each path that renders a sequence $\boldsymbol{x}$ is then

$$\mathbb{P}(\boldsymbol{x}, \boldsymbol{q}|\Theta) = \prod_{t=1}^{T} a_{q_{t-1}q_t}\, e_{q_t}(x_t). \tag{4.3}$$

Here $q_0 = 0$ denotes the initial state.

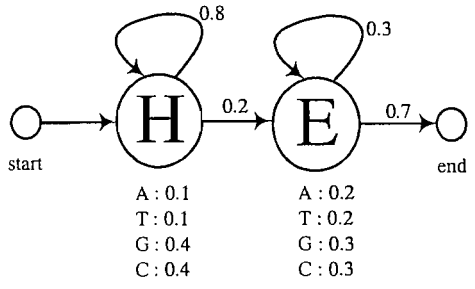
Often we are interested in how well a sequence fits the model. To do this we consider the log-odds of a sequence

$$\text{log-odds of a sequence } = \log\left(\frac{\mathbb{P}(\boldsymbol{x}|\Theta)}{|\mathcal{S}|^{-T}}\right) \tag{4.4}$$

where $|\mathcal{S}|$ denotes the cardinality of the set of symbols $\mathcal{S}$. The log-odds is positive if the sequence is more likely to occur than average. To use an HMM for classification we can set a threshold for the log-odds of a new sequence to belong to the same class as the training data.

Figure 4.1 shows a simple HMM with 2 states. For the given sequence 'ATGCAT' there

are 5 state paths. To obtain $\mathbb{P}(\boldsymbol{x}|\Theta)$ we must add the probabilities for all possible paths. The log-odds can be calculated by setting $|\mathcal{S}| = 4$.



Sequence : A T G C A T	probability
State path : H E E E E E	8.16 x 10^{-8}
H H E E E E	1.09 x 10^{-7}
H H H E E E	3.87 x 10^{-7}
H H H H E E	1.38 x 10^{-6}
H H H H H E	3.79 x 10^{-6}
log likelihood :	-5.42
log-odd :	-1.81

FIGURE 4.1: An example of an HMM with 2 states. There are 5 possible path for a given sequence 'ATGCAT'. The likelihood is calculated by summing possibilities of all possible paths. In this example log likelihood is calculated. Log-odds score is calculated by setting $|\mathcal{S}| = 4$.

4.2 Probability Calculation

Naively, the computation of the likelihood seems to grow exponentially with the length of the sequence. However, all the computations we need can be computed efficiently using dynamic programming techniques. For simplicity we will discuss the first-order model case. We can compute the likelihood using the *forward* algorithm. The forward variable $\alpha_t(i)$ is defined as

$$\alpha_t(i) = \mathbb{P}(x_1, \cdots, x_t, q_t = i|\Theta). \tag{4.5}$$

This variable calculates the joint probability of the partial observation sequence $x_1, \cdots, x_t$ and state i at time t, given an HMM Θ.

Starting from $\alpha_1(i) = a_{0i}e_i(x_1)$, we can find $\alpha_t(i)$ for all states $i \in \mathcal{Q}$ for successive times

$1 \leq t \leq T$ using the recursion

$$\alpha_t(i) = e_i(x_t) \sum_{j \in \mathcal{Q}} a_{ji}\, \alpha_{t-1}(j). \tag{4.6}$$

This follows from the Markovian nature of the model. When we have found $\alpha_T(i)$ we can compute the likelihood by marginalising out the final state

$$\mathbb{P}(\boldsymbol{x}|\Theta) = \sum_{i \in \mathcal{Q}} \alpha_T(i). \tag{4.7}$$

The forward algorithm considers all paths through the model and the probability can be viewed as a weighted average of probabilities given each path.

$$\mathbb{P}(\boldsymbol{x}|\Theta) = \sum_{\boldsymbol{q}} \mathbb{P}(\boldsymbol{x}|\boldsymbol{q}, \Theta)\, \mathbb{P}(\boldsymbol{q}|\Theta). \tag{4.8}$$

There exists an analogous backward algorithm that can also be used to compute the likelihood. We define the backward variable $\beta_t(i)$ to be the probability of matching the sequence $x_{t+1}, \cdots, x_T$ given that we are in state i at time t

$$\beta_t(i) = \mathbb{P}(x_{t+1}, \cdots, x_T | q_t = i, \Theta). \tag{4.9}$$

Again this can be obtained recursively using

$$\beta_t(i) = \sum_{j \in \mathcal{Q}} a_{ij} e_j(x_{t+1}) \beta_{t+1}(j) \tag{4.10}$$

with initial condition $\beta_T(i) = 1$ for all $i \in \mathcal{Q}$. The likelihood is given by

$$\mathbb{P}(\boldsymbol{x}|\Theta) = \sum_{i \in \mathcal{Q}} a_{0,i}\, e_i(x_1)\, \beta_1(i).$$

More importantly, the backward variable can be used in combination with the forward variable to compute important quantities needed for parameter estimation.

In calculating both forward and backward algorithm there are numerical underflow problems caused by the repeated multiplication of small probabilities. One of the solutions is to use logarithm and calculate *log-likelihood*. The log-likelihood is calculated by adding the small probabilities instead of multiplying them.

4.3 HMM Parameter Estimation

4.3.1 Baum-Welch Algorithm

Parameters of an HMM can be estimated so as to maximise the model likelihood $P(x|\Theta)$ on the given sequences (Maximum Likelihood (ML) criterion). This is achieved using the Baum-Welch algorithm [69] which is an example of an Expectation Maximisation (EM) algorithm. Because the state sequences are not directly observable, statistical information is used to adjust the parameters. The model parameters that maximise the likelihood value are calculated repeatedly. The update rule of the transition probabilities are

$$a_{ij} = \frac{n_{ij}}{\sum_{j' \in Q} n_{ij'}}. \tag{4.11}$$

where n_{ij} is the number of transitions from state i to state j summed over the sequence, that is,

$$n_{ij} = \sum_{t=1}^{T} n_{ij}(t). \tag{4.12}$$

The update rule of the emission probabilities can be obtained as

$$e_i(a) = \frac{n_i(a)}{\sum_{a' \in S} n_i(a')} \tag{4.13}$$

where $n_i(a)$ is the number of times the symbol a is emitted in state i. Equations (4.11) and (4.13) are self-consistent equations which are satisfied when the likelihood for the training data is locally maximum. We satisfy the equation by iteratively updating a_{ij} and $e_i(a)$ according to the observed values for n_{ij} and $n_i(a)$ which are computed using the forward and backward algorithm.

In the Baum-Welch algorithm unknown transition and emission frequencies are replaced by their expected values. The algorithm uses the probabilities of transitions and emissions to approximate the corresponding counters. The parameters are re-estimated using the forward and the backward variables. The parameters are updated according to

$$\begin{aligned} n_{ij}(t) &= \mathbb{P}\left(q_{t-1} = i, q_t = j | \boldsymbol{x}, \Theta\right) \\ &= \frac{\alpha_{t-1}(i) a_{ij} e_j(x_t) \beta_t(j)}{\mathbb{P}\left(\boldsymbol{x}|\Theta\right)} \end{aligned} \tag{4.14}$$

$$\begin{aligned} n_i(t) &= \mathbb{P}\left(q_t = i | \boldsymbol{x}, \Theta\right) \\ &= \frac{\alpha_t(i)\beta_t(i)}{\mathbb{P}\left(\boldsymbol{x}|\Theta\right)}. \end{aligned} \tag{4.15}$$

The Baum-Welch algorithm acts as a local search algorithm and is liable to become trapped at a local optimum.

If we train on a limited amount of data this estimation of the emission probabilities is liable to overfit the data. For example, we may not have seen some symbol a emitted at time t in our training data. If we build an HMM with a zero probability of emitting a symbol at this time we would perfectly fit our training data, but we may be reading more into a finite sample of data than we should be. To avoid excessive overfitting we can add some 'pseudo-counts', α_{ij} to our estimate for n_{ij}, and $\alpha_i(a)$ to our estimate for $n_i(a)$. Our estimated transition and emission probabilities are given by

$$a_{ij} = \frac{n_{ij} + \alpha_{ij}}{\sum_{j' \in \mathcal{Q}} (n_{ij'} + \alpha_{ij'})} \tag{4.16}$$

$$e_i(a) = \frac{n_i(a) + \alpha_i(a)}{\sum_{a' \in \mathcal{S}} (n_i(a') + \alpha_i(a'))}. \tag{4.17}$$

Although, this appears to be an *ad hoc* fix, it can be motivated from a Bayesian perspective. We can consider the training set to be a sample from a multinomial distribution and assume a Dirichlet distribution for the prior probability, then the pseudo-counts drop out as the coefficients of the prior [70]. For more details on the training of HMMs see *e.g.* [17, 18, 71].

4.3.2 Gradient Based Methods

In the gradient based method, the parameters of an HMM Θ are updated according to the standard formula,

$$\Theta^{new} = \Theta^{old} - \eta \left. \frac{\partial J}{\partial \Theta} \right|_{\Theta = \Theta_{old}} \tag{4.18}$$

where J is a quantity to minimise. We can define

$$J = E_{ML} = -\log(\mathbb{P}(\boldsymbol{x}|\Theta)) = -\log(L_{tot}), \tag{4.19}$$

where $L_{tot} = \mathbb{P}(\boldsymbol{x}|\Theta)$ is the likelihood of the observation given the HMM variables.

From the definition of forward(4.5) and backward variables (4.9),

$$L_{tot} = \sum_{i \in \mathcal{Q}} \mathbb{P}(\boldsymbol{x}, i|\Theta) = \sum_{i \in \mathcal{Q}} \alpha_t(i)\beta_t(i). \tag{4.20}$$

Differentiating equation 4.19 with regard to Θ, we get

$$\frac{\partial J}{\partial \Theta} = -\frac{1}{L_{tot}} \frac{\partial L_{tot}}{\partial \Theta}. \tag{4.21}$$

Since we have transition probabilities $a_{ij} (i, j \in \mathcal{Q})$, and observation probabilities $e_i(a) (i \in$

$\mathcal{Q}, a \in \mathcal{S}$) independent from each other, we can find a derivative for each of parameter sets.

Using the chain rule,

$$\frac{\partial L_{tot}}{\partial a_{ij}} = \sum_{t=1}^{T} \frac{\partial L_{tot}}{\partial \alpha_t(j)} \frac{\partial \alpha_t(j)}{\partial a_{ij}} \tag{4.22}$$

By differentiating equation 4.20 w.r.t $\alpha_t(j)$ and equation 4.6 w.r.t a_{ij}, and we get

$$\frac{\partial L_{tot}}{\partial \alpha_t(j)} = \beta_t(j) \tag{4.23}$$

$$\frac{\partial \alpha_t(j)}{\partial a_{ij}} = e_j(x_t)\alpha_{t-1}(i). \tag{4.24}$$

Then we can get

$$\frac{\partial J}{\partial a_{ij}} = -\frac{1}{L_{tot}} \sum_{t=1}^{T} \beta_t(j) e_j(x_t) \alpha_{t-1}(i). \tag{4.25}$$

In the same way we can calculate the gradient with regard to observation probabilities. Using the chain rule again,

$$\frac{\partial L_{tot}}{\partial e_j(x_t)} = \sum_{t=1}^{T} \frac{\partial L_{tot}}{\partial \alpha_t(j)} \frac{\partial \alpha_t(j)}{\partial e_j(x_t).} \tag{4.26}$$

Differentiating equation 4.6 w.r.t $e_j(x_t)$ we get

$$\frac{\partial \alpha_t(j)}{\partial e_j(x_t)} = \frac{\alpha_t(j)}{e_j(x_t)}. \tag{4.27}$$

Finally, the gradient w.r.t observation is

$$\frac{\partial J}{\partial e_j(x_t)} = -\frac{1}{L_{tot}} \sum_{t=1}^{T} \frac{\alpha_t(j)\beta_t(j)}{e_j(x_t)}. \tag{4.28}$$

Using equation 4.25 and equation 4.28 we can update HMM parameters.

4.4 Decoding Methods

4.4.1 Viterbi Decoding

Decoding is a task to find a sequence of hidden states that best explains the observations. The most common decoding algorithm is the Viterbi algorithm. The Viterbi algorithm finds the most probable path through the model given the observation sequence. The Viterbi algorithm can be computed using essentially the same algorithm as the forward probability calculation except that the summation is replaced by a maximum operation.

Initially all state except the starting state have the value 0. At the first step we define

$$\delta_1(i) = \pi_i e_i(x_1), \qquad 1 \leq i \leq N \tag{4.29}$$

$$\psi_1(i) = 0 \tag{4.30}$$

To find the highest probability along a single path, at time t, we define $\delta_t(i)$ as follows. For a given model Θ, $\delta_t(j)$ is the maximum likelihood of observing sequence from x_1 to x_t and being in state j at time t. The most probable path q^* can be found recursively using

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}]\, e_j(x_t) \qquad 2 \leq t \leq T, 1 \leq j \leq N \tag{4.31}$$

$$\psi_t(j) = \arg\max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] \qquad 2 \leq t \leq T, 1 \leq j \leq N \tag{4.32}$$

Viterbi algorithm terminates at time T

$$q_T^* = \arg\max_{1 \leq i \leq N} [\delta_T(i)] \tag{4.33}$$

The most probable path of state at each time t is obtained by backtracking the states that yield the maximum probability.

$$q_t^* = \psi_{t+1}(q_{t+1}^*). \qquad t = T-1, T-2, \ldots, 1 \tag{4.34}$$

4.4.2 Posterior Decoding

The posterior decoding method is useful when there are more than one path that has similar probability as the most probable one. In general, several states can produce the same emission, given a model and a sequence. The posterior decoding method considers all probabilities that have the same emission at time t. The probability of being in state i, given a sequence x and an HMM Θ,

$$\begin{aligned}\mathbb{P}(q_t = i|\boldsymbol{x}, \Theta) &= \frac{\mathbb{P}(q_t = i, \boldsymbol{x}|\Theta)}{\mathbb{P}(\boldsymbol{x}|\Theta)} \\ &= \frac{\mathbb{P}(x_1 \dots x_t, q_t = i, |\Theta)\,\mathbb{P}(x_{t+1} \dots x_T|x_1 \dots x_t, q_t = i, \Theta)}{\mathbb{P}(\boldsymbol{x}|\Theta)} \\ &= \frac{\mathbb{P}(x_1 \dots x_t, q_t = i, |\Theta)\,\mathbb{P}(x_{t+1} \dots x_T|q_t = i, \Theta)}{\mathbb{P}(\boldsymbol{x}|\Theta)} \\ &= \frac{\alpha_t(i)\beta_t(i)}{\mathbb{P}(\boldsymbol{x}|\Theta)}. \end{aligned} \tag{4.35}$$

This can be evaluated recursively by the forward and backward algorithms. Then we can get the state sequence

$$q_t^* = \arg\max \mathbb{P}(q_t = i|\boldsymbol{x}, \Theta) \tag{4.36}$$

The posterior decoding gives the most probable state given a sequence. It is useful especially if we are interested in a state at a specific position i and not in the whole sequence of states. However, it does not consider the validity of a path of the states when the path is calculated. Therefore, it may produce a path forbidden in the HMM (when a transition probability is zero). To overcome this problem a hybrid method of posterior and Viterbi algorithm [72] and posterior decoder with homology information has been developed [73].

4.5 Class HMMs

A class HMM (CHMM) is an HMM where the states emit class labels l ($\in \mathcal{L}$), as well as a symbol from the alphabet, $\mathcal{S}$. That is, we can associate with a sequence $\boldsymbol{x}$ a corresponding sequence of symbols $\boldsymbol{y} = (y_1, y_2, \dots, y_T)$. Denoting the set of states by $\mathcal{Q}$, and letting $\boldsymbol{q} = (q_1, q_2, \dots, q_T)$ be a sequence of states again, then the likelihood of a sequence $\boldsymbol{x}$ with class labels $\boldsymbol{y}$ is given by

$$\mathbb{P}(\boldsymbol{x}, \boldsymbol{y}|\Theta) = \sum_{\boldsymbol{q} \in \mathcal{Q}^T} \mathbb{P}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{q}|\Theta), \tag{4.37}$$

where the sum is over all possible paths through the states (paths without transition probabilities have probability zero).

Given a sequence $\boldsymbol{x}$ (or set of sequence) and the corresponding labels $\boldsymbol{y}$, we find a maximum-likelihood (ML) set of parameters

$$\Theta^{ML} = \arg\max_{\Theta} \mathbb{P}(\boldsymbol{x}, \boldsymbol{y}|\Theta). \tag{4.38}$$

This can be calculated efficiently using the modified Baum-Welch algorithm [17]. For the labelled sequence we only allow valid paths through the model to calculate forward and backward variables. When a path is valid the state labels and the sequence labels coincide with each other.

Let $m_{ij}(t)$ be the expected number of transitions from state i to state j at time t along valid paths,

$$\begin{aligned} m_{ij}(t) &= \mathbb{P}\left(q_{t-1} = i, q_t = j | \boldsymbol{x}, \boldsymbol{y}, \Theta\right) \\ &= \frac{\mathbb{P}\left(q_{t-1} = i, q_t = j, \boldsymbol{x}, \boldsymbol{y}, | \Theta\right)}{\mathbb{P}\left(\boldsymbol{x}, \boldsymbol{y} | \Theta\right)} \end{aligned} \tag{4.39}$$

and $m_i(t)$ be the expected number of times the state i is visited at time t along valid paths

$$\begin{aligned} m_i(t) &= \mathbb{P}\left(q_t = i | \boldsymbol{x}, \boldsymbol{y}, \Theta\right) \\ &= \frac{\mathbb{P}\left(q_t = i, \boldsymbol{x}, \boldsymbol{y} | \Theta\right)}{\mathbb{P}\left(\boldsymbol{x}, \boldsymbol{y} | \Theta\right)} \end{aligned} \tag{4.40}$$

then the modified Baum-Welch algorithm replace equation 4.14 and equation 4.15 with equation 4.42 and equation 4.43.

If we define

$$\delta_{y_t, c^j} = \begin{cases} 1 & \text{if } y_t = c^j \\ 0 & \text{otherwise} \end{cases} \tag{4.41}$$

where c^j is the label of the state j, then

$$n_{ij}(t) = \frac{\tilde{\alpha}_{t-1}(i) a_{ij} e_j(x_t) \delta_{y_t, c^j} \tilde{\beta}_t(j)}{\sum_{i' \in Q} \tilde{\alpha}_t(i') \tilde{\beta}_t(i')} \tag{4.42}$$

$$n_i(t) = \frac{\tilde{\alpha}_t(i) \tilde{\beta}_t(i)}{\sum_{i' \in Q} \tilde{\alpha}_t(i') \tilde{\beta}_t(i')}. \tag{4.43}$$

where $\tilde{\alpha}$ and $\tilde{\beta}$ are calculated recursively using the forward and the backward algorithm along valid paths only.

4.5.1 Conditional Maximum Likelihood (CML) Estimation

In the case of maximising the probability of correct labelling, conditional maximum likelihood (CML) can be used [74].

$$\Theta^{CML} = \arg\max_{\Theta} \mathbb{P}(\boldsymbol{y}|\boldsymbol{x}, \Theta) \tag{4.44}$$

where

$$\mathbb{P}(\boldsymbol{y}|\boldsymbol{x}, \Theta) = \frac{\mathbb{P}(\boldsymbol{x}, \boldsymbol{y}|\Theta)}{\mathbb{P}(\boldsymbol{x}|\Theta)} \tag{4.45}$$

We define the negative log conditional likelihood

$$\mathcal{L} = -\log(\mathbb{P}(\boldsymbol{y}|\boldsymbol{x}, \Theta)) = \mathcal{L}_c(\Theta) - \mathcal{L}_f(\Theta), \tag{4.46}$$

where the negative log likelihood $\mathcal{L}_c(\Theta)$ is in the clamped phase and calculated along the valid path, and $\mathcal{L}_f(\Theta)$ is in the free running phase which does not consider labels.

$$\begin{aligned} \mathcal{L}_c(\Theta) &= -\log(\mathbb{P}(\boldsymbol{x}, \boldsymbol{y}|\Theta)) && (4.47) \\ \mathcal{L}_f(\Theta) &= -\log(\mathbb{P}(\boldsymbol{x}|\Theta)). && (4.48) \end{aligned}$$

The derivative of the log likelihood $\mathcal{L}_f(\Theta)$ w.r.t. a generic parameter $\omega \in \Theta$ can be written

$$\begin{aligned} \frac{\partial \mathcal{L}_f(\Theta)}{\partial \omega} &= -\frac{1}{\mathbb{P}(\boldsymbol{x}|\Theta)} \frac{\partial \mathbb{P}(\boldsymbol{x}|\Theta)}{\partial \omega} \\ &= -\frac{1}{\mathbb{P}(\boldsymbol{x}|\Theta)} \sum_{\pi} \frac{\partial \mathbb{P}(\boldsymbol{x}, \pi|\Theta)}{\partial \omega} \\ &= -\frac{1}{\mathbb{P}(\boldsymbol{x}|\Theta)} \sum_{\pi} \mathbb{P}(\boldsymbol{x}, \pi|\Theta) \frac{\partial \log \mathbb{P}(\boldsymbol{x}, \pi|\Theta)}{\partial \omega} \\ &= -\sum_{\pi} \mathbb{P}(\pi|\boldsymbol{x}, \Theta) \frac{\partial \log \mathbb{P}(\boldsymbol{x}, \pi|\Theta)}{\partial \omega} \end{aligned} \tag{4.49}$$

Because the log likelihood is

$$\mathcal{L}_f(\Theta) = \sum_{\pi} \prod_{t=1}^{T} a_{\pi_{t-1}\pi_t} e_{\pi_t}(x_t) \tag{4.50}$$

the gradient of negative log likelihood $\mathcal{L}_f(\Theta)$ in the free running phase is

$$\frac{\partial \mathcal{L}_f(\Theta)}{\partial \omega} = -\sum_{t,i} \frac{n_i(t)}{e_i(x_t)} \frac{\partial e_i(x_t)}{\partial \omega} - \sum_{t,i,j} \frac{n_{ij}(t)}{a_{ij}} \frac{\partial a_{ij}}{\partial \omega}, \tag{4.51}$$

where $n_{ij}(t)$ is the expected number of times a transition from state i to state j at time t and $n_i(t)$ is the expected number of times we are in state i at time t in the free running phase. Both are also defined in equation 4.14 and equation 4.15 respectively. In a similar way the gradient of the negative log likelihood $\mathcal{L}_c(\Theta)$ in the clamped phase is computed.

$$\frac{\partial \mathcal{L}_c(\Theta)}{\partial \omega} = -\sum_{t,i} \frac{m_i(t)}{e_i(x_t)} \frac{\partial e_i(x_t)}{\partial \omega} - \sum_{t,i,j} \frac{m_{ij}(t)}{a_{ij}} \frac{\partial a_{ij}}{\partial \omega}, \tag{4.52}$$

where $m_{ij}(t) = \mathbb{P}(\pi_{t-1} = i, \pi_t = j | \boldsymbol{x}, \boldsymbol{y}, \Theta)$ is the expected number of times a transition from state i to state j at time t along the allowed paths, and $m_i(t) = \mathbb{P}(\pi_l = i | \boldsymbol{x}, \boldsymbol{y}, \Theta)$ is the expected number of times we are in state i at time t along the allowed paths.

When ω is a transition probability

$$\frac{\partial \mathcal{L}}{\partial a_{ij}} = -\frac{m_{ij} - n_{ij}}{a_{ij}} \tag{4.53}$$

and the derivative of $e_i(a)$ can be expressed in a similar way by substituting m_{ij} and n_{ij} with $m_i(a)$ and $n_i(a)$.

$$\frac{\partial \mathcal{L}}{\partial e_i(a)} = -\frac{m_i(a) - n_i(a)}{e_i(a)}. \tag{4.54}$$

For the transition probability we define

$$a_{ij} = \frac{e^{z_{ij}}}{\sum_{j'} e^{z_{ij'}}}, \tag{4.55}$$

where z_{ij} are the new unconstrained auxiliary variables. Now z_{ij} are always positive and normalised.

Now gradient-based optimisation in the auxiliary variables yield a new estimation of z_{ij},

$$z_{ij}^{(t+1)} = z_{ij}^{(t)} + \Delta z_{ij}^{(t)}. \tag{4.56}$$

The parameter estimation is then

$$\begin{aligned} a_{ij}^{(t+1)} &= \frac{\exp(z_{ij}^{(t)} + \Delta z_{ij}^{(t)})}{\sum_{j'} \exp(z_{ij'}^{(t)} + \Delta z_{ij'}^{(t)})} \\ &= \frac{a_{ij}^{(t)} \exp(\Delta z_{ij}^{(t)})}{\sum_{j'} a_{ij'}^{(t)} \exp(\Delta z_{ij'}^{(t)})}, \end{aligned} \tag{4.57}$$

The gradient of $\mathcal{L}$ w.r.t z_{ij} can be written entirely in terms of a_{ij} and $m_{ij} - n_{ij}$. From equation 4.53 and equation 4.55

$$\frac{\partial \mathcal{L}}{\partial z_{ij}} = -\left[m_{ij} - n_{ij} - a_{ij} \sum_{j'} (m_{ij'} - n_{ij'}) \right]. \tag{4.58}$$

We can update parameters by applying equation 4.58 to equation 4.57. In other way we can set equation 4.58 to zero and solve for a_{ij}. However, the solution often becomes negative. To overcome this we adopt an arbitrary number D. The derivative does not change if an arbitrary positive number D is added and subtracted, so we might write

$$\frac{\partial \mathcal{L}}{\partial z_{ij}} = -(m_{ij} - n_{ij} + D a_{ij}) + a_{ij} \left[D + \sum_{j'} (m_{ij'} - n_{ij'}) \right]. \tag{4.59}$$

Then update rule for $a_i j$ is

$$a_{ij}^{(t+1)} = \frac{a_{ij}^{(t)} \left(\frac{m_{ij}^{(t)} - n_{ij}^{(t)}}{a_{ij}^{(t)}} + D \right)}{\sum_{j'} a_{ij'}^{(t)} \left(\frac{m_{ij'}^{(t)} - n_{ij'}^{(t)}}{a_{ij'}^{(t)}} + D \right)} \tag{4.60}$$

This update rule is known as *extended Baum-Welch reestimation* [75][76]. For a fast convergence D is chosen such that D is always larger than or equal to a small positive constant ϵ [77].

$$D = \max \left[\frac{\partial \mathcal{L}}{\partial a_{ij}}, 0 \right] + \epsilon. \tag{4.61}$$

The update rule for observation $e_i(a)$ can be derived in the same way. For more details on training labelled sequences see *e.g.* [71] and [78].

4.5.2 Posterior Label Probability

The posterior label probability (PLP) calculates probability of a label at a certain position. Unlike the Viterbi algorithm PLP sums the probabilities of being in each state at a specific position of the sequence and assigns the dominant label to that element of the sequence.

The PLP of a label at position t is the sum of posterior probability of all states that emit the same label. The PLP for label l at position t is

$$\mathbb{P}(y_t = l|\boldsymbol{x}, \Theta) = \sum_{i \in Q} \mathbb{P}(y_t = l, q_t = i|\boldsymbol{x}, \Theta). \tag{4.62}$$

We assign each state to a particular class. That is, we take the probability of a label given a state to be 1 if the state is assigned to that class and 0 otherwise. Thus the sum in equation (4.62) only gets contributions from states that have been assigned to class l.

4.6 Parameter Tying

A common method to decrease a model complexity is to 'tie' states together so that they have the same emission and/or transition probabilities [79]. States are tied when we believe that they model similar parts of a sequence. This reduces the number of free parameters that need to be learnt and therefore reduces the problem of overfitting. Biological sequences often have approximately the same biological functionality even though they are slightly different in their characters. A way to incorporate the correlation in the biological string is to tie parameters. Parameter tying is used in many HMM applications such as TMHMM [21].

4.7 Topologies of HMMs

The learning problem for HMMs consists of two components: learning the structure of the model and learning its parameters. The Baum-Welch algorithm and the Extended Baum-Welch algorithm are elegant and efficient ways of learning parameters of HMMs, even though there is no standard way to find the global maximum.

On the other hand, structure learning has not been studied much. Structure learning is more important because the topology of models influences the performance of HMMs. Traditionally topologies of HMMs were carefully designed based on the knowledge about the given problem. Well designed HMMs represent statistical phenomena of sequences in their structure. In general, a too simple model is unlikely to be able to generate the data set with a high likelihood, while a too complex model will easily learn the data

set, but is unlikely to generalise well. Figure 4.2 shows several types of HMMs. In these examples, we only allow one symbol to be emitted from each state (in general, one state could emit any symbol form the alphabet, $\mathcal{S}$, with a probability given by the emission probabilities). The fully connected model, Figure 4.2(a), can generate any sequence. The self-loop model of figure 4.2(b) can generate sequences such as AAATTTGCCC, while the left-right model can generate the same sequence as the self-loop, but can also jump states to produces sequences such as AATTCCC.

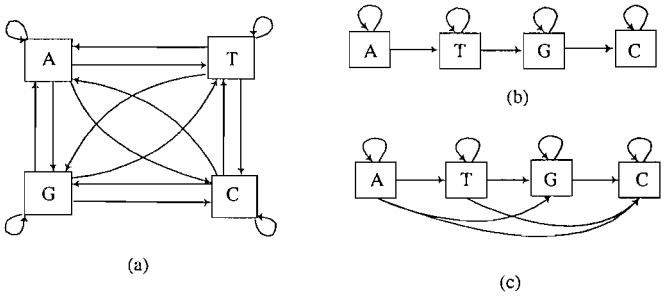


FIGURE 4.2: Several types of HMM topologies; (a) an ergodic model (b) a self loop model (c) a left-right model.

Practically, fully connected models are not used for most of the real problems because of their bad generalisation performance. It is plausible to find a suitable architecture by applying GAs to find the parameter set and deleting insignificant transitions. However, it is hard to achieve good model with a large HMM. Also deleting insignificant transitions is not always appropriate because it may cause loss of required properties of the HMM. Even though several methods using genetic algorithms have been developed for biological sequence analysis to search for the HMM structure, their applications were very limited. Still, most of successful HMMs are hand-designed. The hand-designed models have been preferred because HMMs can encode the knowledge of the sequences. Still, the number of HMM applications for the complex models are limited.

4.8 Machine Learning Methods to Find HMM Topologies

There have been earlier attempts to learn the structure of an HMM. Stolcke [80] transformed an initial large HMM by merging the states. He used the log-likelihood and posterior probability as criteria for choosing which states to merge. But, he did not allow states to be split which might have been useful. A method involving both state splitting as well as deleting negligible states and transitions has also been used to find good HMM topologies [81]. The transition ambiguity and the expected observation differences were used as criteria for splitting a state. These statistical approaches can be used to find a particular pattern like a motif. In practice, however, successful HMMs are constructed by carefully deciding which transitions are to be allowed in the model [18].

Obviously, determining the rule to split and delete HMM states can not be done easily. As the number of the training sequences increases, setting the merging and splitting rule becomes increasingly difficult.

4.8.1 Hybrid of HMMs and GAs

One of the nice features of GAs is that it is easy to incorporate new operators into a GA based on the problem under consideration, and it can be hybridised easily with other machine learning techniques into the GA procedure.

A GA can be used to evolve HMM topologies as the GA can compare a large number of different topologies in parallel. The HMM parameters can be trained easily by hybridising the traditional Baum-Welch training in the GA cycle. Another advantage of HMMs is modularity. Small HMMs can be combined into a large HMMs. Part of the HMM can also work as a module that composes a building block of HMMs. It is the crossover operator in GAs that uses the modularity by swapping those modules.

The use of GAs to find an HMM structure has been tried previously in speech recognition [82]. They considered a 5-state model and evolved a string representing the transitions and emissions of an HMM. However, their method is rather parameter optimisation than evolving the HMM structure. Their approach may be a good way to escape local maxima. Later, they considered evolving a very simple HMM topology [83]. They introduced crossover that swaps states between two HMMs. However, the change on topology was limited to the states with a self loop.

In this thesis we suggest several GA methods for the construction of the HMM topology without prior knowledge of biological sequences. The first model which applied a GA on HMMs used genetic operators which evolve the architecture by changing states. The HMM states are crossed over and added and deleted in this model. The second model used blocks composed of HMM states. We suggest a block model inspired by HMM applications used for biological sequence analysis. The genetic operators are devised to crossover blocks instead of states.

There were attempts of using GAs that practically change the structure of HMMs in bioinformatics area. Yada *et. al* used GAs to insert and delete states and swap transitions. They allowed unlimited number of transitions between states and could model motif pattern. In a similar way, Thomsen [5] developed a genetic scheme to find the HMM structure for protein secondary structure prediction. The fitness functions they used was in the form of Bayesian Information Criterion [84] to balance the complexity and the likelihood.

$$\textit{[Yada et. al's fitness]} \quad w_k = \frac{1}{-2\log\left(\mathbb{P}\left(\boldsymbol{x}_k;\Theta_k\right)\right)+2\lambda p_k} \tag{4.63}$$

$$\textit{[Thomesen's fitness]} \quad w_k = \log\left(\mathbb{P}\left(\boldsymbol{x}_k;\Theta_k\right)\right)+\lambda\frac{\log(l+1)}{2}p_k, \tag{4.64}$$

where p_k is the number of free parameters. The log-likelihood function has intrinsic drawback of local maxima and overfitting. They tried to control overfitting by regulating the number of free parameters. However, the choice of balancing parameter λ is very sensitive and in our preliminary analysis we did not find it useful. Also, it is not guaranteed that an HMM with less free parameters always has better performance.

In the next chapter we present the fitness evaluation method that does not require balancing parameter and free parameters of HMMs. Instead, we use statical information of the data associated with an HMM to reduce overfitting. In chapter 6 we introduce a genetic method evolving HMMs for whole sequences.

Chapter 5

Genetic Algorithms for Hidden Markov Models(GA-HMMs)

5.1 Genetic Algorithms for Hidden Markov Models

HMMs have been applied to many biological sequencing problems successfully. The success of HMMs owes much to their ability to encode biological information in their structure while allowing many unknown quantities to be learnt through the optimisation of their transition and emission probabilities. The constraints imposed by their structure will often limit excessive overfitting of the training data, although some overfitting is still observed when using Baum-Welch training.

Automatic optimisation of the structure of HMMs would potentially be highly beneficial. In many applications the biological mechanism is not fully understood. Given sufficient data it would be possible to learn an HMM architecture that encodes some biological information that we are unaware of. However, in learning the structure of an HMM we do not wish to lose the advantages they currently offer. In particular, we wish to control the complexity of the HMM models and if possible retain their biological interpretability.

In this thesis, we investigate the effectiveness of using Genetic Algorithms (GAs) for optimising the HMM structure. A GA is a robust general purpose optimisation technique which evolves a population of solutions [22]. It is easy to hybridise other algorithms such as Baum-Welch training within a GA. Furthermore, it is possible to design operators which favour biologically plausible changes to the structure of an HMM. That is, to ensure that modules of the states are kept intact. GAs have been widely used to optimise architecture for Neural Networks [23]. However, we could find only a couple of applications of GAs to optimise the structure of an HMM. Yada *et al.* [24] used a GA to find a TATA box model. Thomsen [5] developed a genetic scheme to find the HMM structure for protein secondary structure prediction. They included a term in their fitness func-

tion to penalise over-complex models, however, their results depended critically on this parameter. In this thesis, we explore the use of GAs for evolving HMMs. Our GA differs from previous methods in that we split the training data into two sets. One set is used for training the HMMs using Baum-Welch, the other set is used to evaluate the HMM's fitness. This reduces overfitting of the training data. In addition, to prevent overfitting we performed Baum-Welch only on a proportion of randomly selected individuals in the population at each generation. To discover if GAs are potentially useful for evolving HMMs we implemented a standard GA where a population of HMMs are evolved from one generation to the next. At each generation some proportion of the HMMs are trained with Baum-Welch on a training set. The fitnesses of the HMMs are measured on a evaluation set and the fitter members are selected. Finally the members are mutated and crossed-over to form then next generation. This procedure is shown in figure 5.1. The state labelled *genetic operations* include selection, mutation and crossover.

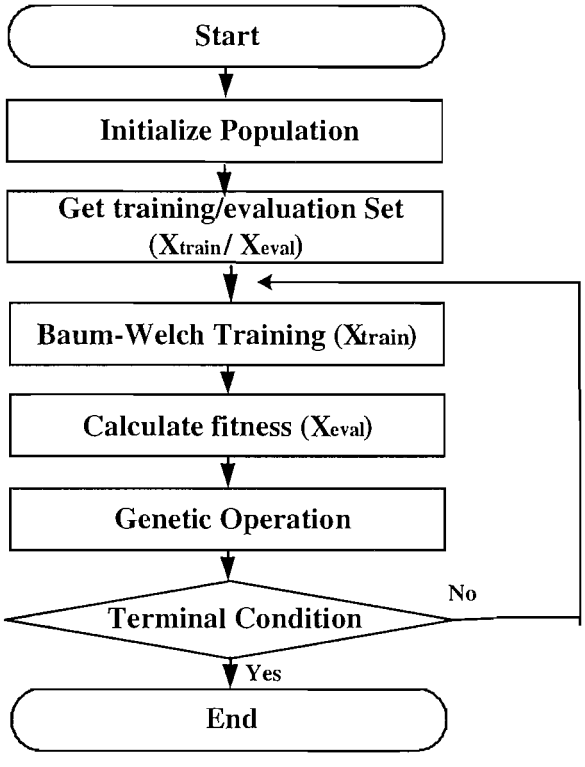


FIGURE 5.1: The GA-HMM algorithm. Baum-Welch training is combined with selection, mutation and crossover to evolve HMMs. We separate the training sequences ($\boldsymbol{x}$) is into the training set ($\boldsymbol{x}_{train}$) and the evaluation set ($\boldsymbol{x}_{eval}$). $\boldsymbol{x}_{train}$ is used for the Baum-Welch training and $\boldsymbol{x}_{eval}$ is used for the fitness calculation.

In the experiments described below, the initial population consists of HMMs with just two states. The number of states will change due to state insertion and deletion mutations and through crossover. Also as part of the initialisation stage the sequence data ($\boldsymbol{x}$) is divided into a set used for training with Baum-Welch ($\boldsymbol{x}_{train}$) and a set used for

evaluating the fitness (x_{eval}). The algorithm terminates when there is no significant change in the structural model.

5.2 Genetic Operations for GA-HMMs

The genetic operations consist of selection, mutation and crossover. Selection uses proportional selection with stochastic universal sampling [65] to reduce genetic drift. For a mutation to be useful it should make changes which cause minimal disruption so that the new HMM has a high probability of having a fitness close to that of the unmutated HMM. We considered mutations that only change either the number of states or the number of transitions by one. This gave us four mutation operators; insert state, delete state, insert transition and delete transition, which are shown in figure 5.2. Insertion of a state can happen between any two states or at either end of the chain. When a state is inserted, the states on its right hand side shift by one as shown in figure 5.2(a). The emission probabilities of the new state are set to randomly selected values. If a state is deleted, all its transitions are removed. Insertion of a transition can happen between any two states and deletion of a transition happens at any state if the state has more than one outgoing transition. Although, these mutations allow highly interconnected HMMs they provide a certain bias towards chain structures because of the state insertion operator.

Crossover takes place between two HMMs and exchanges states. A number of successive states can be crossed over in one operation. Only outgoing transitions from a state are exchanged during crossover. An example of crossover is shown in figure 5.3.

5.3 Selective Baum-Welch

The Baum-Welch algorithm is commonly used to train HMMs. It estimates model parameters from training sequences while maximising the log-likelihood of the model. The log-likelihood of model k is

$$\mathcal{L}_k = \log\left(P(\boldsymbol{x}|\Theta_k)\right), \tag{5.1}$$

where Θ_k denotes the parameters of the kth HMM and $\boldsymbol{x}$ is the given sequences.

Although Baum-Welch maximises the log-likelihood for the sequences, it can overfit the given sequences and produce inferior results compared with HMMs that have experienced fewer training cycles. We tested this by training sequences with Baum-Welch

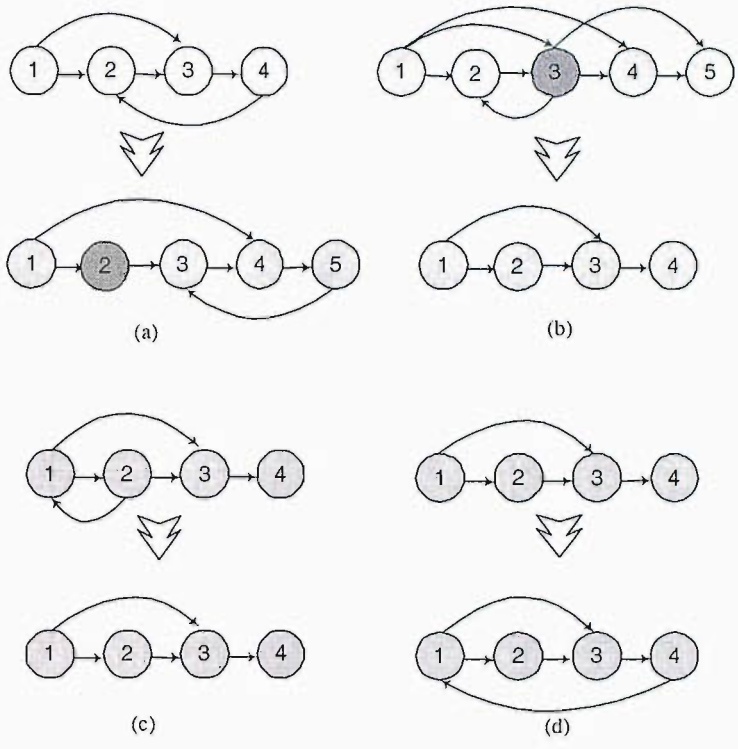


FIGURE 5.2: Four types of mutations (a) Insert state (inserting a state in the second position), (b) delete state (delete the third state), (c) delete transition, (d) insert transition.

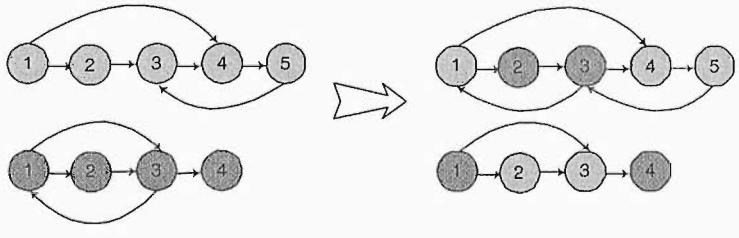


FIGURE 5.3: Crossover. During crossover outgoing transitions move with transition.

without applying the GA algorithm. Figure 5.4 shows the negative log-likelihood versus the number of Baum-Welch iterations in an experiment with one of the models described later. The negative log-likelihood measured on the sequence data decreases monotonically, whereas on unseen data it initially decreases, but then increases again due to overfitting. Initially, we tested a GA where Baum-Welch was performed at each generation. However, this caused overfitting as the HMMs were trained too frequently. A selective Baum-Welch scheme was adopted to reduce overfitting. The generalisation performance was found to improve by changing the algorithm so that Baum-Welch training only occurred with a fixed probability. In this scheme, 20 percent of population are randomly selected in each generation for the Baum-Welch training. In addition, we made sure that the fittest member of the population was never subjected to Baum-Welch

training as this could cause a loss of fitness for the best member of the population. The overfitted HMM structures receive lower fitness value when the HMM is applied to the test set.

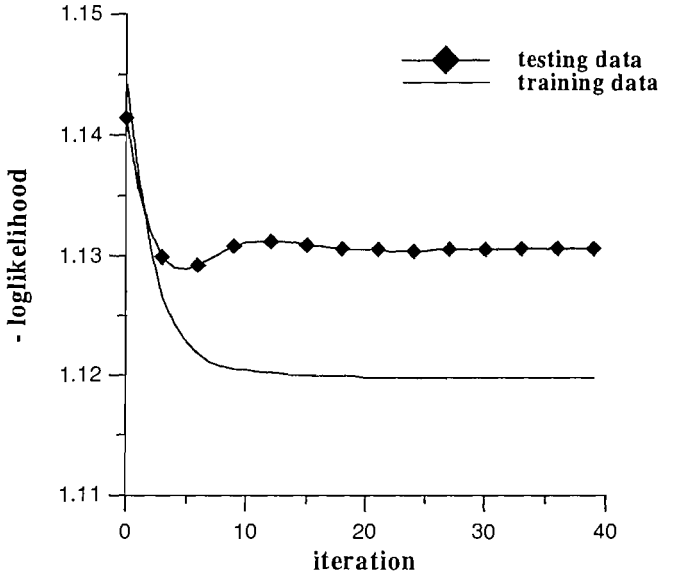


FIGURE 5.4: The graph show the negative log-likelihood for an HMM plotted against the number of Baum-Welch iterations. The negative log-likelihood for the given data is monotonically decreased by the Baum-Welch algorithm. However, the negative log-likelihood measured on independent testing data will typically decrease, reach a minimum and then increase again. In this instance the best generalisation performance was found after 5 iterations. The training data used was from *C. jejuni* and is described in simulation I below.

5.4 Fitness Value

The performance of the GA was found to depend strongly on the fitness function used. The fitness function used by Yada *et al.* [24] incorporated a balance factor between complexity and likelihood

$$w_k = \frac{1}{-2\left(\mathcal{L}(\boldsymbol{x}|\Theta_k)\right) + 2\lambda p_k}. \tag{5.2}$$

Here, $\mathcal{L}(\boldsymbol{x}|\Theta_k)$ is the maximum logarithmic likelihood estimate of the kth HMM, p_k is the number of free parameters in the kth HMM (its complexity) and λ is the balance factor. By adjusting the balance factor, the complexity can be controlled. However, the complexity was found to be very sensitive to the value of λ and it was difficult to choose a value for λ *a priori*.

In this chapter, we avoided the problem of setting the balance factor by separating the

training data($\boldsymbol{x}$) into a set used for Baum-Welch training($\boldsymbol{x}_{train}$) and a set used for evaluation($\boldsymbol{x}_{eval}$). The fitness function used in our GA-HMM algorithm is

$$E_\mu = \frac{1}{-\sum_i \log\left(P(x_i|\Theta_\mu)\right)/l_i}, \tag{5.3}$$

where l_i is the length of a sequence x_i and μ labels the different HMMs (with parameters Θ_μ) of the population. A member of the population is selected with a Boltzmann probability

$$F_\mu = \frac{m_\mu}{\sum_{\nu=1}^{N} m_\nu}, \qquad m_\mu = \mathrm{e}^{s\,E_\mu/\sigma} \tag{5.4}$$

where F_k is the fitness value of the kth HMM and σ is the standard deviation of the fitness in the population and s is a constant that controls the strength of the selection. In the work reported here, we used a value of s equal to 1.

This method is intended to reduce overfitting. As the training and testing sets are separated, overfitted models get lower fitness values in the fitness evaluation stage and will be discarded through the selection operation.

5.5 Implementation

5.5.1 Simulation I: Coding Region Model of *C. jejuni*

To see if a GA was capable of finding biologically interpretable patterns we performed the first experiment using 556 sequences from the coding regions from *Campylobacter jejuni* (hereafter *C. jejuni*). *C. jejuni* is an important human intestinal pathogen. Despite intensive study, much is still not known about how to control and intervene in the disease [85]. A better understanding of gene organization, function and regulation in *C. jejuni* is desirable to provide possible control strategies.

The sequence data comprises a start codon (ATG), some number of codons and a stop codon (TAA, TAG or TGA). A simple HMM architecture for detecting this region would consist of a 3 state loop (figure 5.5). A third order state model was used to model the codon region. That is, instead of a state emitting a symbol from a single base unconditional distribution, the state emits a symbol, which is dependent on the three previous bases in the sequence, through a conditional probability distribution. We again used a 2-state HMM as an initial model for the coding region as shown in figure 5.6. The transition and emission probability for the start codon and stop codon was not evolved. Each state is a third order. The GA was run for 600 generations and of the 566 sequences 160 sequences were used for testing. Table 5.1 shows the parameters used in the experiment.

TABLE 5.1: GA-HMM parameters used in the experiment I

Parameter	value
Population size	30
Offspring size	4
Iteration	600
Crossover rate	0.07
Insert & delete transition rate	0.07
Insert & delete state rate	0.13

FIGURE 5.5: HMM architecture for *C. jejuni* coding region

Figure 5.7 shows some of the HMM architectures produced by the GA in this experiment. Even though some results were hard to interpret, almost every simulation showed special biological patterns in the structure. Figure 5.7(a) shows the same result as the common model shown in figure 5.5. Figure 5.7(b) has a path 1-3-4,(2-3-4)—here, brackets indicate a loop. Figure 5.7(c) has (1-5-6) and (1-2-3, 4-5-6) loops. Figure 5.7(d) has 1-2-3, (9-7-8), 9-10-11 or 1-2-3, 4-5-6, (6-6-6), 6-7-8, (9-7-8), 9-10-11 loops. These HMM solutions, although complicated, predominantly generate triplets. We performed ten experiment and on every experiment with coding region we could find the clean 3 state loops as shown Figure 5.7 (a)(b)(c). Figure 5.7(d) shows the worst result we have. This result was due to insufficient time to evolve. The numbers shown are transition probabilities. Other transition probabilities are easily calculated, because the sum of outgoing transitions from a state is always 1. Because of the transition probabilities the most probable loop are 1-2-3, (9-7-8) or 1-2-3, (9-7-8), 9-10-11.

Figure 5.8 shows the maximum fitness value versus iterations that produced the result of figure 5.7(b). In the graph we show first 40 iterations. The other part of the graph is just a straight line. In the figure we can find abrupt increase of the learning curve. It is because the GA-HMM found a new structural model that can replace the old ones.

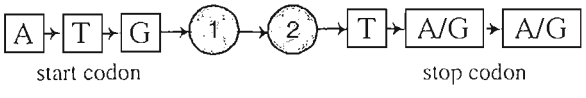


FIGURE 5.6: Initial HMM architecture

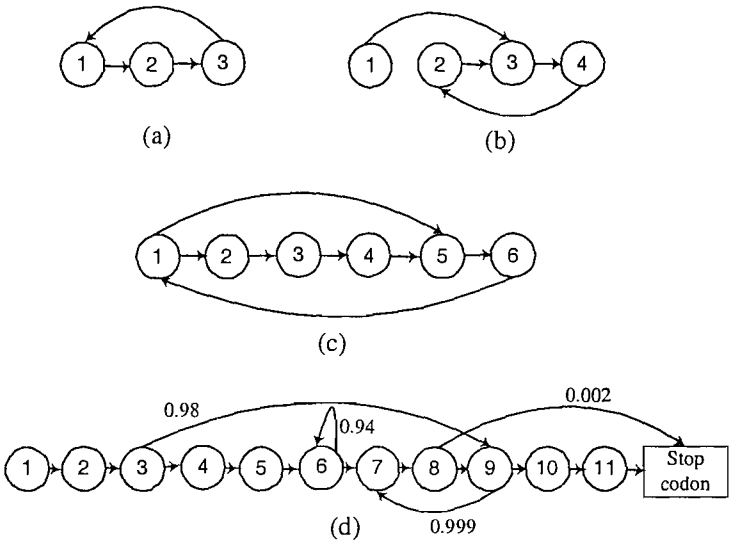


FIGURE 5.7: Result of simulation for the *C. jejuni* coding region: (a) has loop (1-2-3),(b) has a path 1-3-4,(2-3-4) (c) has (1-5-6) and (1-2-3, 4-5-6) loops. (d) has 1-2-3, (9-7-8), 9-10-11 or 1-2-3, 4-5-6,(6-6-6), 6-7-8, (9-7-8),9-10-11.

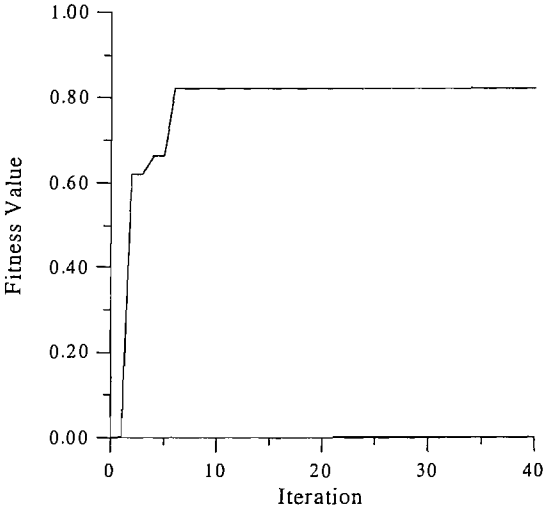


FIGURE 5.8: After training the *C. jejuni* sequences, GA-HMM found one structure model for the periodic signal.

5.5.2 Simulation II: Promoter Model of *C. jejuni*

Our second experiment was to find an HMM for the promoter region of *C. jejuni*. In many bacteria, there is a conserved sequence of TTGACA at around 35 base pairs (bp) before the transcription start site (position '-35'[1]) and TAtAAT (where 't' indicates either T or A) at around the -10 region, called the TATA box [86]. A ribosome binding site is located between the transcription start site and the start codon of the gene, and it is typically AAGGA. In the promoter region of *C. jejuni*, the TTGACA region is weakly conserved, but a T-rich domain is common upstream of the TATA box [87]. The *C. jejuni* genome contains more Ts and As than Gs and Cs. Although the Ribosomal binding sites can often be spotted, the sequence is not always in the same position in relation to the coding region. In some cases, the sequence AAGGA is highly mutated. It is difficult for non-experts to figure out which part of the sequence is the TATA box because most part of the promoter region is composed of Ts and As.

Petersen *et al.* suggested an HMM architecture for this promoter region [88]. Their model includes the TATA box (TAtAAT) and a Ribosomal binding site (AAGGA). During the testing of various models a periodic pattern was discovered upstream of the TATA box, and this was included in the model. This model is shown in figure 5.9. The forward transitions are used to represent sequences with varying length. The states in the background region are tied together to represent non-specific sequences with a small number of states. The promoter consists of approximately repeated blocks with an observed period of 10.6 nucleotides. A hand crafted model for this region used in [88] is shown in figure 5.10.

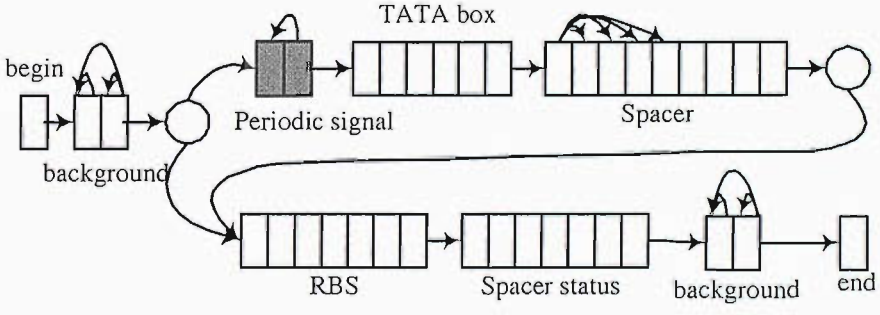


FIGURE 5.9: Model for predicting the promoter region of *C. jejuni* from L. Petersen *et al.*, (2003). In simulation II we try to learn the periodic region, starting from HMMs with two states.

In our experiment we evolved a population of HMMs for periodic signal region using a GA starting from two states models. The parameters controlling the GA are shown in table 5.2. The data set consists of 175 sequences and 135 sequences were used for training (x) and 40 for testing (x_{test}). Five fold cross validation tests were performed

[1] '-35' means it is located 35bp before the transcription start site.

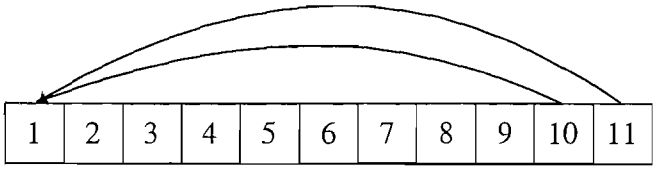


FIGURE 5.10: The hand constructed HMM model for the periodic signal of *C. jejuni* promoter used in L. Petersen *et al.*, (2003).

to obtain a more reliable estimate of the generalisation performance. Figure 5.11 illustrates the partition of the data set when conducting a cross-validation test. Of the 135 sequences 95 were used for Baum-Welch training (x_{train})and 40 were used for measuring the fitness (x_{eval}). The data used in this experiment can be obtained from http://www.ecs.soton.ac.uk/~ kjw02r/data.html.

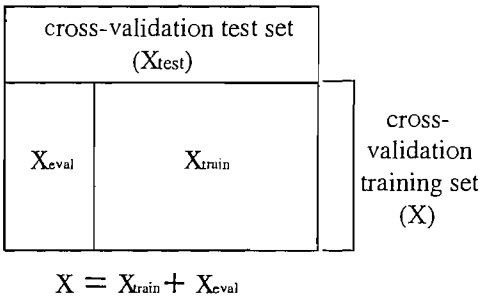


FIGURE 5.11: The data separating scheme for a cross-validation experiment. The cross-validation training set is composed of a training(x_{test}) set and a evaluation set(x_{eval}).

TABLE 5.2: GA-HMM parameters used in the experiment II

Parameter	value
Population size	30
Offspring size	4
Iteration	400
Crossover rate	0.06
Insert / delete transition	0.07
Insert / delete state	0.13

Figure 5.12(a) shows the fitness of the best member of the population versus the generation for the first data partition. Only evaluation sequences which are not involved in the training are used in calculating the fitness value (equation 5.3). Because the GA-HMM reduces overfitting, the fitness value of the fittest member of the population always increases. The graphs for the other cross validation sets look similar. In figure 5.12(b) the average number of states in the HMM is shown. We observe that the number of states grows until it reaches around 10 which appears to support the observed periodicity found in this region of the promoter. At this stage the number of states remains roughly fixed

while the fitness value continues to rise indicating that the fine structure of the model is still evolving. Occasionally, the number of states found by the GA-HMM was less than 10 states. We believe this is because the GA-HMM found a local optimum around 5-6 states.

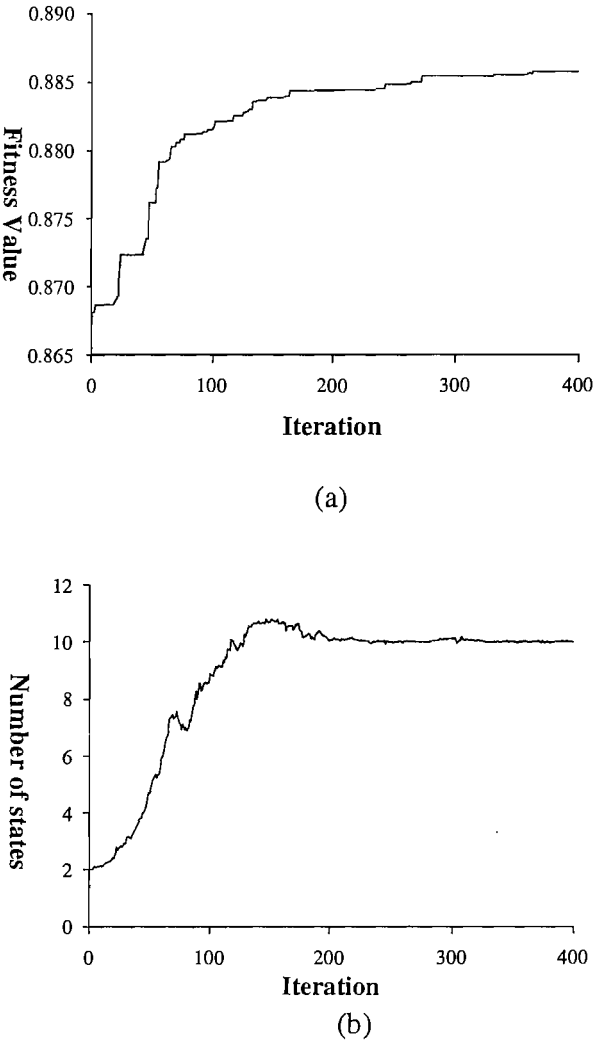


FIGURE 5.12: The simulation result of one of the cross-validation test with the first data set during GA-HMM training: (a) shows the fitness value of fittest individual on each iteration (b) shows average number of states for periodic signal. The GA started with a population consisting of 2 states. After 150 generations the HMM have a length of 10 states. Although the length does not significantly change thereafter the fitness continues to improve indicating that the finer structure is being fine tuned.

An example of one of the HMM structures found by the GA is shown in figure 5.13. The model is considerably more complex than that proposed by Petersen *et al.* shown in figure 5.10. Table 5.3 shows a comparison of the result for the test set $(-1/\log(P(x_{test}|\Theta)))$ and their variances obtained from the 5 cross-validation experiments for our GA-HMM. In the same table we show the results for the model proposed in [88] and trained using

Baum-Welch. The hand crafted model is trained only with the Baum-Welch algorithm. Of 175 sequences in their model, for each test 140 sequences are used for the Baum-Welch training and 35 sequences are used to calculate the fitness. Because the Baum-Welch algorithm finds a locally optimal solution, the final results depend on the initial values of the transition and emission probabilities.

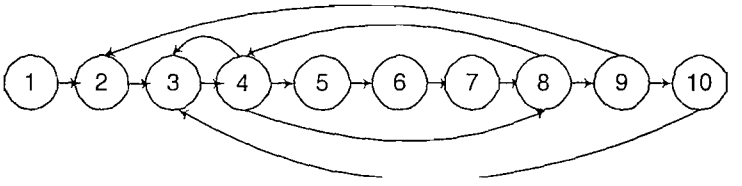


FIGURE 5.13: After training the *C. jejuni* sequences, GA-HMM found one structure model for the periodic signal.

TABLE 5.3: Comparison between Baum-Welch and GA-HMM. The results show the fitness value $(-1/\log(P(\boldsymbol{x}_{test}|\Theta)))$ and standard deviation for five different partitionings of the data.

Test	Petersen's model	GA-HMM
1	0.8855(1.7e-3)	0.8863(1.2e-3)
2	0.8854(1.0e-3)	0.8861(1.3e-3)
3	0.8887(2.0e-3)	0.8877(1.6e-3)
4	0.8796(0.7e-3)	0.8797(0.6e-3)
5	0.8713(1.4e-3)	0.8767(2.0e-3)

The performance of the HMM found using the GA and the hand designed HMM are roughly similar. Even though it does not show better performance in test 3, it produces slightly better result in the other tests. The paired t-test was conducted to assess the statistical difference [89]. This analysis compares the means of two groups to see if those two groups are statistically different. Let $X_{GA}(i)$ and $X_P(i)$ be the results for the GA-HMM and Petersen *et al.*. Then, the t-value for the k samples can be defined by

$$t - value = \frac{\bar{d}}{\sqrt{\sigma_d^2/k}} \tag{5.5}$$

where $\bar{d}$ is the mean of the differences $X_{GA}(i) - X_P(i)$, and σ_d is the standard deviation of this mean.

This statistic has $n-1$ degrees of freedom. In this experiment, the degrees of freedom is 4. With this result the significance level at which two distributions differ can be determined. The t-value obtained is 1.09. By using the t-value and the degrees of freedom the probability of null hypothesis can be obtained from the table of t-distribution. To obtain more precise probability we interpolate the values in the table of t-distribution.

The probability of this result, assuming the null hypothesis, is 0.336. From this test we had slightly better performance by using GA-HMM, but the improvement was not significant. However, this result shows the proposed method can replace a part of the hand-crafted model at least.

5.5.3 Simulation III: Comparison with Other Methods

5.5.3.1 The Effect of the Separation Scheme and the Selective Baum-Welch

In this chapter we introduced two methods to make the HMM suffer less from overfitting while evolving its structure. To see if how those methods improve the performance, we compared the results of each algorithm. We tested with *C. jejuni* promoter sequences. For this comparison we prepared another set of sequences with 139 training sequences and 34 testing sequences. We used the same configuration used in the simulation II. The algorithms we tested are 1) non-separation of training method 2) separation method 3) separation with selective Baum-Welch, and 4) hand-designed model. For each experiment we ran the test 30 times to get statistical information of the results. When the non-separation training method is used all the training sequences (139) are used for evaluation, while for the separation method 104 sequences are used for the Baum-Welch training and 35 sequences for the evaluation.

Figure 5.14 compares the fitness value graphs of the fittest member on each iteration. The graph of the non-separation method increases monotonically (figure 5.14(a)). The learning curve graph of separation scheme does not always increase. When the Baum-Welch training algorithm overfits the training set it receives penalty from the evaluation set and makes the learning curve fluctuate (figure 5.14(b)). The graph of the selective Baum-Welch method increases monotonically (figure 5.14(c)). The graph always increase since the fittest member is not trained and replaced only when other members of the population have higher fitness values.

Table 5.4 compares the results from each method. We calculated the average and the standard deviation of the fitness value of 30 HMMs independently trained by the GA-HMM. The fitness value is calculated against the 35 test sequences.

TABLE 5.4: The average and the standard deviation of fitness value of the 30 independently trained HMMs

	mean	standard deviation
non-separation	0.8837	1.5e-3
separation	0.8841	1.3e-3
selective Baum-Welch	0.8855	1.0e-3
hand-design	0.8772	2.3e-2

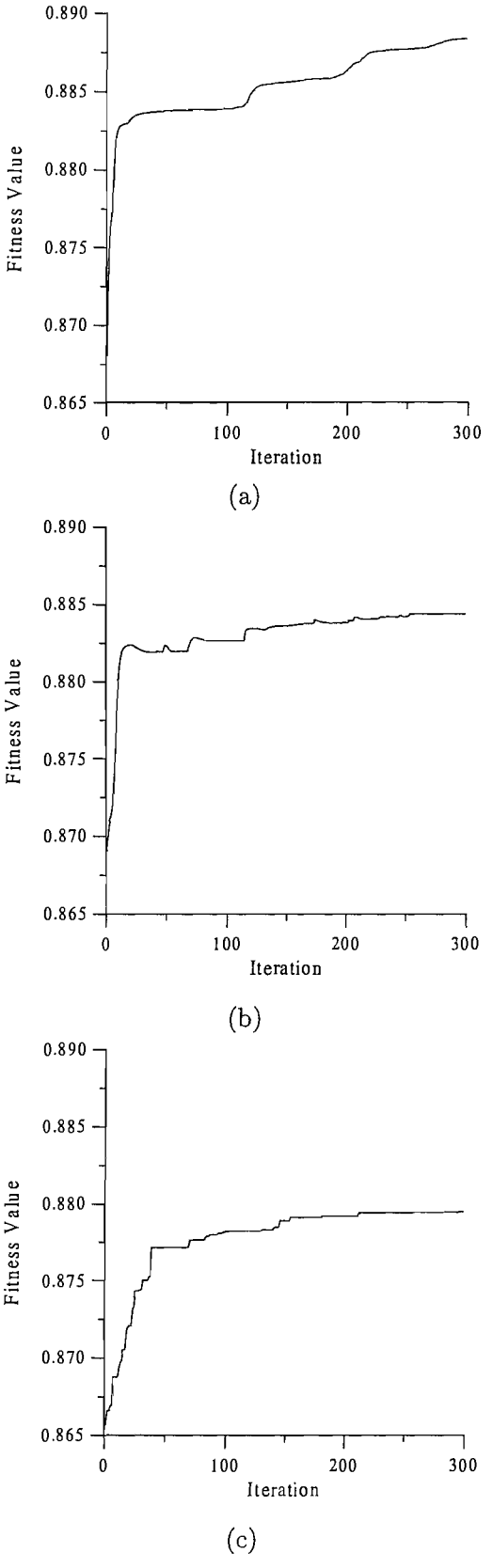


FIGURE 5.14: The learning curve graphs of each method (a)non-separation (b)separation (c) separation + selective Baum-Welch

We conducted the student's t-test [89] with the results. We have 58 (30+30-2) degree of freedom in this case. The probabilities of each comparison, assuming the null hypothesis, are in table 5.5. The probability shows the difference can happen by chance. Selective Baum-Welch is significantly superior to non-separation and separation schemes. We cannot be so certain about hand-designed because it has a large variance. Therefore the differences of hand-designing method and the other methods are not significant. Nevertheless, this test shows that evolving HMMs produces very useful models and can replace hand-designing method.

TABLE 5.5: The result of 6 t-tests. The probability that the difference can happen by chance is calculated.

method	probability
non-separation vs. separation	0.264
non-separation vs. selective BW	3.4e-6
separation vs. selective BW	5.5e-5
hand-design vs. non-separation	0.188
hand-design vs. separation	0.158
hand-design vs. selective BW	0.088

5.5.3.2 On the Balance Factor

Yada *et. al* used a balance factor in the fitness function to control the complexity and the likelihood. To see how the balance factor works we conducted the simulation with various balance factor values. We tested the similar fitness function as they used. The fitness function we used is

$$E_\mu = \frac{1}{-\sum_i \log\left(P(x_i|\Theta_\mu)\right)/l_i + \lambda p_k}, \tag{5.6}$$

where λ is a balance parameter and p_k is number of free parameters and in this case the number of states. Thomsen [5] used a similar fitness function. We checked how the balance factor works and compared it with our method. We changed the value of balance parameter to check how it affect the performance. For each configuration we conducted 10 experiments. Table 5.6 shows the result of the experiments.

There may exist a balance parameter that shows better performance than the result shown in table 5.6. In the simulation, however, we could not find any usefulness of the balance factor for this problem. Interestingly, a single state model shows a pretty good performance for this periodic signal model. Comparing the table 5.6 with table 5.4, we found the separation methods and the selective Baum-Welch method are superior to the method using a balance factor. Our separation method, which statistically balances the complexity and the likelihood, can be a good candidate to reduce the overfitting at least.

TABLE 5.6: The effect of the change of the balance factor.

balance parameter (λ)	mean	standard deviation	mean of number of states
10e-2	0.8837	5.4e-4	1.2
10e-3	0.8837	1.0e-3	3.6
10e-4	0.8827	1.4e-3	6.7
10e-5	0.8822	5.4e-3	8.1
10e-6	0.8812	3.9e-3	9.0
10e-7	0.8826	3.5e-3	9.4
10e-8	0.8819	5.9e-3	7.9
0	0.8837	1.3e-3	7.7

5.6 Discussion

The experiments described here suggest that genetic algorithms are quite capable of finding reasonable HMM architectures for biological sequence analysis. Even with a rather naive implementation, the GA was able to achieve comparable or slightly superior generalisation performance to a hand designed HMMs. Both experiments show that a major drawback of automating the design of HMM architectures is that the resulting model may be difficult to interpret biologically. Although we used genetic operators that favoured the building of chain structures they nevertheless allowed considerable cross linking within the chain. A drawback of constraining the search is that we may inhibit the GA from discovering completely novel types of architecture. One of the merits of GAs is the capability of dealing with substructures of the solution. In the GA-HMM the crossover operators can swap a series of states in one operation. The emission probabilities are also crossed over with the states. This enables the GA-HMM to swap any meaningful part of the HMM structure. In this sense, the proposed crossover scheme can treat such modularity, even though the strategy does not completely implement the modular structure.

We tried to prevent the GA from producing overfitted models by measuring the fitness on a different set of data from that used for training Baum-Welch. The improvement was not significant but it was a new trial that controls the size of an HMM statistically. We also selectively trained a proportion of the population. Those methods enable GA-HMM to evolve a new solution without suffering too much overfitting. From the comparison with the method using the balance factor, our method shows superiority for the given problem.

Our experiments were carried out on short sections of an HMM. Our preliminary results showed that it seems unlikely GA-HMM would be able to find large HMM structures that are competitive with hand designed architectures. Nevertheless, even in the short term GAs may be able to 'tune' a hand designed HMM especially in areas where the biological significance of a region is poorly understood.

The evolving scheme presented here is useful to find a simple HMM structure. For the modelling of more complex sequences the proposed method could not evolve an HMM to represent larger models. GA-HMMs usually find a smaller model than we may want to get. In some cases, we may want to get an interpretable model even though it deteriorates the performance. To evolve a large HMM efficiently we use blocks composed of HMM states. In the next chapter we will introduce a block method and new genetic operators.

Chapter 6

Block-HMMs (Block Hidden Markov Models)

6.1 Biological Block Model

The GA-HMM was used to learn topology of HMMs. It has proven that the separation of training sequences and the selective Baum-Welch have advantages in training HMM structures. However, the arbitrarily genetic operations leads easily to make complex models. To constrain the search of HMM topologies to biologically meaningful structures we represent the HMM structure as a number of blocks. The blocks we consider are one of three basic structures that are frequently used in biological sequence analysis. These are: linear, self-loop and forward blocks. The self-loop and forward block can be either tied (we follow the convention of shading tied blocks) or untied. That is, all the emission and transition probabilities are set equal. In the case of linear blocks we did not consider tying because tying a linear blocks seemed unrealistic for biological sequences and can be replaced with a single-state self-loop block. Examples of these block structures are illustrated in figure 6.1.

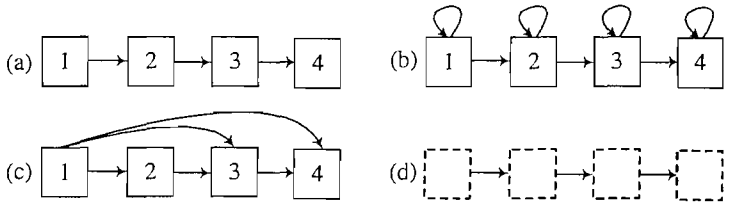


FIGURE 6.1: HMM blocks that compose the whole HMM structure: (a) linear block (b) self-loop block (tying is optional) (c) forward-jump block (tying is optional) (d) zero block.

Linear blocks consist of N states (labelled from 1 to N) where state n is only connected to state $n+1$ (with $1 \leq n < N$). Self-loop blocks are linear blocks in which each state

has an additional loop to itself. A forward block is a linear block where the first state is also connected to the last M states (with $1 <= M < N$). Zero blocks are empty blocks with no states: they can replace other block types during the GA procedure and thus allow the exploration of simpler topologies.

The block models described in this thesis are motivated by applications of HMMs in biological sequence analysis. Biological sequences (DNA or protein) often contain 'motifs', which are more or less conserved words, and with more or less homogeneous intervening sequence, which is characterised by the composition of letters (amino acids or nucleotides). The motifs might for instance correspond to binding sites for other molecules. Such a sequence can be modelled by an HMM containing submodels for the motifs (linear chains of states) and models for the intervening sequences, each of which can be a single state or multiple emmission tied states, if a length distribution is modelled. Other types of sequences are changing between various types of homogeneous sequences. An example is membrane proteins that contain membrane helices 20–30 amino acids long, which are dominated by hydrophobic amino acids and intervening sequence that is typically more hydrophillic [21]. Such sequences can be modelled with a block of tied states, one block for each type of sequence. Sometimes sequences contain periodic patterns. The region of a gene that codes for a protein is made up of codons, which are nucleotide triplets, each of which codes for one amino acids. The first codon is known as a start codon (often the three bases ATG) and the last codon (which actually does not code for an amino acid) is a stop codon (TAA, TGA or TAG). This gives rise to a three-periodic pattern, which can be modelled as previously shown in figure 5.5.

6.2 Genetic Operators for the Block-HMM

Blocks are fully linked together to form the whole HMM architecture. This is illustrated in figure 6.2. The final state of a block has transitions to the first states of all the blocks. The resulting HMM can be thought of as a fully connected graph consisting of 'supervertices' made up of blocks whose internal states are not fully connected. We call this structure a Block-HMM. Special patterns like periodic signals can be generated with a path between blocks.

Each block is represented by a pair. The first element defines the length of the block while the second element gives the type (a, b or c corresponding to linear, self-loop or forward-jump block respectively). The type also specifies whether the nodes are tied or untied (t or u) and in the case of forward blocks the number of forward connections. The full HMM is represented as a string of pairs as shown in figure 6.3. For example, the HMM in figure 6.2 would be represented by '((3,a),(2,bu),(3,ct1))'. As the blocks are equivalent in their connectivity there is no information in the ordering of the blocks.

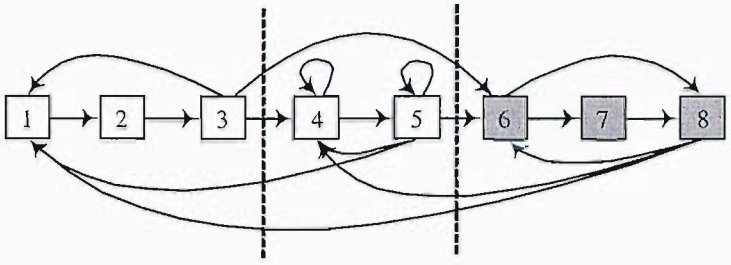


FIGURE 6.2: An example of HMM composed of blocks. Three blocks are used in this model and all the blocks are fully connected to each other.

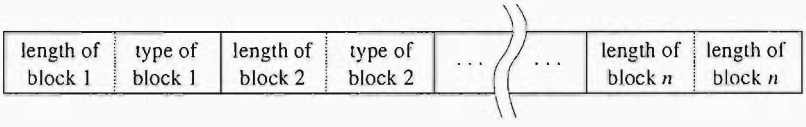


FIGURE 6.3: The string representation of a Block-HMM. The information on the lengths and the types of the blocks are stored.

To evolve the Block-HMM without losing block property we designed new genetic operators. In crossover, two parent strings are chosen at random. Some number of randomly chosen blocks are then swapped to create two children. The children then replace the parents. When we swap blocks the transition probabilities leaving the block are kept unchanged. Since the position of the blocks does not carry any meaning, we do not impose any constraint on which blocks are swapped. The number of blocks is kept fixed, however, as the blocks can have variable lengths, the number of states is not fixed. We also allow blocks consisting of no states (zero blocks), which effectively allows us to have a variable number of blocks up to some maximum. The evolution of a variable size structure is similar to the situation common in many applications of Genetic Programming. This scheme in a way emulates natural evolution which can cross over DNA sequences with different lengths. We chose a generational GA as a way to present the DNA sequences crossed over as a block.

Figure 6.4 shows an example of the crossover scheme. The last block of the first child crosses over with the first block of the second child. To simplify the diagram, transitions between blocks are not shown here. Under the crossover scheme the properties of the blocks are not broken. This allows us to exchange meaningful blocks without causing too much disruption.

Mutations can take place in any block of the HMM. There are a variety of different mutations that we allow. Mutations can change the length of a block. Forward-jump block mutations can change the number of transitions. For example, in the case of a 5-state forward-jump block, there are 6 different types of mutations possible. These are illustrated in figure 6.5. The mutation can add (figure 6.5(a)) or delete (figure 6.5(b))

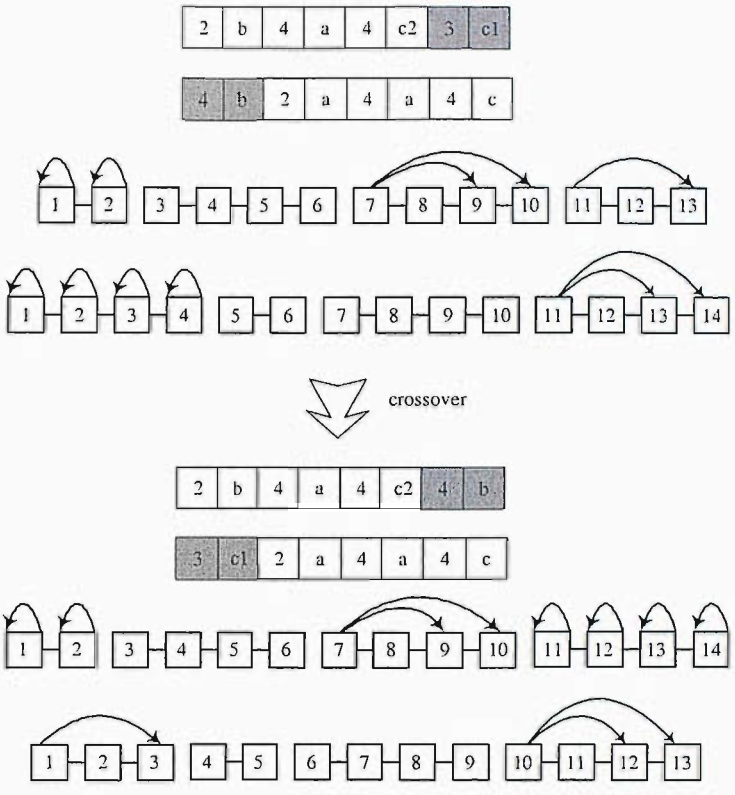


FIGURE 6.4: Crossover in Block-HMMs. The crossover swaps the HMM states without breaking the property of HMM blocks

a transition inside a block. To prevent losing the property of the block, deletion of a transition is not allowed when there is only two transitions (to the second state and the last state). In the same way adding transition does not take place when the first state of the block has transition to all the other transition. In figure 6.5(c) and figure 6.5(d) a state is deleted from a block. The outcome depends on which block is deleted. In figure 6.5(e) and figure 6.5(f) a state is added to the block. Again the outcome depends on which block is added. In the cases of linear and self loop blocks, there is only one way to add and delete a state. These six different types of mutation supply the Block-HMM with sufficient variation without changing the properties of the block.

In addition to changing the length of the block and the transitions, we also allow mutations that change the type of the block. For self-loop and forward jump blocks, we can mutate between tied and untied versions. We can also mutate the type altogether. Mutations from and to a zero block are also allowed (figure 6.6).

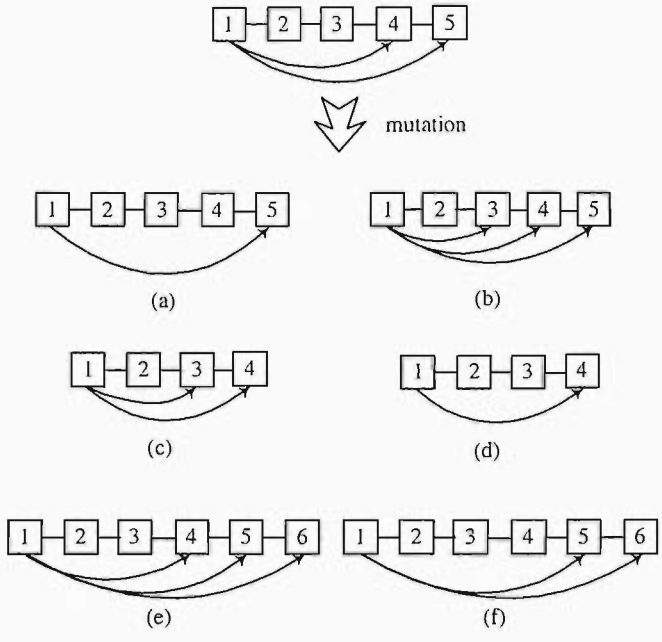


FIGURE 6.5: Six possible types of mutations from a 5-state jump forward block: (a) a transition from the first to the fourth state is deleted (b) a transition from the first to the third state is added (c) the second or the third state is deleted (d) the fourth state is deleted (e) a state is added between the fourth and the fifth state (f) a state is added between the first and the fourth state.

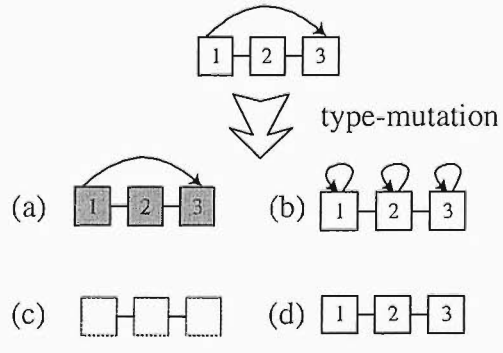


FIGURE 6.6: Type-mutations: (a) to a tied block (b) to a self loop block (c) to a zero block (d) to a linear block.

6.3 Training Procedure

The number of blocks is chosen at the beginning of the training and kept fixed. The length and type of blocks was randomly chosen so that there were on average the same number of linear, self-loop, forward-jump and zero blocks.

To test our HMM we split our data into a training and a test set. The training data was further split into a Baum-Welch training set and a evaluation set, as describe in chapter 5. We take as fitness values the reciprocal of the negative log-likelihood of the evaluation data set.

$$E_\mu = \frac{1}{-\sum_i \log\left(P(x_i|\Theta_\mu)\right)/l_i} \tag{6.1}$$

where l_i is the length of a sequence x_i and μ labels the different HMMs (with parameters Θ_μ) of the population. A member of the population is selected with a Boltzmann probability

$$F_\mu = \frac{w_\mu}{\sum_{\nu=1}^{N} w_\nu}, \qquad w_\mu = \mathrm{e}^{s\,E_\mu/\sigma} \tag{6.2}$$

where σ is the standard deviation in the distribution of fitnesses in the population. The parameter s controls the selection strength. Stochastic universal sampling is used to reduce genetic drift in selection [65].

At each generation we applied one iteration of the Baum-Welch training algorithm. We then evaluated the fitness of the population and performed selection (with selection strength $s = 1$), mutation and crossover. Mutation and crossover are applied with a small probability. We continued until there was no significant change in the structure of the HMM.

6.4 Experiments on Block-HMMs

6.4.1 Experiment with Artificial Data

To test the performance of the Block-HMM, we conducted three experiments with artificial data. The first two experiments were to find an HMM to represent data generated from the languages $(\mathrm{ATG})^+$ and $(\mathrm{AAGATGAGGACG})^+$ where '+' means any number of repetitions. We used a population composed of HMMs with 2 blocks. Table 6.1 shows parameters used in these toy experiments. That is, at each generation we choose two individuals and perform crossover to create two children which replace the two parent strings, one individual where we mutate either the length or the number of transitions in a randomly chosen block and one individual where we mutate the type of a randomly chosen block. For this simple problem we did not allow tying. The solutions to these two problems found by the GA are illustrated in figure 6.7. The resulting HMMs are

reasonable solutions to the problem although probably not those that a human would have come up with.

TABLE 6.1: Block-HMM parameters used in the experiment.

Parameter	value
Population size	20
Iteration	300
Crossover rate	0.05
Mutation rate	0.05
Type mutation rate	0.05

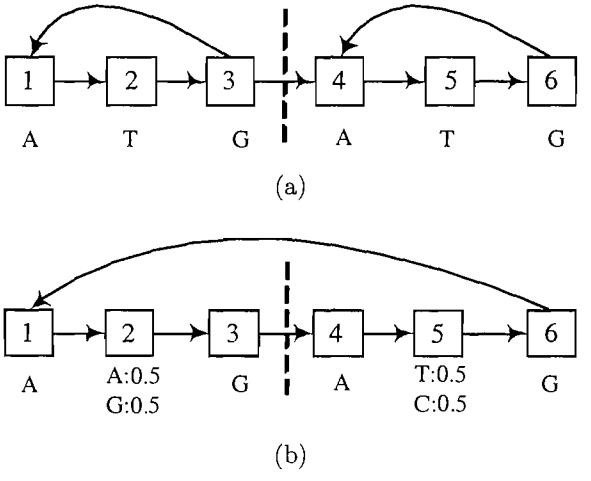


FIGURE 6.7: The result of Block-HMM with 2 blocks. (a) $(ATG)^+$ (b) $(AAGATGAGGACG)^+$

The third test we carried out was to find an HMM to recognise the language $(AAGATGAGGACG)^+(ATGC)^+$. The first half of the sequence is the same as that used in the previous experiment, while the second half is a repetition of four symbols. The number of iteration used in this experiment is 600. Figure 6.8 shows a graph of the maximum log-likelihood value versus the iteration for four different runs of the GA. In case 3 of the three block model, the GA has still not found a particularly good HMM after 600 iterations.

The best solutions found by the GA in the four runs are shown in figure 6.9. Note that we have not shown transitions which the Baum-Welch algorithm has driven to zero. In the first run (figure 6.9(a)) we initiated the GA with two blocks, while in the next three runs (figure 6.9 (b),(c) and (d)) we used a three block model.

These toy examples show that a GA is capable of finding reasonable structures for the given problem. However, the models are not the simplest that could solve the problem nor do they always have an optimal structure. Nevertheless, they demonstrate that the GA can find reasonably parsimonious solutions which give a good approximation of the

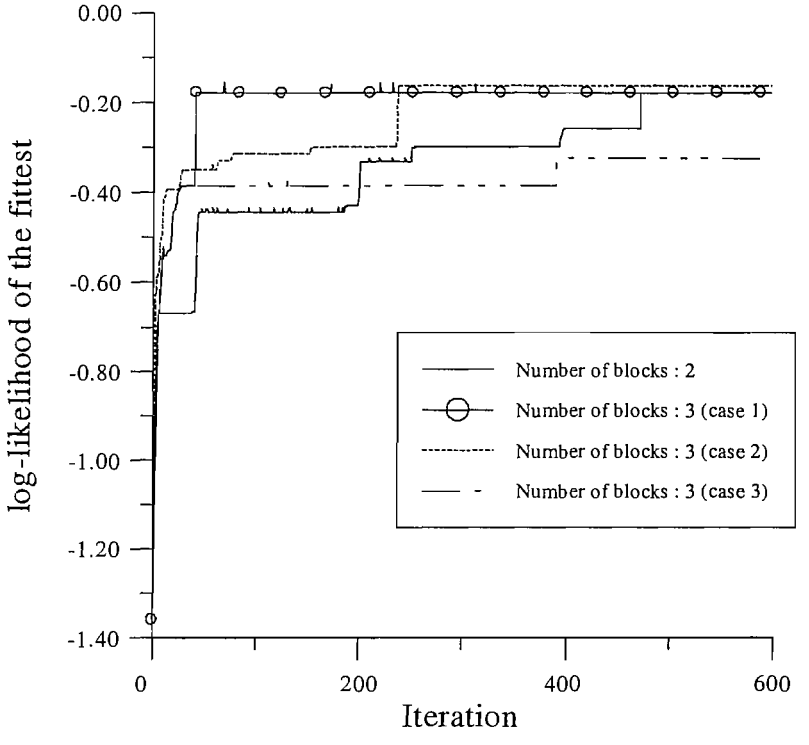


FIGURE 6.8: The behaviour of the Block-HMM is shown as a function of the iteration for 4 runs.

true likelihood. To test the performance of the Block-HMM on more complex sequences, we used biological sequences in the following experiment.

6.4.2 Coding Region Model of *C. jejuni*

To investigate the Block-HMM's ability to find an HMM structure for biological sequences, we performed an experiment with 200 sequences from the coding regions of *C. jejuni*. Figure 5.5 shows a typical HMM that could be used to represent the coding region [90]. Of the 200 sequences available, 150 sequences are used for training and 50 sequences for evaluation. From looking at the data alone it is almost impossible for non-specialists to see that the data consists of codons.

We conducted the experiment twice, once using four blocks and once with three blocks. The initial lengths of the range between 3 to 7. The GA parameters used in the simulations are shown in table 6.2.

Figure 6.10 shows the resulting structures of the Block-HMM found using the GA. In figure 6.10(a) the second state of the first block is not used. In figure 6.10(b) the emission probabilities of the state between 7 and 15 are tied. Detailed transition and emission probabilities are available from the web page mentioned at the end of this chapter.

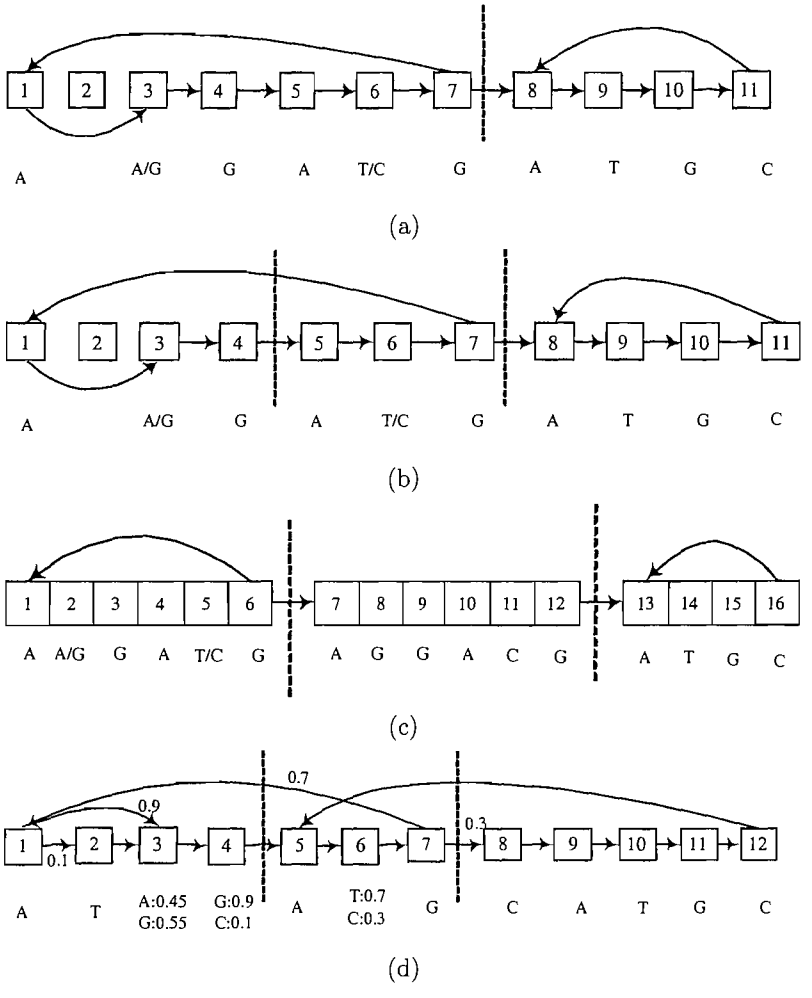
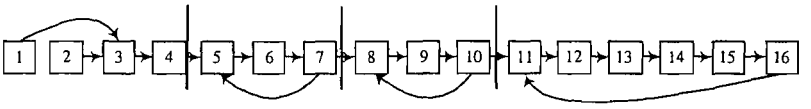


FIGURE 6.9: The result of Block-HMM for the $(AAGATGAGGACG)^{+}(ATGC)^{+}$ (a) with 2 blocks (b) with 3 blocks (case 1) (c) with 3 blocks (case 2) (d) with 3 blocks (case 3).

TABLE 6.2: Block-HMM parameters used in the experiment with biological sequences.

Parameter	value
Population size	30
Iteration	600
Crossover rate	0.13
Mutations rate	0.13
Type mutation rate	0.13

Although these are not the most parsimonious solutions they do seem to have identified the triplet nature of the sequences.



(a)

(b)

FIGURE 6.10: The result of Block-HMM. It searched the 3 state loops with GAs.

6.4.3 Promoter Model of *C. jejuni*

We conducted experiments to investigate whether the Block-HMM can find the conserved patterns of *C. jejuni* promoters. A hand designed HMM structure is shown in figure 5.9. In this experiment we investigated if Block-HMM can find a whole HMM structure instead of finding a part of an HMM structure.

The sequence contains more Ts and As than Gs and Cs. Although the Ribosomal binding sites can be easily found, the sequence is not always in the same position in relation to the coding region. In some cases, the sequence AAGGA doesn't appear or is highly mutated. It is difficult for non-experts to figure out which part of the sequence is the TATA box because most part of the promoter region is composed of Ts and As.

The GA parameter are the same as those used in the previous experiment. Of 175 sequences available, 132 sequences are used for Baum-Welch training and 43 sequences are used for the fitness evaluation. We conducted simulations using 9, 8 and 7 blocks. To obtain the result showing conserved regions in order, we limited the transition during the crossover. Crossover cuts the parents HMM into two or three large blocks. The large blocks are relocated in order allowing one transition to the next block.

The best structures found by the GA are shown in figure 6.11. The Block-HMM could find the 'AAGGA' and 'TAtAAT' regions with 9 blocks (figure 6.11 (a)) and 8 blocks (figure 6.11 (b)). In addition, it found the presence of semi-conserved TGx upstream of TATA box which is characteristic of the *C. jejuni* promoter region [88]. However, when the number of blocks is 7 (figure 6.11 (c)), the Block-HMM could find only AAGGA sequence. The TATA box was buried inside other states. Interestingly, the 9 block model (figure 6.11 (a)) also found the 10-base periodicity just before the TATA box, which was discovered in [88]. In figure 6.11, the emission probabilities of the states in

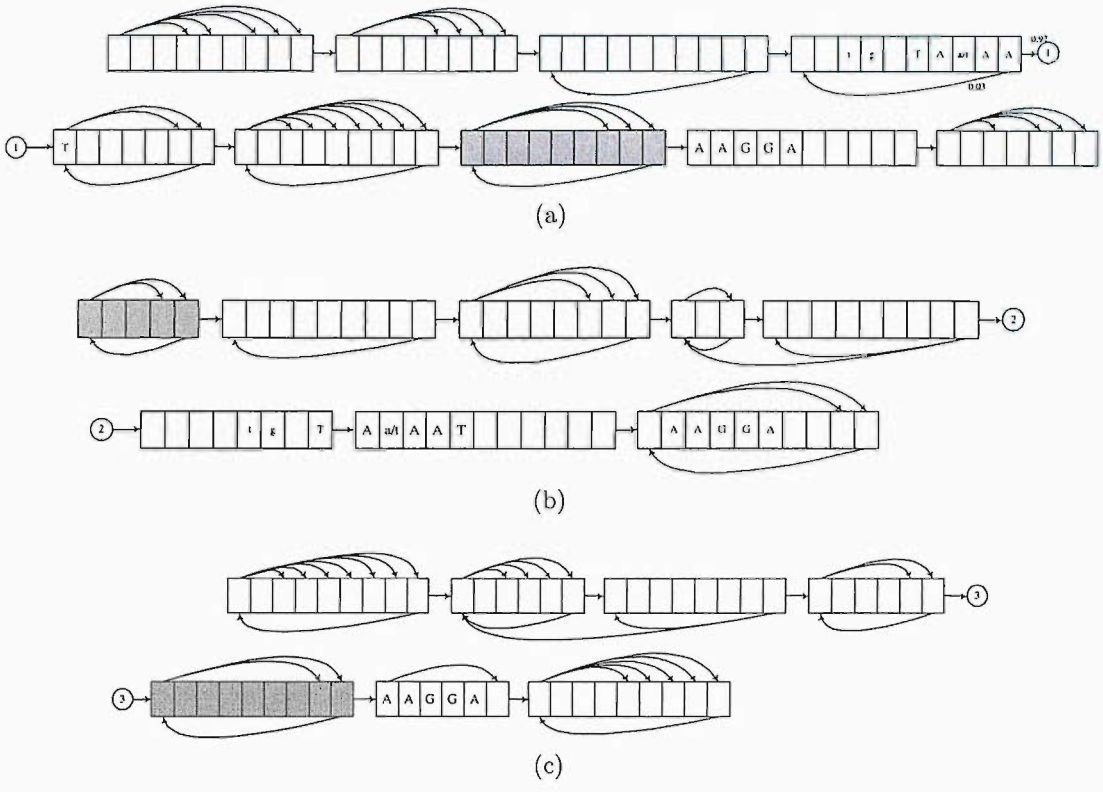


FIGURE 6.11: The best structures found by the GA for (a) 9, (b) 8 and (c) 7 blocks. The 'AAGGA' sequence is found on every simulation and 'TAtAAT' sequence is found in (a) and (b). Each cell represents a state. Emissions of shaded cells are tied.

the shaded cells are tied.

To perform a quantitative test of the generalisation performance of structures found in the previous experiment we conducted a discrimination test. In order to collect a sufficient amount of data we carried out a five-fold cross-validation experiment. This replicates a test performed by Petersen *et al.* [88]. As the structures generated in the previous test (shown in figure 6.11) were found using all the training data, we retrained all the emission and transition probabilities starting from random values. However, to retain the main structure we removed all transition probabilities less than 0.002 and to maintain the positions of TATA box and Ribosomal binding site we introduced pseudo-counts in these conserved regions. For example, if the emission probabilities of one state is A:0.9 T:0.1 G:0.05 C:0.05 then the pseudo-count becomes 90, 10, 5, 5 for the A,T,G,C respectively. This ensures that these conserved regions occur in the same place when we retrain the model in the cross-validation procedure. We used Baum-Welch to retrain the weights using 140 out of the 175 sequences as training data. The remaining 35 sequences were used as test data.

To perform a discrimination test we require a background sequence. To obtain this, we use a 500 000bp sequence generated by a third-order Markov chain that had been

trained on the *C. jejuni* genome. To test the discrimination we set a log-odds threshold so that there were 10 or less false positives and then measured the number of true positives. Table 6.3 shows the total number of false positives (FP) and true positives (TP) summed over all of the five-fold cross-validation tests.

TABLE 6.3: The result of the Block-HMM

Simulation	Number of FP	Number of TP	TP rate
Block-HMM (9 blocks)	7	132	75%
Block-HMM (8 blocks)	9	126	72%
Block-HMM (7 blocks)	10	120	69%
Petersen's HMM	10 (best case)	119	68%

We repeated this test on ten different HMM structures found by using a GA. In all but one case, we could get 7∼9 false positives and a true positives rate ranging from 70% to 76%. Only on one simulation out of the ten, did the GA fail to find a TATA box. These results are superior to those published for a hand-crafted HMM [88], although we must be slightly cautious in interpreting our results as the structures were found using the same data that it was tested on. We were forced to do this because the dataset was small. We did take the precaution of retraining all the weights to reduce the risk of biasing our results, but the pseudo-counts were influenced by the whole data set. Although, the results are not conclusive, they are very suggestive.

6.4.4 Discussions

In this chapter, we have described a GA which evolves the structure of HMMs in a biologically constrained way. We have performed a number of tests to illustrate that the resulting HMMs are relatively easy to interpret. Furthermore on the problem of the promoter model of *C. jejuni* the results are competitive with an expert designed HMM. This is quite remarkable given that our GA had no prior knowledge of conserved regions such as the TATA box and Ribosome binding site as well as other structures which have been acquired by experts over many years. These are promising results from early work.

In order to reduce overfitting we have split our training data into a Baum-Welch training set and an evaluation set used in selecting members of the population. Although we could potentially overfit on the evaluation set, this does not seem to be a problem in practice. One explanation for this is that once a trained HMM overfits the training data when we run the Baum-Welch algorithm, they then perform badly on the fitness evaluation set. Since we use Baum-Welch at every generation, overfitted models will always be disadvantaged.

The blocks introduced in this study were adopted from the frequently used HMM topology. Beside those three block other topologies have been constructed in bioinformatics

depending on the applications [18]. And other possible block models can be addressed. We believe, however, for biological sequence model those 3 block models can be used as building blocks that organize the whole topology.

The crossover operator proposed is one of the possible methods that use the modularity property of HMMs. It enriches the search domain by increasing the number of possible solutions. On the other hand, the unconstrained choice of blocks for the crossover can lead the whole process to be disruptive. Therefore, as the number of the blocks increases the Block-HMM requires more iterations to converge. A more strategic method of crossover might improve the efficiency of the process.

Evolving HMM structures is a time consuming process since training and evaluating the likelihoods for long sequences takes many operations. At present, this is a restriction on automatic structure finding. However, as computers get faster and the number of applications grow we expect that automatic HMM structure search will become increasingly important. GAs seem ideally placed to play an important part in this development.

Chapter 7

Block-HMMs for the Prediction of Proteins Secondary Structure

7.1 Proteins Secondary Structure Prediction

The 3D structure of proteins is very important to understand their function. The secondary structure of proteins determines the conformation of 3D structure. Predicting the secondary structure of proteins has become one of the most studied problems in bioinformatics. The problem tackled is to provide a label for each residue in a protein sequence depending on its secondary structure. That is, whether the protein residue is part of an alpha-helix, a beta-sheet or some other structure. This is a first step towards predicting the structure and function of a protein from its sequence.

There are several publicly available predictors. However, direct comparison between the secondary structure predictors is difficult because there is not a standard test set. Currently, the EVA server [91] benchmarks those secondary structure predictors developed so far. It automatically analyses 19 protein secondary structure prediction servers continuously.

Our approach to the secondary structure prediction problem is to evolve an HMM using a GA. Even though HMMs have been successfully applied to many problems in biological sequence modelling, they have not been very successful as protein secondary structure predictors. This is partly due to the difficulties in modelling the complex nature of protein structures through the topology of an HMM. The predictors using HMMs show a bit lower performances than the predictors using NNs. Thomsen studied the same problem of secondary structure predication as we have [5]. His prediction rate on the training set using the standards Q_3 measure is 49%. HMMSTR [6] is the hand-designed HMM model and its prediction rate is 74.3% in the cross-validation test.

In this chapter we use the evolutionary method to optimise the structure of an HMM for secondary structure prediction. During the evolutionary optimisation, the HMM's structure is built up using biologically meaningful building blocks. Three independently trained Block-HMMs were cascaded by a second layer of neural networks. To improve the prediction rate homologous sequences are used. We tested the trained HMMs with a common subset of data from the EVA server [91].

7.2 Methods

7.2.1 Dataset

The SABMark Twilight Zone data set (version 1.63) [92] provides a set of representative structures. This data set consists of 2230 high quality structures partitioned into 236 folds. Although many proteins in the data set share a common fold, no pair of protein sequences can be aligned with a BLAST E-value below 1 or a sequence similarity above 25%. For the proteins with a common fold in the data set, it is not possible to identify a traceable evolutionary common origin.

Structures that caused problems with the DSSP program (see below) or that had chain breaks were removed, which resulted in a final data set of 1662 structures belonging to 234 fold groups (two groups are removed by this process). With these 234 groups we performed a five fold cross-validation test. In order to create a stringent test set we made sure that proteins with a common fold do not appear in both the training and test sets.

The secondary structure was calculated using the program DSSP [43]. DSSP assigns secondary structure to eight different classes: α-helix (H), isolated β-bridge (B), β-strand (E), 3_{10}-helix (G), Π-helix (I), turn (T), bend (S) and other. The DSSP results were retrieved using the DSSP front end in the Biopython toolkit [93]. Like most other prediction methods we used a reduction scheme whereby H and G are converted to H, E and B are converted to E, and all the other to C.

7.2.2 Block-HMMs for Labelled Sequences

Block-HMMs restrict their search to a subset of topologies made up of blocks of states. Each block is assigned with a label that corresponds to one of the three secondary structure classes. The states that make up the blocks emit amino acid symbols. Secondary structure prediction is done by inferring the values of the hidden states for a given amino acid sequence, and examining the secondary structure labels of the blocks these states belongs to. The blocks were chosen to be biologically meaningful structures. Four types

of blocks are used: linear, self-loop, forward blocks and zero blocks. Initially, the blocks are fully linked to form HMM architectures as shown in figure 7.1. To deal with labelled sequences, every state in the same block share the same label.

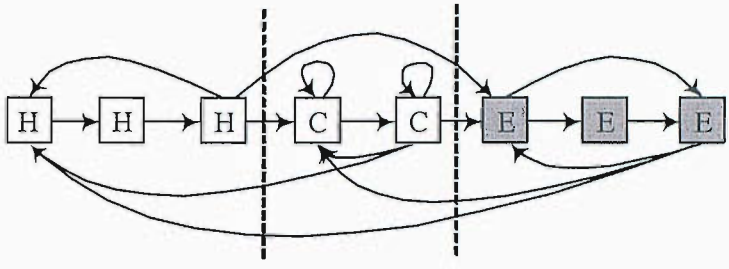


FIGURE 7.1: An example of an HMM composed of blocks resulting from the Block--HMM procedure. Three blocks are used in this model and all the blocks are fully connected to each other. The blocks are divided by dotted lines. The states in tied blocks are shaded in grey.

The various blocks can model different types of sequence fragments. A linear block can model a particular conserved sequence pattern. The self-loop block can model a sequence of any length while the forward jump block can be used to represent subsequences with varying length up to some fixed length.

7.2.3 Genetic Operators for Block-HMM

We used three genetic operators in Block-HMM: crossover, mutation and type-mutation. The number of blocks is kept fixed but the number of the states of an HMM can be changed by the genetic operators. Crossover and mutation work in a similar way we presented in the previous chapter. When a block increases its size during mutation the added state has the same label with the block it belongs to.

We designed a new type-mutation to handle the label on a state. The new type-mutation changes the type or the label of a block. Type-mutations from and to a zero block are also allowed. Figure 7.2 illustrates a type-mutation from a 3-state forward length block. When a type mutation transforms the type of a block, new transition probabilities are generated randomly. Self-loop and forward jump blocks can type-mutate between tied and untied versions. The labels are also changed under the type–mutation. When type-mutating from a zero-block, a new block is created and replaces the zero-block. The size of the block as well as all the parameters of the states in the block are assigned randomly.

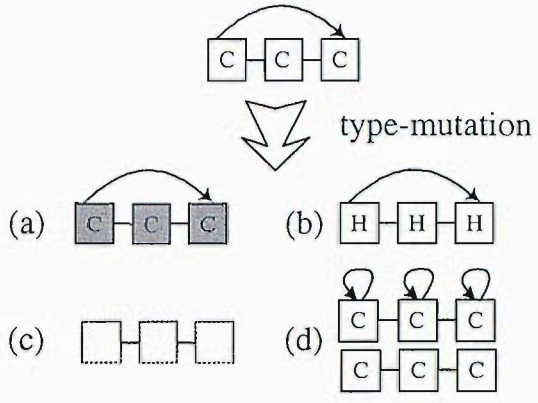


FIGURE 7.2: Type-mutations: (a) to a tied block (b) to a block with a different label (c) to a zero block (d) to a self loop block or a linear block. When a type mutation transforms the type of a block, new transition probabilities are generated randomly.

7.2.4 Parallel Genetic Algorithms

Evolving HMMs particular for this application is highly CPU-intensive. To overcome this we used a Parallel Genetic Algorithms (PGAs) [67] run on a cluster of computers. From the algorithmic point of view the parallel processing is the computational realisation of the natural parallel evolutionary strategy.

We used a master-slave model to implement the Block-HMM on the clustered computers. The master-slave model using P slaves are illustrated in figure 7.3. In the master-slave model, a population is generated on P processors. The time consuming training and evaluation procedures are dealt on slave-processors. Since the computational time for the training and evaluation is different depending on the size of HMM, the master waits until all the processes on the slaves finish. The slaves send the fitness value and the trained HMM to the master. Then the master applies the genetic operators on the individuals and sends the individual information back to each slave-processor. On the slave side the processor waits for message from the master, evaluates the fitness, trains the HMM, and returns the trained HMM and fitness value to the master.

7.2.5 Training with Block-HMM

We have used a hybrid GA with traditional GA operators to explore the space of HMM topologies in combination with Baum-Welch optimisation of the transition and emission probabilities.

To obtain suitable HMM architectures we tested various numbers of blocks between 26 and 35. Labels are allocated randomly to each of the blocks. The size of the block (number of states in a block) is randomly assigned between 1 and 4. Table 7.1 shows parameters used in the simulation.

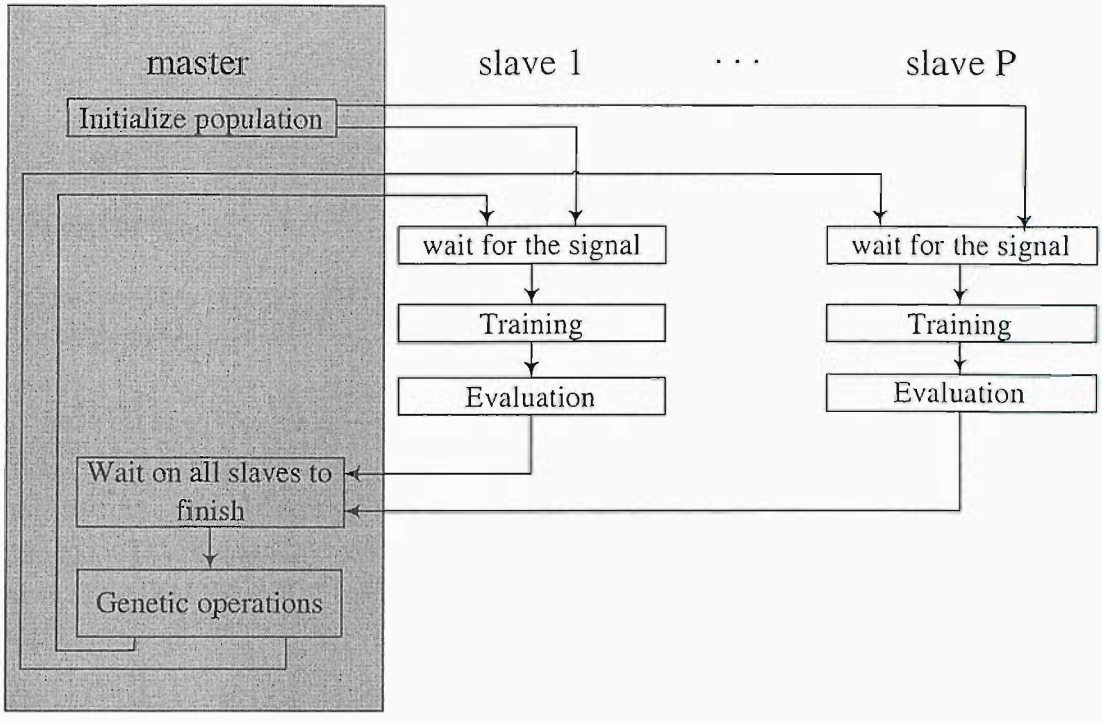


FIGURE 7.3: The flowchart of master-slave model of parallel genetic algorithms. We used P processors to implement this master-slave model.

TABLE 7.1: Block-HMM parameters used in the experiment.

Parameter	value
Population size	30
Iteration	400
Number of blocks in an HMM	26–35
The initial length of a block	1–7
Crossover rate	0.07
Mutation rate	0.07
Type-mutation rate	0.07

To find an HMM that does not overfit the training data, we divided our training set into a set used for the Baum-Welch training (5/7 of the data) and a set for fitness evaluation (2/7 of the data). The fitness value is calculated from the fitness evaluation set only. Given an HMM (Θ), we take the reciprocal of the negative log-likelihood as the fitness value:

$$E_\mu = \frac{1}{-\sum_i \log\left(P(x_i|\Theta_\mu)\right)/l_i} \tag{7.1}$$

where l_i is the length of a sequence x_i and μ labels the different HMMs (with parameters Θ_μ) of the population. A member of the population is selected with a Boltzmann

probability

$$F_\mu = \frac{m_\mu}{\sum_{\nu=1}^{N} m_\nu}, \qquad m_\mu = e^{s\,E_\mu/\sigma} \tag{7.2}$$

where σ is the standard deviation of the fitness in the population and s is a constant that controls the strength of the selection. In the work reported here, we used a value of s equal to 0.3.

The best member of a population is always selected, and a subset of other members are selected by using stochastic universal sampling [65]. Some of the members are mutated or subjected to crossover. Then, all the members of the generation undergo Baum-Welch optimisation using the training data set. We saved the best HMM at each of the 400 generations, *i.e.* during the whole run of the GA. At the end of the run, the best HMM is selected and trained again with the Baum-Welch algorithm, this time using all the sequences. This is done because the last HMM is not always the best HMM generated during the whole GA run. Finally, the HMM is trained further using the discriminative training method [42].

Figure 7.4 shows one of the results of Block-HMM. The simulation used 26 blocks but the result shows only 19 blocks: the remaining 7 were zero blocks. The full HMM structure with 42 states is found on the web site. The full HMM structure with 42 states is illustrated in figure 7.5. Transitions with a probability less then 0.1 are not shown in this figure.

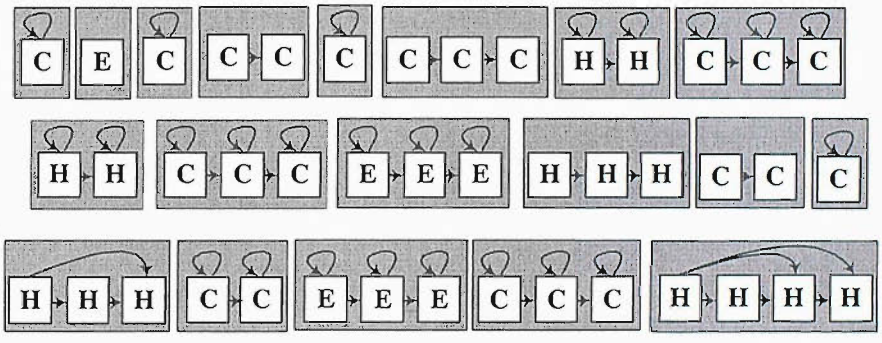


FIGURE 7.4: An example of an HMM evolved using Block-HMM. It is composed of 19 non-zero blocks and 42 states. Transitions between blocks are not shown here (including the transition from a block to itself).

7.2.6 Incorporating Evolutionary Information

Secondary structure prediction rates can be boosted by using evolutionary information. In most systems, the position specific scoring matrix (PSSM) is used as an input of the predictor. Instead of using PSSM, we ran our predictor on a set of homologous sequences and then combined the results. To obtain the homologous sequences we ran PSI-BLAST

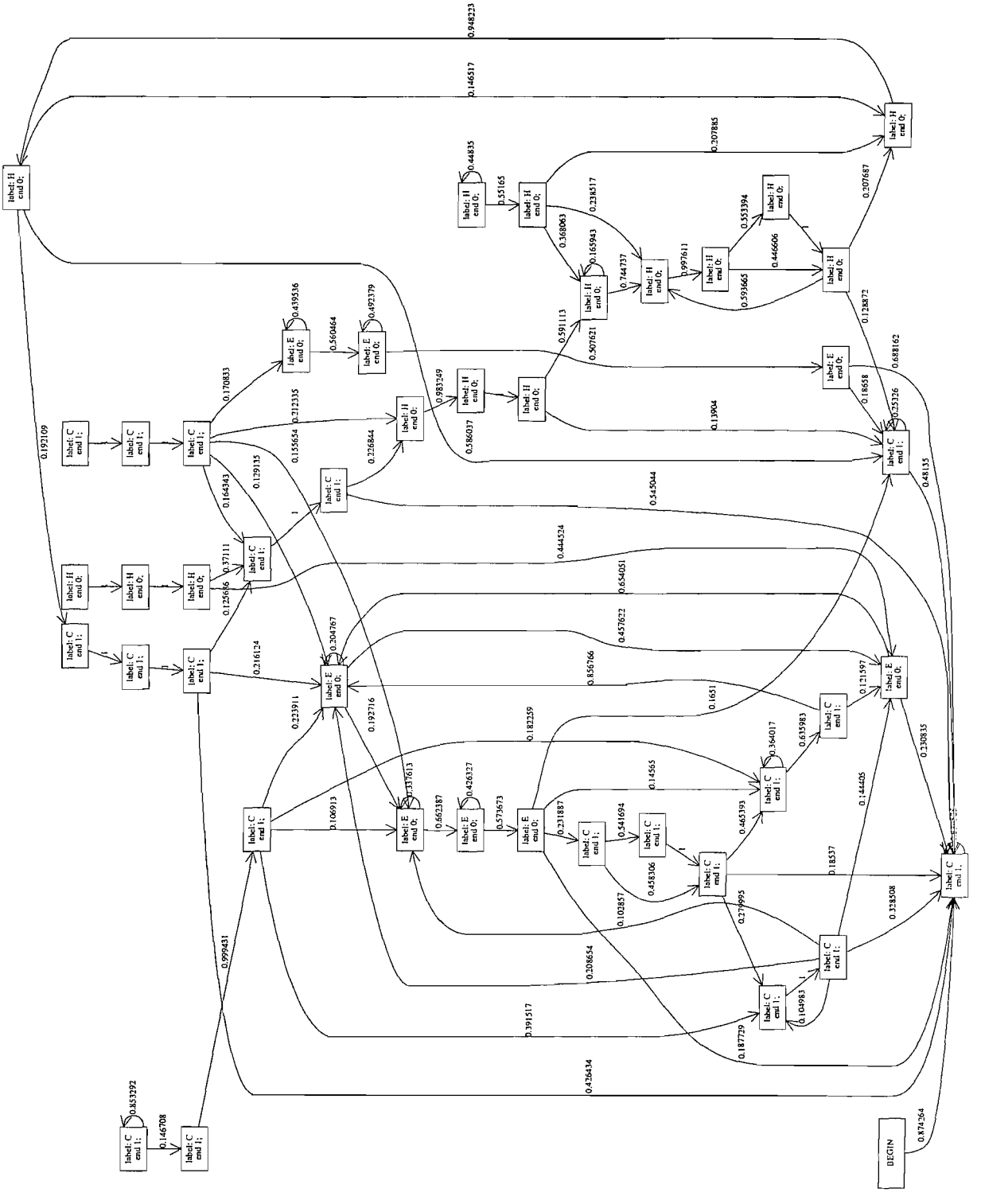


FIGURE 7.5: The full HMM structure with 42 states. Transitions less then 0.1 are not shown. This figure is drawn with the drawing tool provided by L.G.T. Joergensen.

[51] against the UniProt 90 protein sequence database [57] downloaded on 17th of Feb. 2005. We used 3 iterations of PSI-BLAST with an E value threshold of 0.001. The posterior label probabilities (PLPs) were calculated by decoding each of the homologous sequences against the trained HMM. The gaps are ignored when the sequences are decoded. There are redundant amino acids needed for the pairwise alignment when a query sequence is shorter than the homologous sequences. After decoding the sequences are aligned again without redundant amino acids. In figure 7.6 the second amino acid of the second homologous sequence, 'K', is redundant and dropped when the sequences are aligned again. The PLPs of each sequence are aligned along with the sequence. If there are gaps in the sequence, the gapped PLP is set to 0 for all labels at all positions with the gap.

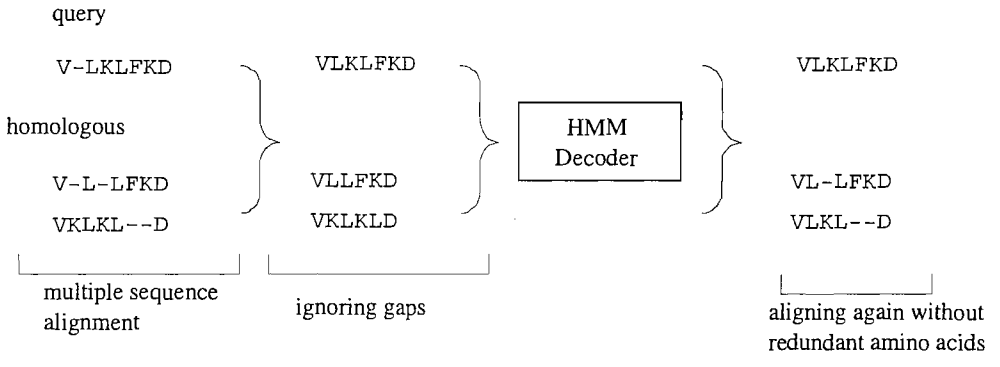


FIGURE 7.6: Decoding and aligning homologous sequences. Gaps ignored during decoding and redundant amino acids are deleted after decoding.

After aligning the decoding results, we calculated the weight of each sequence according to the position-based sequence weight [59]. The weight is given in equation 2.8. We take the weighted sum of the PLP for each column of the alignment.

An ensemble of several predictors usually improves the prediction rate. We used an ensemble of three independently trained HMM predictors. Because they are trained independently by running the Block-HMM, they are different in size and the parameters. It improves further compare to the predictor using three HMMs with the same structure. On each HMM, the outputs are summed up and normalised before they are used as an input of the second layer of the predictor. When the second layer is not used the outputs of the three HMMs are summed up again and the dominant label is used as our final prediction of the secondary structure.

7.2.7 The Second (structure-to-structure) Layer

To improve the performance even further we used a 3-layer perceptron consisting of 3 input nodes, 3 hidden nodes and 3 output nodes (figure 7.7). The PLPs of the ensemble

of three HMMs are used as an input of the neural networks. To train the neural networks the gradient descent method with a momentum term was used [94]. This network is quite simple compared to other structure-to-structure layers published in the literature. The final predictor is illustrated in figure 7.8.

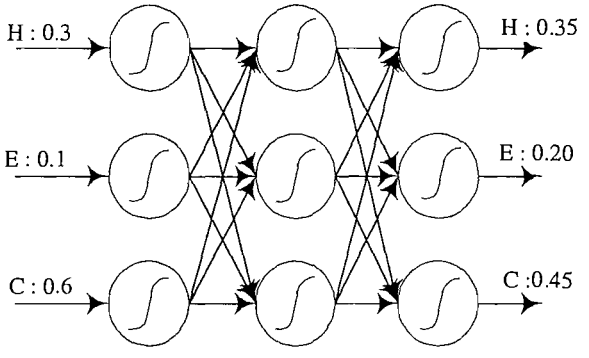


FIGURE 7.7: The structure-to-Structure layer. It is composed of simple 3-layer neural networks.

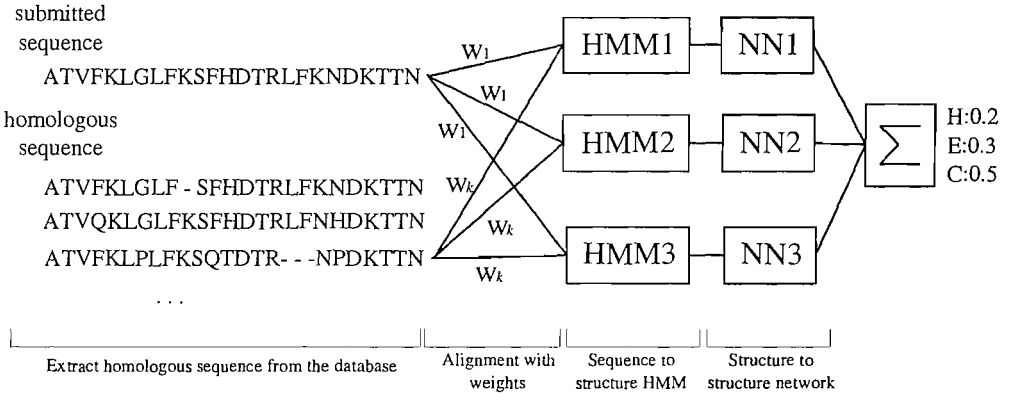


FIGURE 7.8: Schematic overview of predicting secondary structure with three HMMs evolved with Block-HMMs.

7.3 Implementation and Results

7.3.1 Cross-validation Results

We used posterior label probability to decode the HMM. The most probable label is selected as an output. Table 7.2 shows the result of 5 cross-validation tests without incorporating evolutionary information, and table 7.3 shows the result of 5 cross-validation tests. This result indicates that about 6-7% of performance enhancement has been achieved on every field except Q_E by using evolutionary information. The enhancement of Q_E was only 3%.

About 0.5% of enhancement could been achieved by using the ensemble method. The decoding result of three independently trained HMMs using Block-HMM were averaged. The ensemble of more than 3 HMMs did not improve the prediction rate much. Even though the overall Q_3 is comparable to other predictors, the Q_E is just 59.9%. To improve Q_E rate as well as overall prediction rate we added a neural network. Later, the final average Q_3 accuracy of the cross-validation tests using NNs is 75.1% and its segment overlap (SOV) scores [44] is 71.7%. Using the NNs made very little difference to the overall performance but substantially improved Q_E and SOV_E at the cost of small loss in performance of Q_H and a more decrease in Q_C. Interestingly, however, the decrease in SOV_C is just 1%.

TABLE 7.2: Average of 5 fold cross-validation results without incorporating evolutionary information.

Test	Q_3	Q_H	Q_E	Q_C	SOV	SOV_H	SOV_E	SOV_C
test1	68.0	64.9	56.0	75.2	63.2	62.8	59.8	65.5
test2	70.1	65.4	59.0	75.5	66.3	63.3	62.1	67.0
test3	67.6	67.1	52.8	74.5	63.9	65.4	59.5	65.1
test4	68.2	67.0	58.5	74.0	63.1	64.9	61.3	65.4
test5	67.5	64.7	55.8	74.7	62.6	62.5	58.5	65.9
total	68.3	65.9	56.4	74.8	63.9	63.8	59.8	65.8

TABLE 7.3: Average of 5 fold cross-validation results.

Test	Q_3	Q_H	Q_E	Q_C	SOV	SOV_H	SOV_E	SOV_C
Single HMM (test1)	74.8	71.2	60.0	82.1	69.8	70.9	66.1	69.9
(test2)	76.2	66.2	61.8	82.5	72.9	66.9	67.0	71.9
(test3)	74.5	70.8	56.9	83.2	70.4	71.5	64.4	70.1
(test4)	73.9	71.5	60.6	82.6	68.7	71.4	66.6	69.7
(test5)	73.2	66.6	58.0	80.8	69.4	66.2	62.7	68.9
Single HMM (total)	74.5	69.2	59.4	82.2	70.0	69.4	65.3	70.1
Ensemble of 3 HMMs	75.0	69.4	59.9	82.8	70.6	69.7	65.9	70.6
Ensemble + NNs	75.1	67.8	70.8	77.5	71.7	68.4	73.4	69.6

Thomsen [5] reported 49% for the Q_3 rate with his genetic methods. Figure 7.10 shows the best model he achieved. It is composed of 8 states excluding a begin state. In the figure, A, B, and C represent α-helix, β-sheet, and coil respectively. The HMM topology he found looked too simple to represent the complex nature of protein sequences. HMMSTR [6] is the most successful secondary structure predictor that makes use of HMMs. It is hand designed with more than one hundred states. However, the prediction accuracy (Q_3) of HMMSTR is 74.3%. Even though our evolved HMMs have considerably fewer states and a simpler structure, they produced better results.

Direct comparison with other predictors is difficult as there is no standard benchmark test. Most of the other top ranked predictors use test sequences with less than 25%

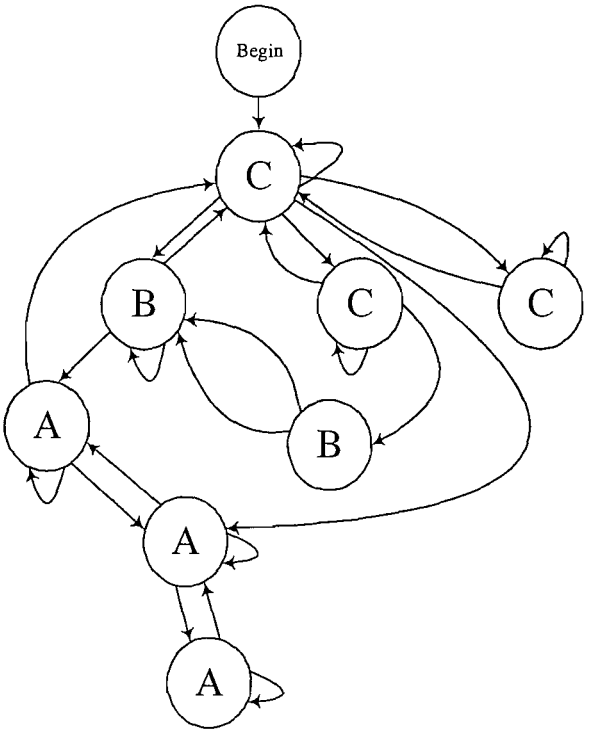


FIGURE 7.9: The best HMM topology of Thomsen. It is composed of 8 states excluding begin state. In the figure A, B, and C represent α-helix, β-sheet, and coil respectively. Adopted from [5] and redrawn.

identity with any sequence in the training set, while we used the more stringent criteria of a maximum BLAST E-value of 1 and avoiding proteins with a common fold in test and training sets. One of the most successful predictors is PSIPRED [13] which has a reported Q_3 value of 76.0%. In this case as well, the test and training sets did not share proteins with a common fold. However, their test set was considerably smaller than the one we used. The Q_3 accuracy of our method is slightly lower than for PSIPRED and the other top ranked predictors. However, given the fact that we used a very "hard" test, it is fair to assume that our results are at least comparable.

7.3.2 Benchmarking

In an attempt to benchmark our method with an existing predictor we used the EVA common subset no. 6 data set published on January 2004 [91]. We ensured that no sequence in the EVA set was present in our training set. For the benchmark we compared our prediction results with those of PSIPRED and YASPIN [16]. The result shows that the prediction rate of our method is slightly better than that of the other predictors (table 7.4).

This data set consists of only 32 sequences therefore some caution is necessary in inter-

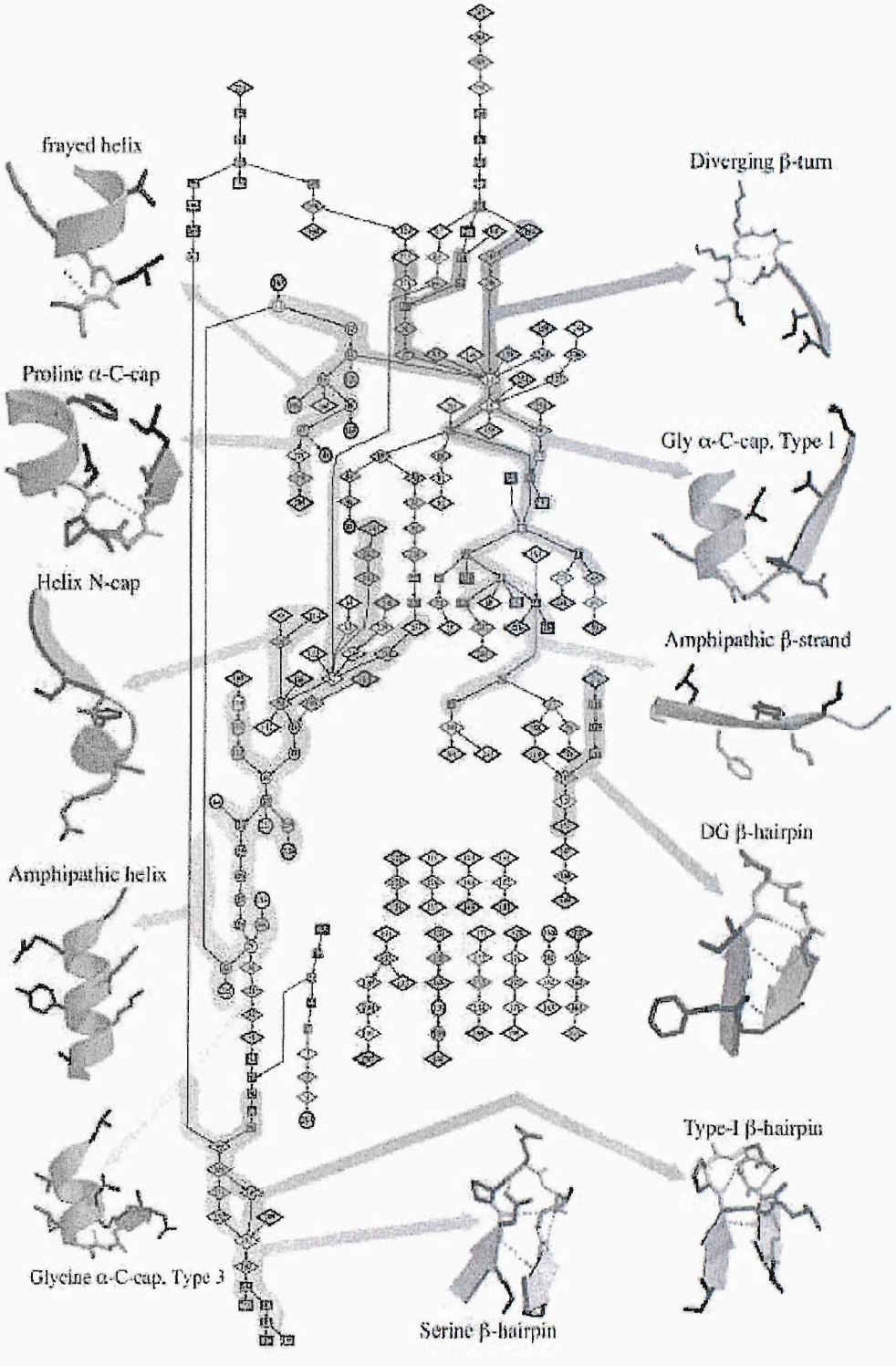


FIGURE 7.10: The HMM topology of HMMSTR adopted from [6].

preting the results. Nevertheless, this test indicates that the evolved HMM method has a prediction rate that is comparable to the other top ranked predictors. The disappointing result of YASPIN was due to failure to predicting long helical regions of proteins such as *1q90:L*, *1q90:M*, and *1q90:N*. This failure is uncharacteristic of an otherwise excellent predictor.

TABLE 7.4: The benchmarking result with 32 sequences from the EVA common subset no. 6

Test	Q_3	Q_H	Q_E	Q_C	SOV	SOV_H	SOV_E	SOV_C
PSIPRED	78.0	84.8	72.0	74.3	73.6	84.9	72.9	67.4
YASPIN	71.5	75.2	76.8	76.5	68.1	76.4	74.5	71.9
BLOCK-HMM	78.3	83.9	75.8	75.9	74.3	82.9	77.3	70.2

7.3.3 P.S.HMM: Protein Secondary structure predictor using HMM

P.S.HMM (Protein Secondary structure predictor using HMM) is a web server that emails the predicted structure of query protein sequence to users. The URL of P.S.HMM is http://www.binf.ku.dk/users/jason. Figure 7.11 shows the P.S.HMM server. The input of the predictor uses protein sequences in FASTA format. A sequence in FASTA format consists of a single-line description, followed by lines of sequence data. The first character of the description line is a '>' symbol in the first column. The P.S.HMM server is developed using Biopython. Figure 7.12 shows the result of the predicted sequence sent to the requested e-mail address.

7.4 Discussion

Optimising HMM structures using an evolutionary algorithm has several benefits. Firstly, the structure of an HMM is automatically evolved without prior knowledge of the protein sequences. This is remarkable given that other methods for secondary structure prediction require considerable calibration. Instead we had a benefit of using the general knowledge of biological sequences by using HMM block models. Compared to the hand-designed HMMSTR [6], the evolvable method produced better results with a smaller number of states. In the case of neural networks, the selection of the number of units needs careful attention. Here again, the evolving HMM method is an attractive alternative.

Secondly, our method does not require a sliding window as most other secondary structure prediction methods do. The size of the window is chosen in order to obtain good performance (for example, PSIPRED has a window size of 15 [13]). The evolving HMM

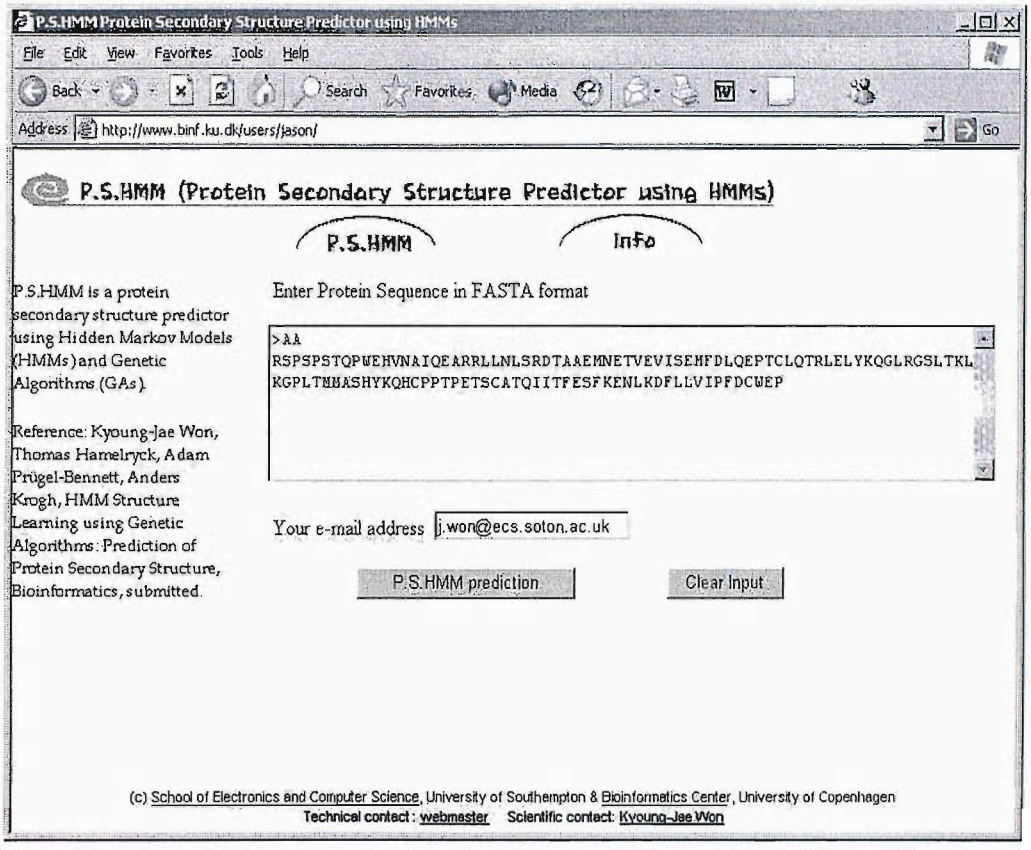


FIGURE 7.11: The P.S.HMM server. A protein sequence in FASTA format is used as an input.

```
P.S.HMM prediction result.

>AA
Seq : RSPSPSTQPWEHVNAIQEARRLLNLSRDTAAEMNETVEVISEMFDLQEPTCLQTRLELYK
Pre : xxxxxxxxxxHHHHHHHHHHHHHHHHxxxxxxxHHHHHHHHHHHHHHHxxxxxxxHHHHHHHHHHHH

Seq : QGLRGSLTKLKGPLTMMASHYKQHCPPTPETSCATQIITFESFKENLKDFLLVIPFDCWE
Pre : HxxxxxxHHHHHxHHHHHHHHHHHHxxxxxxxxxxxxxxEEEEHHHHHHHHHHHHHHxxxxxxxxxx

Seq : P
Pre : x


The prediction resulf of P.S.HMM
```

FIGURE 7.12: The result of secondary structure prediction.

method uses the whole sequence as input, which avoids the use of a fixed sequence window that might affect performance in specific cases.

The prediction rate in the cross validation tests is a bit lower than that of the other predictors. One of the reasons is that we used a relatively hard data set to test our performance. For the benchmarking, the fully trained model was compared with other predictors. PSIPRED is known as one of the top ranking predictors and YASPIN is a recently developed challenger. In the benchmark test the overall Q_3 rate of our predictions is slightly better than the other methods tested. Because we conducted the benchmark with a small number of sequences, we cannot claim that our method is better than others. However, the result provides strong evidence that our method has at least as good a performance as other published methods.

In our classification, a helical region has at least 3 residues (*i.e.* a 3-residue 3_{10} helix as determined by DSSP). At the moment our method can predict a helical region of less than 3 residues. This can be corrected for example by deleting all helices having 1 or 2 residues. However, this does not greatly affect the prediction rate and consequently we have kept this issue untouched.

At present the Block-HMM method is relatively slow because it has to train and calculate fitness for all the HMM members in the population. We could solve it by using parallel programming. To evolve an HMM using GAs with 30 members in a population, we used 31 2.4 GHz P4 processors each with 512 Mb RAM run in parallel. One processor controls communication between other processors. Under these conditions each processor trains one HMM. Ideally, the CPU time consumed in each processor is the time to train and evaluate an HMM multiplied by the number of iteration. Approximately 7 hours was required to produce an HMM with 40 states. Prediction using three trained HMMs without evolutionary information takes about 30 seconds.

Chapter 8

Inside Block-HMMs

Block-HMMs use an evolutionary strategy to search for a solution. Like other GA applications, they rely on many parameters such as the population size and the genetic operators. They also have special properties created by the use of blocks. These factors altogether affect the whole performance of Block-HMMs.

In this chapter we delve into the factors that Block-HMMs use. For this investigation we perform the protein secondary structure prediction with a new set of 200 training sequences and 64 test sequences. Firstly, we compare GA-HMMs and Block-HMMs for this problem. In the following simulations, we investigate the performance of the Block-HMM as we change the parameters such as the genetic operators and the number of the HMM blocks. We also investigate the performance changes as we vary the separation ratio of the training sequences and see if the input separation scheme and the selective Baum-Welch reduce overfitting for protein secondary structure prediction problem. We also check how the population size effects the performance. Lastly, we compare the Block-HMM with a hill climbing method.

During the simulation, when no specific parameter is given, the parameter set in table 8.1 is used. After we obtained an HMM by running each algorithm, the evolved HMM is trained again with the Baum-Welch algorithm and the discriminative training algorithm [42] using the whole 200 training set as we previously conducted in chapter 7.

TABLE 8.1: Default parameters used for the protein secondary structure prediction.

Parameter	value
Population size	30
Iteration	150
Number of blocks in an HMM	12
Crossover rate	0.07
Mutation rate	0.07
Type-mutation rate	0.07

8.1 GA-HMMs versus Block-HMMs

The GA-HMM allows any number of transitions from a state, while the Block-HMM allows only transitions that do not break the property of blocks. In the Block-HMM the crossover operator is also restricted as they allow only swapping of the blocks. We investigate how the GA-HMM evolves HMM structures and compare it with the Block-HMM. We started with the initial 6-state model. The initial HMM is illustrated in figure 8.1. When we trained this model with Baum-Welch algorithm and the discriminative training algorithm, it had the Q_3 rate of 63.8%.

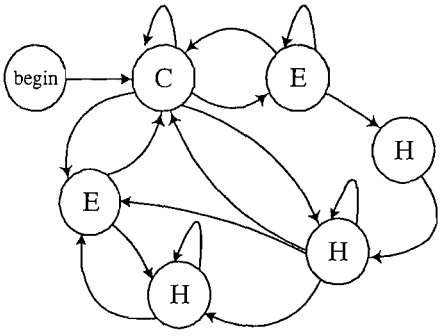


FIGURE 8.1: The initial HMM used for the GA-HMM simulation. The Q_3 of this initial model is 63.8%.

With this initial model we ran the GA-HMM. Figure 8.3 (a) shows the log likelihood of the best HMM versus iteration when we ran the GA-HMM. The graph shows steep changes before 50 iterations caused by the structural changes in the HMM, but after 50 iterations it does not show such changes and looks like a normal Baum-Welch training curve on an HMM. The average Q_3 rate of the 6 GA-HMM tests is 64.71%. Figure 8.2 shows one of the results after running the GA-HMM.

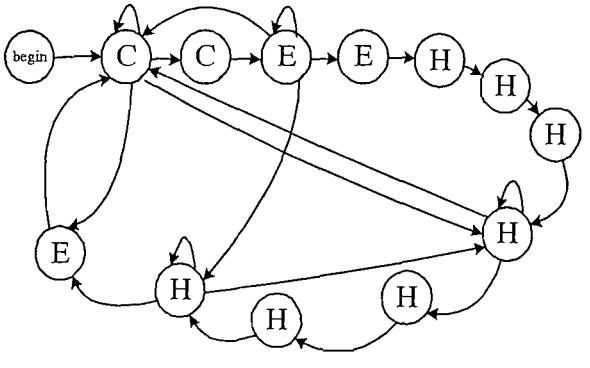


FIGURE 8.2: The result HMM topoogy after the GA-HMM simulation.

The result of GA-HMMs shows the increase of the number of the states and some additional transitions. This result can be comparable with Thomsen's result (figure 7.10)

which is independently developed using similar genetic operators [5]. Thomsen evolved an initial 3-state HMM topology and produced a result with 8 states. Both of the methods could not evolve the initial model efficiently for this problem. We conducted the same simulation until the GA-HMM reached 2000 iterations, but we could not find any significant improvement.

Block-HMMs usually use a larger number of states for the initial population. To get rid of the initial advantage of Block-HMMs, we tested the GA-HMM with one of the initial members of the population of the Block-HMM. The initial model has 34 states and its Q_3 rate was 65.48% before it is evolved. Figure 8.3 (b) shows the log-likelihood graph of the best HMM versus iteration. The log likelihood graph is higher than the GA-HMM graphs with the initial 6-state HMM (figure 8.3 (a)). The average Q_3 rate of the six independent tests is 66.16%.

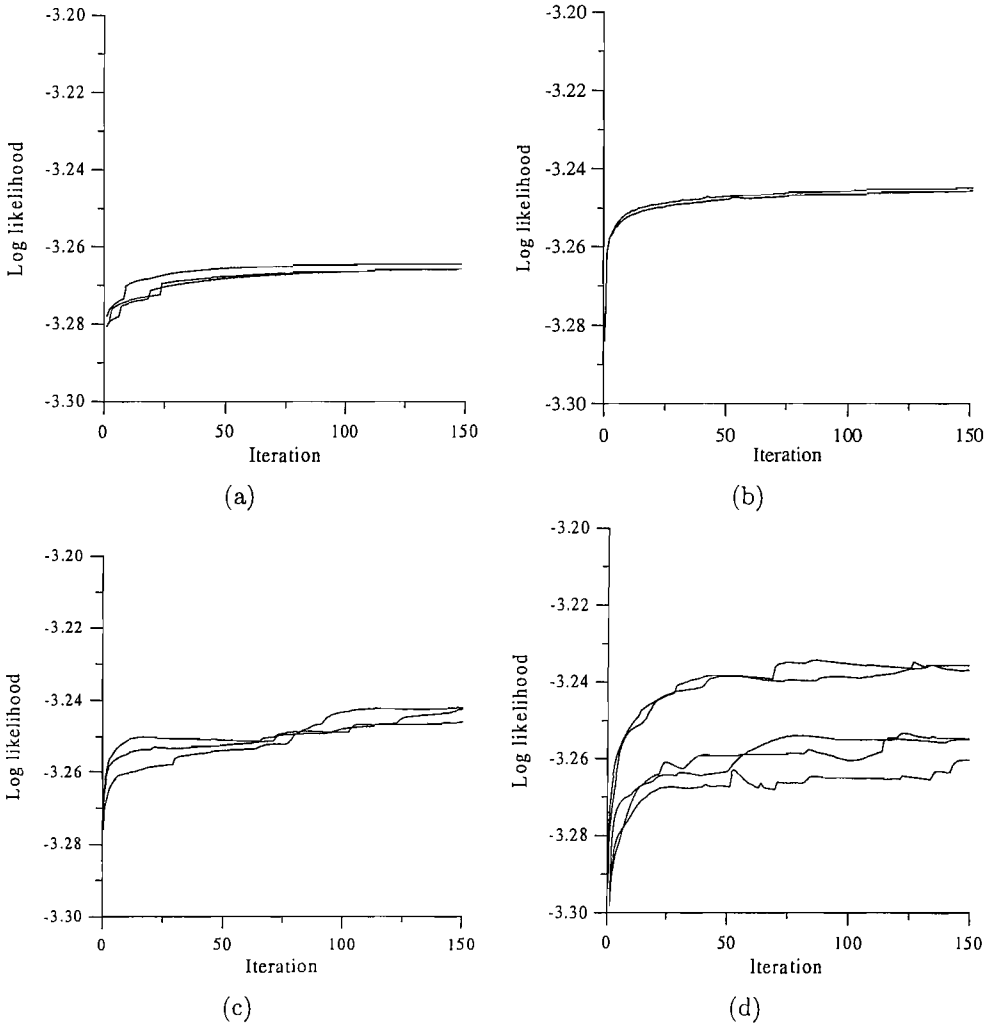


FIGURE 8.3: The comparison of the log-likelihood graph of best HMMs (a) GA-HMM from a initial 6-states HMM (b) GA-HMM from a initial model with 34 states (c) Block-HMM from a initial model with 34 states (d) Block-HMM from a random population.

We conducted the simulation using the Block-HMM with the same initial HMM. Figure 8.3 (c) shows the log-likelihood of the best HMM versus iteration. The results demonstrated that the Block-HMM could change the structure more than the GA-HMM could. The Q_3 rates of all the result was better than GA-HMM's result and the average Q_3 rate of 10 independent simulations was 66.88%. The Block-HMM continued to change the log-likelihood until it reached the end of the iteration. The improvement of prediction rate from the initial model is 1.4%, while the GA-HMM could improve only 0.7%. Figure 8.3 (d) is the log-likelihood of the best HMM versus iteration when we evolved it from a random population. The structure of each member of the population is randomly generated. The graphs are more widely spread than the other results. Even though some of the log-likelihood graphs are lower than the curves in figure 8.3 (c), all the results have better Q_3 performance and we can get the Q_3 of 67.15%. From the simulation, we can conclude that the Block-HMM is superior for this secondary structure problem. One of the reasons of success of the Block-HMM in the protein secondary structure prediction problem is that the Block-HMM crosses over without breaking the group of states that share the same label. When we use labelled sequences, breaking those group of states easily makes illegal sequence paths through the HMM. The Block-HMM can avoid those useless operations by using blocks. Also, type-mutation can replace a block with any other block type, which might give Block-HMMs additional jumps to search the solution.

GA-HMMs may have better fine tuning capability than Block-HMMs because they allow any number of transitions. However, the result with protein secondary structure shows that Block-HMMs are much more suitable for the secondary structure prediction problem. This may reveal that Block-HMMs benefit from restricting their topology.

8.2 Genetic Operators

Three genetic operators are used in Block-HMMs: crossover, mutation and type-mutation. To investigate how each genetic operator effect the performance of Block-HMMs, we ran simulations using only one of these operators for each test.

The figure 8.4 shows the log-likelihoods of the five best HMMs versus iteration when each of the operator is used. The crossover operators sometimes bring rather abrupt changes in the graph (figure 8.4 (b)) and the graphs are widely spread. Mutation operators, on the other hand, give small changes (figure 8.4 (a)). The Q_3 rates of each simulation are shown in table 8.2. Even though we used a different operator in each case, the Q_3 performances were not significantly affected. All of the three operators are important methods to change HMM structures directly. However, it is difficult to pinpoint which operator plays the more important roles from this simulation.

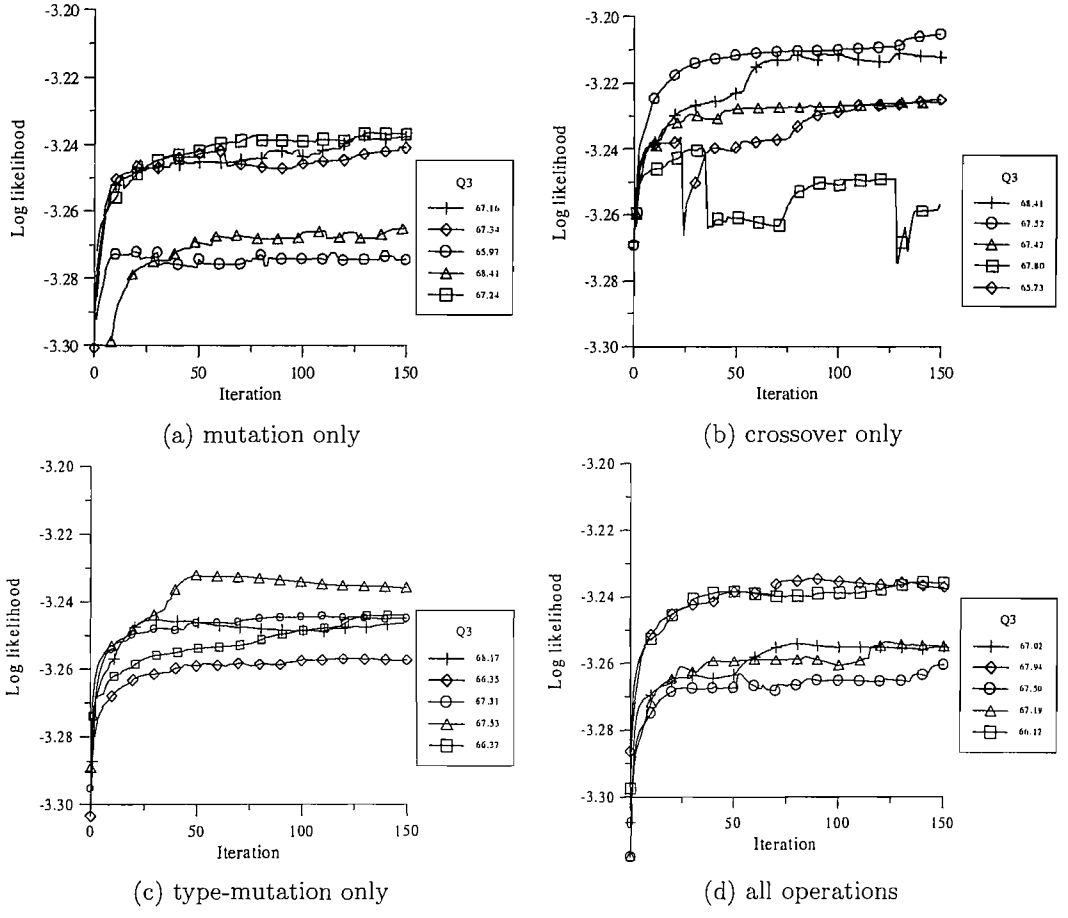


FIGURE 8.4: Block HMM graphs with each operation (a)mutation only (b) crossover only (c) type-mutation only (d) all operations.

TABLE 8.2: The comparison of Q_3 rates and the number(♯) of states when using one of each operator.

	mutation only		crossover only		type-mutation only		all operations	
	Q_3	♯ of states	Q_3	♯ of states	Q_3	♯ of states	Q_3	♯ of states
test1	67.16	34	68.41	35	68.17	33	67.02	25
test2	67.34	49	66.75	33	66.35	19	67.94	27
test3	65.97	29	67.42	28	67.31	38	67.50	30
test4	68.41	34	67.8	27	67.53	22	67.19	26
test5	67.24	29	65.73	25	66.37	22	66.12	26
average	67.22	35	67.22	29.6	67.15	27.8	67.15	27.2

The result also shows that the high log-likelihood does not necessarily mean high Q_3 rate. The highest log-likelihood in the figure 8.4 (d) (□), has the worst Q_3 rate (66.12%). Also, the size of HMMs does not strictly determine the performance in the simulation, because HMMs with similar sizes still show differences in the individual performance.

8.3 Number of Blocks

The number of blocks is an important factor in the Block-HMM. The number of blocks is fixed in the Block-HMM, but due to the zero blocks the number of blocks representing an HMM changes. In this section we investigate the performance change as we vary the number of blocks in the Block-HMM. For the simulation we varied the number of blocks from 5 to 25. The size of blocks are assigned randomly from 0 to 7. Figure 8.6 shows the log-likelihood of the best HMM when the numbers of blocks used are 5, 8, 13, 15, 17 and 25, respectively.

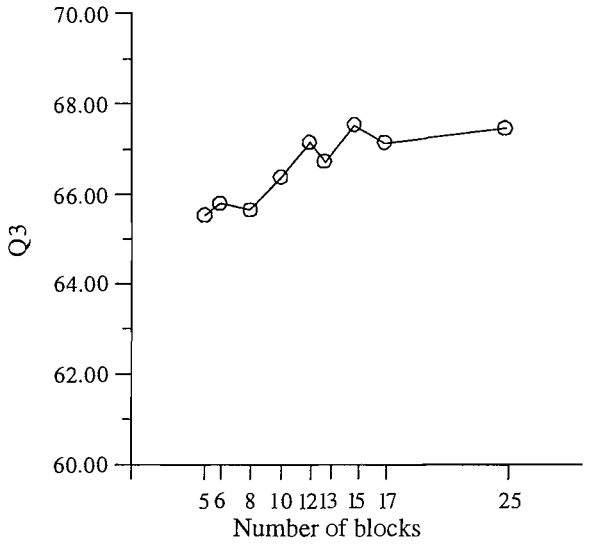
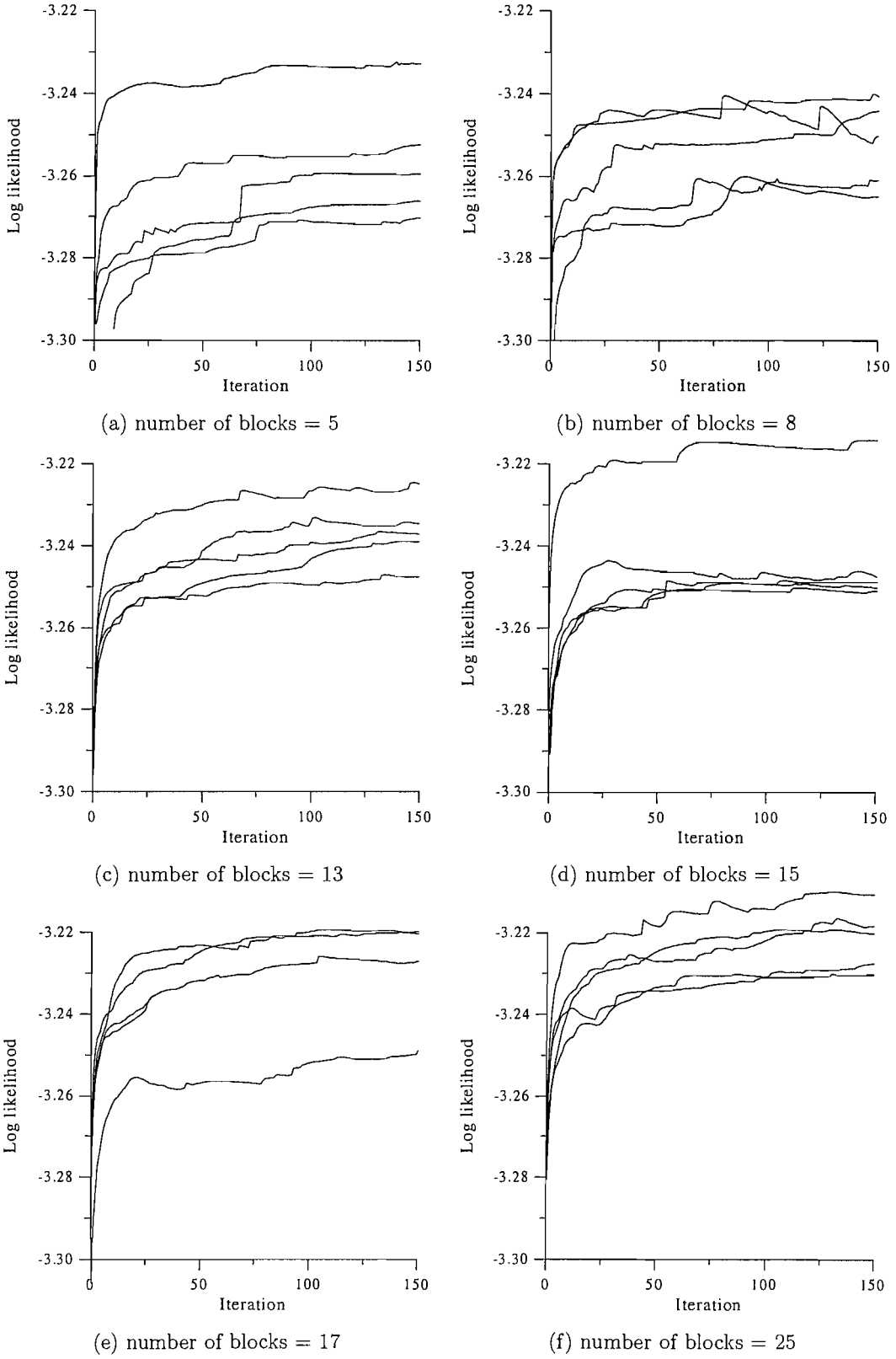


FIGURE 8.5: The Q_3 rate on each case of the number of blocks.

TABLE 8.3: The average Q_3 rate and the average number of states on each case of the number of blocks.

♯ of blocks	5	6	8	10	12	13	15	17	25
Q_3	65.54	65.79	65.65	66.37	67.15	66.72	67.53	67.14	67.46
♯ of states	16.8	19.8	18.2	27.6	27.2	31.0	32.4	37.4	45.6

For each configuration five tests were conducted. The results show the tendency that the log-likelihood graphs are a little higher as the number of blocks increases. HMMs with a larger number of states usually have higher log-likelihood. Table 8.3 shows the average Q_3 rate and the average number of states of the HMMs evolved on each case.



(a) number of blocks = 5

(b) number of blocks = 8

(c) number of blocks = 13

(d) number of blocks = 15

(e) number of blocks = 17

(f) number of blocks = 25

FIGURE 8.6: The log-likelihood of the best HMM when the number of blocks are 5, 8, 13, 15, 17 and 25.

As the number of blocks increases, the number of states of the HMM increases. This affect the log-likelihood. The Q_3 rate increases as the number of blocks increases up to 12. However, the improvement of performances stops once the Block-HMM has enough number of blocks (figure 8.5).

8.4 Overfitting

Overfitting occurs when an HMM is trained too much with a training sequences. Overfitting is hard to avoid and it becomes more serious as the number of states of an HMM increases. To avoid overfitting we proposed two schemes: the separation of dataset and the selective Baum-Welch method. The separation scheme divides the training sequences into two sets. One set (training set) is used for the Baum-Welch training and the other set (evaluation set) is used for the fitness evaluation. The selective Baum-Welch scheme randomly judges whether to train the HMM or not. The simulation result with promoter region of *C. jejuni* demonstrates that the two schemes helped the GA-HMM to reduce the overfitting of the training sequences. For the DNA sequences we calculated the log-likelihood of the test set to check the generalisation performance. In this section, we investigate how those two methods effect the performance of Block-HMMs for the protein secondary structure prediction problem.

Figure 8.7 shows the log-likelihood of the best HMMs versus number of iterations. For this test we used 20 protein sequences to see the overfitting phenomena easily. When the separation scheme is not used ($\triangle$) the log-likelihood graph monotonically increases over iteration. When the separation scheme is used the log-likelihood curve of the training set ($\Diamond$) is below the graph of the non-separation method. It also fluctuates because the trained HMM is assessed by using the evaluation set. The log-likelihood graph of the evaluation set ($+$) shows that the increase of the log-likelihood of the training set does not always mean the increase of the evaluation set's log-likelihood. Unlike the log-likelihood, the average Q_3 of the non-separation scheme (57.41 ± 0.41) is less than that of the separation scheme (58.85 ± 0.73). The log-likelihood graphs of the two scheme become similar to each other as the number of sequences increases. The difference in Q_3 also decreases as the number of sequences increases.

8.4.1 Separation versus Non-separation

We investigate the effect of the separation method as we change the ratio of the separation. Figure 8.8 shows the log-likelihood of the best HMMs versus number of iterations when the separation ratio varies. When the training sequences are not enough (1:6) the curve fluctuates substantially. The graphs fluctuate less as the ratio of the training set increases. Table 8.4 shows the average Q_3 rates of five tests for each configuration. The

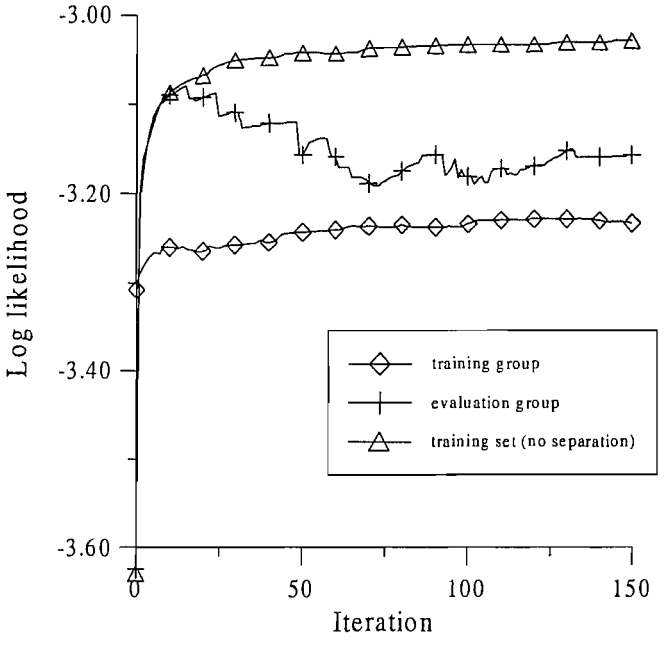


FIGURE 8.7: The log-likelihood of best HMMs versus number of iterations with 20 training sequences.

results show that the best results are obtained when the number of sequences in the training set is similar or a bit larger than the number of sequences in the evaluation set.

TABLE 8.4: The comparison of average Q_3 rates of five tests for each case (number of sequence in training set: in evaluation set).

test	1:6	2:5	3:4	4:3	5:2	6:1
average Q_3	64.18	65.79	67.55	66.94	67.15	66.65

To see if the separation scheme reduces the effect of overfitting we conducted the simulation on both the separation and the non-separation methods. We conducted the student's t-test with the results of both methods. For each method we conducted 30 independent tests. When we tested the separation method we used half of the sequences as the evaluation set and the other half as the training set.

The average Q_3 rate of the 30 tests is 67.66% when the separation method is used and 67.36% when the non-separation method is used. The probability of the t-test result, assuming the null hypothesis, is 0.119. This suggests that the separation method is likely to be superior, although more testing would be necessary to be sure.

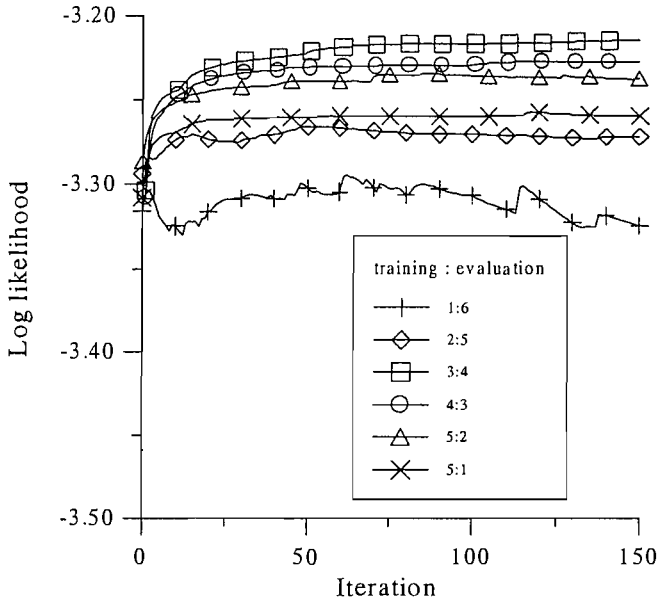


FIGURE 8.8: The log-likelihood of the best HMMs versus iteration when the ratio of the separation ratio varies.

8.4.2 Selective Baum-Welch Methods

Selective Baum-Welch methods were introduced in chapter 5 to reduce overfitting. The selective Baum-Welch method increased the generalisation performance when we tested the log-likelihood of the DNA sequences. However, the log-likelihood does not always guarantee the Q_3 performance in the protein secondary structure prediction problem. We investigate how the selective Baum-Welch works in the Block-HMM for the protein secondary structure prediction problem. About 1/4 of the members of a population are randomly selected to train with the Baum-Welch algorithm. Figure 8.9 shows a log-likelihood graph when the selective Baum-Welch method was used.

The graph of the selective Baum-Welch method increases monotonically. Because the best member is not Baum-Welch trained and is replaced only when other members have more fitness value, the graph of the selective Baum-Welch method always increases.

The average Q_3 rate of this method from 10 tests was 66.63%. Under the same condition, the Block-HMM without the selective Baum-Welch method was 66.87%. The differences of the two methods are insignificant as the probability associated with the t-test result is 0.5. On the contrary to DNA sequences, the selective Baum-Welch method did not have a significant effect on the performance of the Block-HMM for the protein sequences prediction problem.

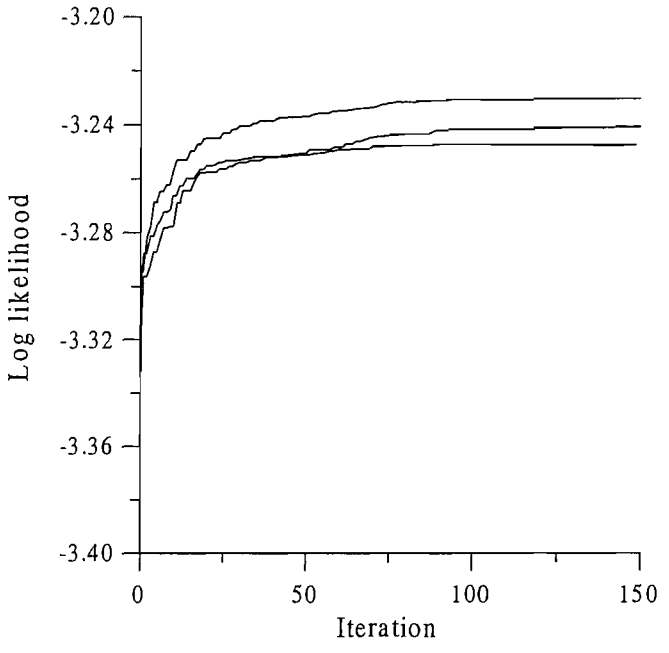


FIGURE 8.9: The log-likelihood of the best HMMs versus number of iterations when the selective Baum-Welch method is used.

8.5 Population Size

The population size is a unique property of GAs. The population size tells how many solutions the GA maintains. To investigate how the population size effect the performance of Block-HMMs, we checked the performance on each configuration of the population size.

Figure 8.11 shows the log-likelihood graphs when the population size are 4, 8, 16, 30, 40, and 50, respectively. Obviously, the Block-HMM could not evolve efficiently when the population size is 4 because of the disruption caused by genetic operators. The log-likelihood usually have higher values as we increased the population size. Table 8.5 and figure 8.10 show the average and the maximum Q_3 rates of ten independent tests. The average Q_3 rate has a higher value as the population size becomes larger. However, the maximum Q_3 rates remain similar when the population size is larger than 16.

The result shows that the performance was not improved even though we use the larger population size. However, it increased the number of results that showed good performance, which led to the increase of average Q_3 rate.

TABLE 8.5: The average and the maximum Q_3 rate when the population size changes.

population size	4	8	16	30	40	50
average Q_3	64.27	66.63	67.27	67.67	67.50	68.07
maximum Q_3	66.45	68.40	68.74	68.57	68.73	68.68

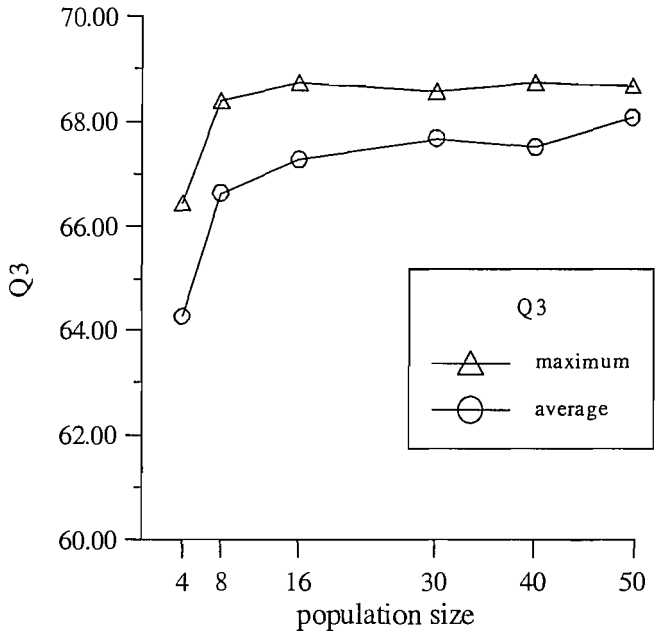
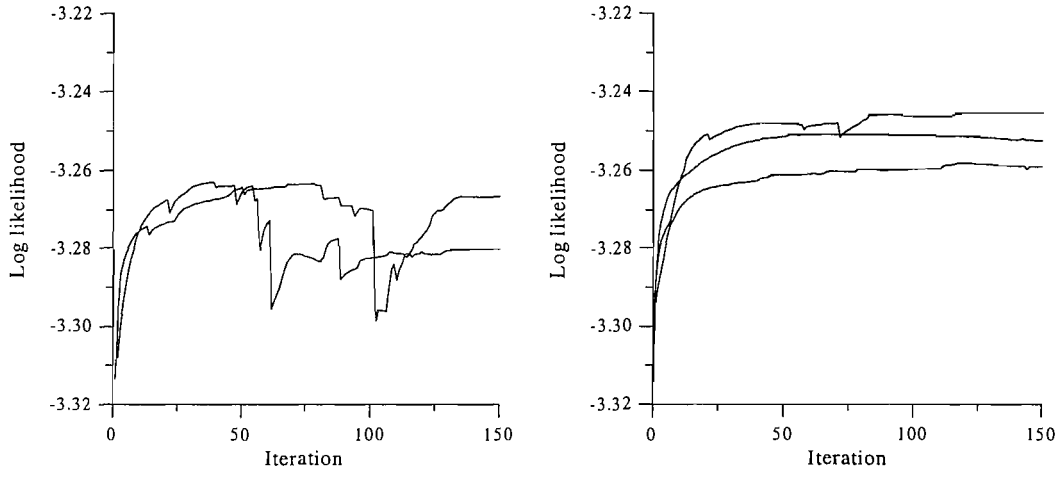


FIGURE 8.10: The average and the maximum Q_3 rate when the population size changes. The maximum Q_3 rate does not change much.

8.6 Comparison with Hill Climbing method

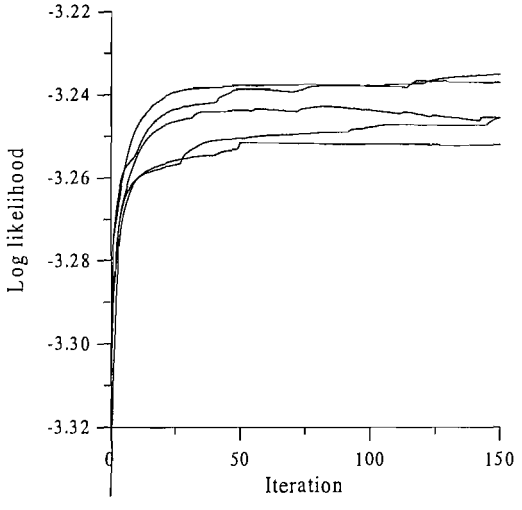
Hill climbing [95] methods are search algorithms where a new solution replaces the old one whenever the new solution has a better performance. In this section we compare the Block-HMM with the hill climbing method. For this simulation the hill climbing method is designed to have HMM blocks and use the same mutation operator and the type-mutation operator. We also used the separation method of training sequences for the hill climbing method. Half of the training sequences were used for the training set and the other half for the evaluation set.

Figure 8.12 shows the log-likelihood graph of the hill climbing method. We can find the structural changes in the log-likelihood graph. The average Q_3 rate of independent 30 tests is 66.62%. On the same configuration the Q_3 of the Block-HMM was 67.67% when the population size is 30. The result of the t-test also shows the significant differences between the two result (probability is 6.8×10^{-6}). The statistical difference is large, but this result is a bit unfair to the hill climbing method because the result has not been compared on an equal number of evaluations. The performance of the hill climbing

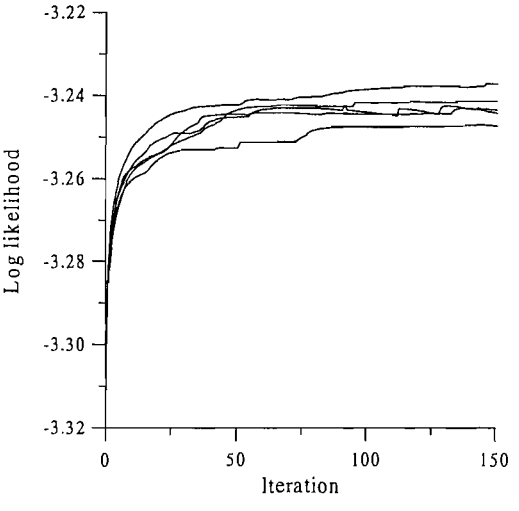


(a) population size = 4

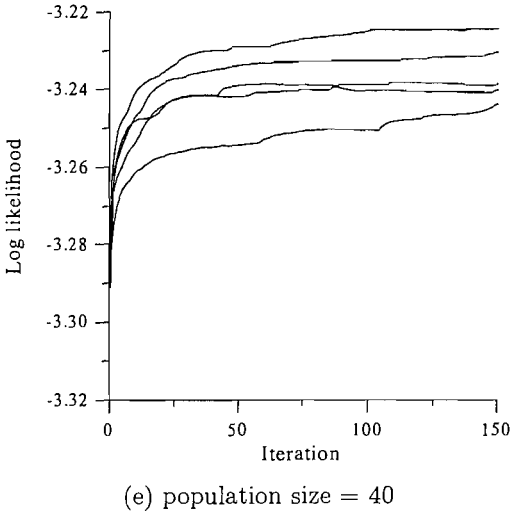
(b) population size = 8



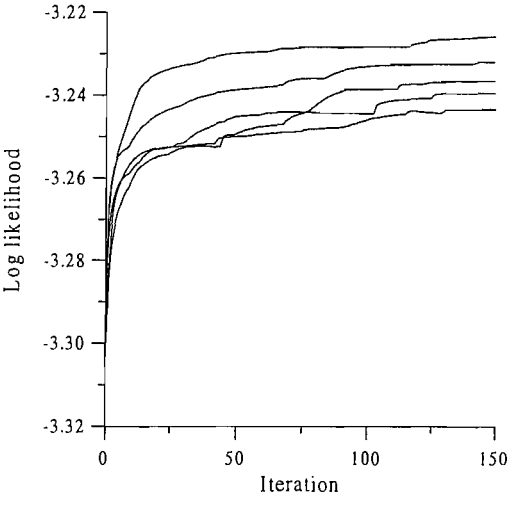
(c) population size = 16



(d)population size = 30



(e) population size = 40



(e) population size = 50

FIGURE 8.11: The log-likelihood of the best HMMs versus number of iterations when the population size are 4, 8, 16, 30, 40, and 50

method is better than the Block-HMM with a population size of 4. This comes from the fact that the Block-HMM is disrupted by its crossover operator. We did not use elitism for this chapter because selecting the best one each time did not contribute to the performance much. Especially, for this simulation the population size of 4 would be too small to use the elitism. Previously, we have shown that the Block-HMM could evolve well without using crossover operation. The reason for the Block-HMM's superiority over the hill climbing method may come from that the Block-HMM maintains the pool of a population and can search many solutions at the same time while selecting the better members of the population.

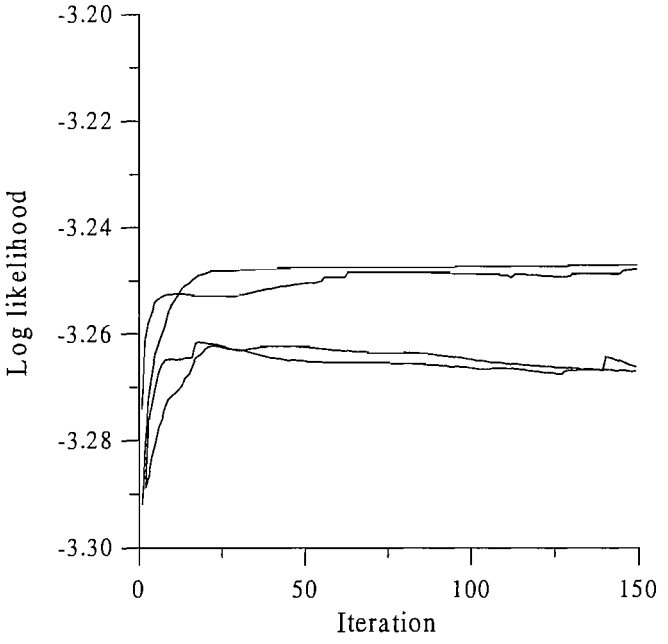


FIGURE 8.12: The log-likelihood of the HMM trained using the hill climbing method.

8.7 Discussion

In this chapter we investigated what factors contribute to the performance of the Block-HMM. From the simulation of comparing the GA-HMM and the Block-HMM we found that the block mechanism enables the Block-HMM to generate the plausible initial population and to evolve the population in an efficient way. When evolving a new structure in the GA-HMM, it can easily cause illegal paths through an HMM by adding or deleting a state or swapping them in the HMM irrelevantly. The genetic operators designed for the Block-HMM could tune the structure without causing too much disruption by suggesting plausible architecture at each time. Also, each block has a very simple structure. This may help the Block-HMM to suffer overfitting less.

The separation of the dataset has been used in evolving the HMM structure. From the simulation we demonstrated that the separation method has some significant effects to reduce overfitting. This method uses the statistical property of the dataset itself, while other hybridising methods uses the property of the applied HMM, the complexity. However, it may not be appropriate to use the property of an HMM because it is hard to define how much complexity of the HMM is needed for the given problem. We also used the selective Baum-Welch method to reduce overfitting. However, in the case of protein sequences, the selective Baum-Welch method was not beneficial.

The number of blocks determines the size of an HMM. The size of an HMM did not affect the performance much when it was large enough. However, it may not be a good strategy to use an extreamely large number of blocks, because it requires a lot of computational effort. Using a large population size increases the chance to find a good solution. Also, increasing the population size requires additional computational effort.

Because of the time-consuming process of the Block-HMM and limitation of the computational power, we conducted limited number of tests. It is difficult to draw strong conclusion about the Block-HMMs from the tests conducted in this chapter. Some more simulations result may help to understand how the Block-HMM works.

Chapter 9

Conclusions

In spite of the numerous number of HMM applications developed for biological sequences analysis, the structure training methods have not been studied much. Even though heuristic methods for structure training have been experimentally applied, they have not exceeded the human's expertise in designing HMM structure.

In this thesis, we presented a method to train the HMM structure. Evolutionary strategies were designed to search for the HMM structure as well as the HMM parameters. This automatic way of training HMM structure method considered possible issues raised in HMMs in its evolutionary cycle. Firstly, it hybridised the Baum-Welch parameter training algorithm for efficient parameter convergence. Secondly, it developed the self learning method by strictly separating the evaluation from a different set of training data. This approach keeps itself from overfitting the training data and leads the HMM to have better generalisation performance. Block-HMMs use block models which are highly motivated by the HMM applications for analysing biological sequences.

With all those property, the evolving HMM method could find better models than experts can come up with. Especially, the Block-HMM was designed to evolve HMM structures for the whole sequences and the performance of the Block-HMM was superior to the hand-crafted model. It may be natural that there is no one globally superior solution in finding the structural model that can represent biological phenomena. For the given biological sequences there would be many possible HMM structures. The Block-HMM suggests one of the possibilities of an automatic structure searching method. Besides those block types suggested in this thesis, other types of block also can be used to produce better results. Nevertheless, the proposed method is a successful example of evolving HMM structures.

By investigating the Block-HMM, we could conclude that the use of the HMM blocks combined with the properly designed genetic operators enabled such an improvement. Even though the performance improvement by the separation of the training set was not

significant, it was proved to be a good candidate to evaluate a trained HMM on each genetic cycle.

The GA-HMM and the Block-HMM methods were used to model the promoter and the coding region of *C. jejuni* and protein sequences. Due to the automatic nature of our methods, drawing a biological meaning from the HMM might be difficult. However, in some cases it may not be desirable to draw a biological meaning from the trained HMM. Instead, the whole HMM can be used to interpret and understand the biological sequences.

There is still need for research in the Block-HMM. Besides new types of blocks, the fitness function is another area to be developed. Decoding methods may be directly applied for the fitness evaluation. At this stage, nevertheless, there exist advantages in using Block-HMMs. One of the main advantages of using Block-HMMs is that they can be directly applied to other sequence prediction methods easily. Since the Block-HMM is designed to generate a general model for the given sequences, it can be applied to any kinds of DNA or protein sequences.

Glossary

3' end The end of a nucleic acid that doesn't have a nucleotide bound to its 3' of the terminal residue.

5' end The end of a nucleic acid sequence where the 5' position of the terminal residue isn't bound by a nucleotide.

Amino acid Any of 20 basic building blocks of proteins. It is composed of a free amino (NH2) end, a free carboxyl (COOH) end and a side group (R).

Base pair (bp) A pair of complementary nitrogenous bases in a DNA molecule (adenine-thymine and guanine-cytosine). Also, the unit of measurement for DNA sequences.

capped 5'-ends A methylated (has a -CH3 attached) guanosine nucleotide attached to the 5'-end (the beginning) of an eukaryotic mRNA, thought to give the mRNA stability.

Codon A group of three nucleotides that specifies addition of one of the 20 amino acids during translation of an mRNA into a polypeptide. Strings of codons form genes and strings of genes form chromosomes.

Consensus sequence A sequence of nucleotide bases which are extremely similar among many different genes of different systems. In eukaryotes, this sequence is known as the "TATA box" or the "Hogness box," and it has the general sequence of TATAAAA. In the bacteria *Escherichia coli* it is known as the "Pribnow box" and has the general sequence of TATAATG. The sequence is most often found in promoters and does important things like binding important proteins (including RNA polymerase) to initiate transcription.

Conserved sequence A base sequence in a DNA molecule (or an amino acid sequence in a protein) that has remained essentially unchanged throughout evolution.

CpG islands Areas of DNA which consist mostly of the base pair sequence CGCGCGCG$\cdots$ (alternating cytosine and guanine nucleotide bases) that are usually found upstream of many genes and are thought to help regulate gene expression. They are often methylated (have methyl groups attached to the DNA segments).

DNA (Deoxyribonucleic acid) An organic acid and polymer composed of four nitrogenous bases–adenine, thymine, cytosine and guanine linked via intervening units of phosphate and the pentose sugar deoxyribose. DNA is the genetic material of most organisms and usually exists as a double-stranded molecule in which

two antiparallel strands are held together by hydrogen bonds between adenine-thymine and cytosine-guanine.

Downstream The region extending in a 3' direction from a gene.

Eukaryote An organism whose cells possess a nucleus and other membrane-bound vesicles, including all members of the protist, fungi, plant and animal kingdoms; and excluding viruses, bacteria and blue-green algae.

Exon A DNA sequence that is ultimately translated into protein.

Gene A locus on a chromosome that encodes a specific protein or several related proteins. It is considered the functional unit of heredity.

Gene splicing Combining genes from different organisms into one organism.

Human Genome Project A project coordinated by the National Institutes of Health (NIH) and the Department of Energy (DOE) to determine the entire nucleotide sequence of the human chromosomes.

Intron A non-coding DNA sequence within a gene that is initially transcribed into messenger RNA but is later snipped out.

Messenger RNA (mRNA) The class of RNA molecules that copies the genetic information from DNA, in the nucleus and carries it to ribosomes, in the cytoplasm, where it is translated into protein.

Open reading frame A long DNA sequence that is uninterrupted by a stop codon and encodes part or all of a protein.

Poly(A) polymerase Catalyzes the addition of adenine residues to the 3' end of pre-mRNAs to form the poly(A) tail.

Polymerase (DNA) Synthesizes a double-stranded DNA molecule using a primer and DNA as a template.

Prokaryote A bacterial cell lacking a true nucleus; its DNA is usually in one long strand.

Promoter A region of DNA extending 150-300 bp upstream from the transcription start site that contains binding sites for RNA polymerase and a number of proteins that regulate the rate of transcription of the adjacent gene.

Protein A polymer of amino acids linked via peptide bonds and which may be composed of two or more polypeptide chains.

Reading frame A series of triplet codons beginning from a specific nucleotide. Depending on where one begins, each DNA strand contains three different reading frames.

Ribosome Cellular organelle that is the site of protein synthesis during translation.

Ribosome-binding site The region of an mRNA molecule that binds the ribosome to initiate translation.

RNA (ribonucleic acid) An organic acid composed of repeating nucleotide units of adenine, guanine, cytosine and uracil, whose ribose components are linked by phosphodiester bonds.

RNA polymerase Transcribes RNA from a DNA template.

Stop codon Any of three mRNA sequences (UGA, UAG, UAA) that do not code for an amino acid and thus signal the end of protein synthesis.

TATA box An adenine and thymine rich promoter sequence located 25-30 bp upstream of a gene, which is the binding site of RNA polymerase.

Template An RNA or single-stranded DNA molecule upon which a complementary nucleotide strand is synthesized.

Transcription The process of creating a complementary RNA copy of DNA.

Translation The process of converting the genetic information of an mRNA on ribosomes into a polypeptide. Transfer RNA molecules carry the appropriate amino acids to the ribosome, where they are joined by peptide bonds.

Upstream The region extending in a 5' direction from a gene.

UTR Untranslated region.

Bibliography

[1] National Health Museum, "http://www.accessexcellence.org/".

[2] Kimball's Biology Pages, "http://biology-pages.info".

[3] W. K. Purves, G. H. Orians, and D. Sadava, *Life : The Science of Biology, 4th Edition.* Sinauer Associates and WH Freeman, 1992.

[4] MDL Chime, "http://www.mdl.com/products/framework/chime/".

[5] R. Thomsen, "Evolving the Topology of Hidden Markov Models Using Evolutionary Algorithms," *Lecture Note in Comuter Science*, vol. 2439, pp. 861–870, 2002.

[6] C. Bystroff, V. Thorsson, and D. Baker, "HMMSTR: a Hidden Markov Model for Local Sequence-Structure Correlations in Proteins," *Journal of Molecular Biology*, vol. 301, pp. 173–190, 2000.

[7] S. Brunak, J.Engelbrecht, and S. Knudsen, "Prediction of human mRNA donor and acceptor sites from the DNA sequence," *Journal of Molecular Biology*, vol. 220, pp. 49–65, 1991.

[8] E. C. Uberbacher and R. J. Mural, "Locating protein-coding regions in human DNA sequences by a multiple sensor-neural network approach," in *Proc. Natl. Acad. Sci. USA*, vol. 99, 1991, pp. 11 262–11 265.

[9] E. C. Uberbacher, Y. Xu, and R.J.Mural, "Discovering and understanding genes in human DNA sequence using GRAIL," *Meth. Enzymol.*, vol. 266, pp. 259–281, 1996.

[10] M. G. Reese and F. Eeckman, "Novel Neural Network Prediction Systems for Human Promoters and Splice Sites," in *Proceedings of the Workshop on Gene-Finding and Gene Structure Prediction,Pennsylvania*, 1995.

[11] E. E. Snyder and G. D. Stormo, "Identification of protein coding regions in genomic DNA," *Journal of Molecular Biology*, vol. 248, pp. 1–18, 1995.

[12] B. Rost and C. Sander, "Prediction of protein secondary structure at better than 70% accuracy," *Journal of Molecular Biology*, vol. 232, pp. 584–599, 1993.

[13] D. T. Jones, "Protein Secondary Structure Prediction Based on Position-specific Scoring Matricws," *Journal of Molecular Biology*, vol. 292, pp. 195–202, 1999.

[14] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri, "Exploiting the past and the future in protein secondary structure prediction," *Bioinformatics*, vol. 15, no. 11, pp. 937–946, 1999.

[15] G. Pollastri, D. Przybylski, B. Rost, and P. Baldi, "Improving the Prediction of Protein Secondary Structure in Three and Egight Classes Using Recurrent Neural Networks and Profiles," *PROTEINS: Structure, Fuction, and Genetics*, vol. 47, pp. 228–235, 2002.

[16] K. Lin, V. A. Simossis, W. R. Taylor, and J. Heringa, "A simple and fast secondary structure prodiction method using hidden neural networks," *Bioinformatics*, vol. 21, no. 2, pp. 152–159, 2005.

[17] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," in *Proceeding of IEEE*, vol. 77, no. 2, 1989, pp. 257–286.

[18] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis. Cambridge.* Cambridge University Press, 1998.

[19] A. Bateman, E. Birney, R. Durbin, S. Eddy, K. Howe, and E. Sonnhammer, "The Pfam Protein Families Database," *Nucleic Acids Res.*, vol. 28, no. 1, pp. 263–266, 2002.

[20] A. Krogh, M. Brown, I. Mian, S. Sjolander, and D. Haussler, "Hidden Markov models in computational biology. Applications to protein modeling," *Journal of Molecular Biology*, vol. 235, no. 5, pp. 1501–1531, 1994.

[21] A. Krogh, B. Larsson, G. von Heijne, and E. Sonnhammer, "Predicting transmembrane protein topology with a hidden Markov model: Application to complete genomes," *Journal of Molecular Biology*, vol. 305, no. 3, pp. 567–580, 2001.

[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley (Reading, Mass), 1989.

[23] X. Yao, "Evolving Artificial Neural Networks," in *Proceedings of the IEEE*, vol. 87, no. 9, 1999, pp. 1423–1447.

[24] T. Yada, M. Ishikawa, H. Tanaka, and K. Asai, "DNA sequence Analysis using Hidden Markov Model and Genetic Algorithm," *Genome Informatics*, vol. 5, pp. 178–179, 1994.

[25] G. Stormo, "Consensus patterns in DNA," *Methods in Enzymology*, vol. 180, pp. 211–220, 1990.

[26] ——, "Computer methods for analyzing sequence recognition of nucleic acids," *Annu. Rev. Biophys. Biophys. Chem.*, vol. 17, pp. 241–263, 1988.

[27] O. Berg and P. von Hippel, "Selection of DNA binding sites by regulatory proteins," *Journal of Molecular Biology*, vol. 193, pp. 723–750, 1987.

[28] J. M. Claverie, "Some useful statistical properties of position-weight matrices," *Computers and Chemistry*, vol. 18, no. 3, pp. 287–294, 1994.

[29] S. Matis, Y. Xu, M. B. Shah, D. Buley, X. Guan, J. R. Einstein, R. J. Mural, and E. C. Uberbacher, "Detection of RNA Polymerase II Promoters and Polyadenylation Sites in Human DNA Sequence," *Computers and Chemistry*, vol. 20, pp. 135–140, 1996.

[30] C. M. O'Neill, "Training back-propagation neural networks to define and detect DNA-binding sites," *Nucl. Acids Res.*, vol. 19, no. 313-318, 1991.

[31] A. P. Bird, "CpG islands as gene markers in the vertebrate nucleus," *Journal of Molecular Biology*, vol. 193, pp. 723–750, 1987.

[32] F. Larsen, R. Gundersen, R. Lopez, and H. Prydz, "CpG islands as gene markers in the human genome," *Genomics*, vol. 13, pp. 1095–1107, 1992.

[33] J. Jurka, J. Walichiewicz, and A. J. Milosavljevic, "Prototypic sequences for human repetitive DNA," *J. Mol. Evol.*, vol. 35, pp. 286–291, 1992.

[34] A. Milosavljevic and J. Jurka, "Discovering simple DNA sequences by the algorithmic similarity method," in *CABIOS*, vol. 9, no. 4, 1993, pp. 407–411.

[35] S. Batzoglou, L. Pachter, J. Mesirovi, B. Berger, and E. Lander, "Human and mouse gene structure: comparative analysis and application to exon prediction," *Genome Research*, vol. 7, pp. 950–958, 2000.

[36] V. Solovyev, S. A., and C. Lawrence, "Predicting internal exons by oligonucleotide composition and discriminant analysis of splicable open reading frames," *Nucl. Acids Res.*, vol. 22, pp. 5156–5163, 1994.

[37] R. Guigo, "Computational Gene Identification: an open problem," *Computers and Chemistry*, vol. 21, no. 4, pp. 215–222, 1997.

[38] E. Birney and R. Durbin, "Using GeneWise in the Drosophila annotation experiment," *Genome Research*, vol. 10, no. 4, pp. 547–548, 2000.

[39] D. Kulp, D. Haussler, M. Reese, and F. Eeckman, "A generalized hidden Markov model for the recognition of human genes in DNA," in *ISMB*, 1996, pp. 134–142.

[40] C. Burge and S. Karlin, "Prediction of complete gene structures in human genomic DNA," *Journal of Molecular Biology*, vol. 268, pp. 78–94, 1997.

[41] V. Solovyev, A. Salamov, and C. B. Lawrence, "Identification of human gene structure using linear discriminat functions and dynamic programming," in *Proc. of Third Int. Conf. on Intelligent Systems for Molecular Biology*, vol. 3, 1995, pp. 367–375.

[42] A. Krogh, "Two methods for improving performance of a HMM and their application for gene finding," in *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, 1997, pp. 179–186.

[43] W. Kabsch and C. Sander, "Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features," *Biopolymers*, vol. 22, pp. 2577–2637, 1983.

[44] A. Zemla, C. Vencovas, K. Fidelis, and B. Rost, "A Modified Definition of Sov, a Segment-Based Measure for Protein Secondary Structure Prediction Assessment," *Proteins*, vol. 34, pp. 220–223, 1999.

[45] V. I. Lim, "Algorithms for prediction of alpha helices and structural regions in globular proteins," *Journal of Molecualr Biology*, vol. 88, pp. 873–894, 1974.

[46] P. Y. Chow and G. D. Fasman, "Prediction of the secondary structure of proteins from their amino acid sequence," *Advances in Enzymology*, vol. 47, pp. 45–148, 1978.

[47] J. Garnier, D. J. Osguthorpe, and B. Robson, "Analysis and implications of simple methods for predicting the secondary structure of globular proteins," *Journal of Molecular Biology*, vol. 120, pp. 97–120, 1978.

[48] S. A. Benner and D. Gerloff, "Patterns of divergence in homologous protins as indicators of secondary and tertiary structure: a prediction of the structure of the catalytic domain of protein kinases," *Advan. Enzyme Reg.*, vol. 31, pp. 121–181, 1990.

[49] N. Qian and T. J. Sejnowski, "Predicting the secondary structure of globular proteins using neural network models," *Journal of Molecular Biology*, vol. 202, pp. 865–884, 1988.

[50] H. Bohr, J. Bohr, S. Brunak, R. Cotterill, B. Lautrup, L. Nørskov, O. Olsen, and S. Petersen, "Predicting the secondary structure of globular proteins using neural network models," *Journal of Molecular Biology*, vol. 202, pp. 865–884, 1988.

[51] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucl. Acids Res.*, vol. 24, pp. 3389–3402, 1997.

[52] S. K. Riis and A. Krogh, "Improving Prediction of Protein Secondary Structure using Structured Neural Networks and Multiple Sequence Alignments," *Journal of Computational Biology*, vol. 3, pp. 163–183, 1996.

[53] S. Hua and Z. Sun, "A Novel Method of Protein Secondary Structure Prediction with High Segment Overlab Measure: Support Vector Machine Approach," *Journal of Molecular Biology*, vol. 308, pp. 397–407, 2001.

[54] J. J. Ward, L. J. McGuffin, B. F. Buxton, and D. T. Jones, "Secondary structure prediction with support vector machines," *Bioinformatics*, vol. 19, no. 13, pp. 1650–1655, 2003.

[55] J. Guo, H. Chen, Z. Sun, and Y. Lin, "A Novel Method for Protein Secondary Structure Prediction Using Dual-Layer SVM and Profiles," *PROTEINS: Structure, Function, and Bioinformatics*, vol. 54, pp. 738–743, 2004.

[56] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403–410, 1990.

[57] R. Leinonen, F. Diez, W. Fleischmann, R. Lopez, and R. Apweiler, "Uniprot archive," *Bioinformatics*, vol. 20, no. 17, pp. 3236–3237, 2004.

[58] S. Henikoff and J. Henikoff, "Amino acid substitution matrices from protein blocks," *Proc. Natl. Acad. Sci. USA*, vol. 89, no. 22, pp. 10 915–10 919, 1992.

[59] ——, "Position-based Sequence Weights," *Journal of Molecular Biology*, vol. 243, pp. 574–578, 1994.

[60] S. Kirkpatrick, "Optimization by simulated annealing: quantitative studies," *Statistical Physics*, vol. 34, pp. 975–986, 1984.

[61] A. Prügel-Bennett and J. L. Shapiro, "The dynamics of a genetic algorithm for simple random Ising systems," *Physica D*, vol. 104, pp. 75–114, 1997.

[62] G. Syswerda, "Simulated crossover in genetic algorithms," in *Proceedings of the Second International Conference on Genetic Algorithms*, 1987.

[63] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1999.

[64] P. Galinier and J. K. Hao, "Hybrid evolutionary algorithms for graph coloring," *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 379–397, 1999.

[65] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates (Hillsdale), 1987.

[66] A. Prügel-Bennett and J. L. Shapiro, "An analysis of genetic algorithms using statistical mechanics," *Physical Review Letters*, vol. 72, no. 9, pp. 1305–1309, 1994.

[67] J. P. Cohoon, S. U. Hedge, W. N. Martin, and D. Richard, "Punctuated equilibria: A parallel genetic alrotithm," in *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates, 1987, p. 148.

[68] B. Manderick and P. Spiessens, "Fine-grained parallel genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, 1989, p. 428.

[69] L. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains." *The Annals of Mathematical Statistics*, vol. 41, pp. 164–171, 1970.

[70] K. Sjolander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I. S. Mian, and D. Haussler, "Dirichlet mixture: A method for improved detection of weak but significant proten sequence homology," in *CABIOS*, vol. 12, 1996, pp. 327–345.

[71] A. Krogh and S. K. Riis, "Hidden Neural Networks," *Neural Computation*, vol. 11, pp. 541–563, 1999.

[72] P. Fariselli, P. Martelli, and R. Casadio, "The posterior-Viterbi: a new decoding algorithm for hidden Markov models," *Quantitative Biology, "http://arxiv.org/abs/q-bio.BM/0501006"*, 2005.

[73] L. Käll, A. Krogh, and E. Sonnhammer, "An HMM posterior decoder for sequence feature prediction that includes homology information," *Bioinformatics*, vol. 21, pp. 251–257, 2005.

[74] B. Juang and L. Rabiner, "Hidden Markov models for speech recognition," *Technometrics*, vol. 33, no. 3, pp. 251–272, 1991.

[75] P. Gopalakrishnan, D. Kanevsky, A. Nádas, and D. Nahamoo, "An inequality for rational functions with applications to some statistical estimation problem," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 107–113, 1991.

[76] Y. Normandin and S. Morgera, "An improved MMIE training algorithm for speaker independent, small vocabulary, continuous speech recognition," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1991, pp. 537–540.

[77] Y. Normandin, R. Cardin, and R. Mori, "High-performance connected digit recognition using maximum mutual information estimation," *IEEE Transactions on Speech and Audio Processing*, vol. 2, pp. 229–311, 1994.

[78] S. K. Riis, "Hidden Markov Models and Neural Networks for Speech Recognition," Ph.D. dissertation, Technical University of Denmark, 1998.

[79] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Transactions on Pattern Anal. Machine Intell.*, vol. 5, pp. 179–190, 1983.

[80] A. Stolcke, "Bayesian Learning of Probabilistic Language Models," Ph.D. dissertation, University of California at Berkeley, 1994.

[81] Y. Fujiwara, M. Asogawa, and A. Konagaya, "Motif Extraction using an Improved Iterative Duplication Method for HMM Topology Learning," in *In Pacific Symposium on Biocumputing '96*, 1995, pp. 713–714.

[82] C. W. S. K. Chau, C. K. Diu, and W. R. Fahrner, "Optimization of HMM by a Genetic Algorithm," in *International Conference on Aucoustics, Speech and Signal Processing*, 1997, pp. 1727–1730.

[83] S. Kwong, C. Chau, K. Man, and K. Tang, "Optimisation of HMM topology and its model parameters by genetic algorithms," *Pattern recognition*, vol. 34, pp. 509–522, 2001.

[84] K. Murphy and S. Mian, "Modelling Gene Expression Data using Dynamic Bayesian Networks," Computer Science Division, University of California, Berkeley, CA, Tech. Rep., 1991.

[85] C. R. Friedman, J. Neimann, H. C. Wegener, and R. V. Tauxe, *Epidemiology of Campylobacter jejuni infection in the United States and other industrialized nations.* ASM Press, Washington, DC., 2000, pp. 121–139.

[86] J. M. Berg, J. L. Tymoczko, and L. Stryer, *Biochemstry*, 5th ed. Michelle Julet, 2002.

[87] M. M. Wosten, M. Boeve, M. G. Koot, A. C. van Nuene, and B. A. van der Zeijst, "Identification of *Campylobacter jejuni* promoter sequence," *J. Bacteriaol.*, vol. 180, pp. 594–599, 1998.

[88] L. Petersen, T. S. Larsen, D. W. Ussery, S. L. W. On, and A. Krogh, "*RpoD* promoters in *Campylobacter jejuni* exhibit a strong periodic signal instead of a -35 box," *Journal of Molecular Biology*, vol. 326, no. 5, pp. 1361–1372, 2003.

[89] T. M. Mitchell, *Machine Learning.* McGraw-Hill Companies, Inc., 1997.

[90] A. Krogh, *An introduction to hidden Markov models for biological sequences*, ser. Computational Methods in Molecular Biology. Amsterdam: Elsevier, 1998, ch. 4, pp. 45–63.

[91] B. Rost and V. Eyrich, "Eva: large-scale analysis of secondary structure prediction," *Proteins*, vol. 5, pp. 192–199, 2001.

[92] I. Van Walle, I. Lasters, and L. Wyns, "SABmark–a benchmark for sequence alignment that covers the entire known fold space." *Bioinformatics*, vol. 21, pp. 1267–8, 2005.

[93] T. Hamelryck and B. Manderick, "PDB file parser and structure class implemented in Python." *Bioinformatics*, vol. 19, pp. 2308–10, 2003.

[94] D. Rumelhart, G. Hinton, and R. Williams, "Learining representations by back-propagating error," *Nature*, vol. 323, pp. 533–536, 1986.

[95] L. Davis, "Bit-climing, representational bias, and test suite design," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pp. 18–23.